Polkadot Launch Phases

The Polkadot network has a phased roll-out plan, with important milestones toward decentralization marking each phase. Keep up-to-date with the Polkadot's phased roll-out plan at by viewing the roadmap

Current Phase: Parachain Rollout

The PoA Launch

The Genesis block of the Polkadot network was launched on May 26, 2020, as a Proof of Authority (PoA) network. Governance was restricted to the single Sudo (super-user) key, which was held by Web3 Foundation to issue the commands and upgrades necessary to complete the launch process. During this time, validators started joining the network and signaling their intention to participate in consensus.

Nominated Proof of Stake

Once Web3 Foundation was confident in the stability of the network and there was a sufficient number of validator intentions, Web3 Foundation used Sudo — a superuser account with access to governance functions — to initiate the first validator election. Following this election, the network transitioned from PoA into its second phase, Nominated Proof of Stake (NPoS), on June 18, 2020.

Governance

After the chain had been running well with the validator set, the Sudo key issued a runtime upgrade that enabled the suite of governance modules in Polkadot; namely, the modules to enable a Council, a Technical Committee, and public referenda.

Removal of Sudo

The Sudo module was removed by a runtime upgrade on July 20, 2020, transitioning the governance of the chain into the hands of the token (DOT) holders.

From this point, the network has been entirely in the hands of the token holders and is no longer under control of any centralized authority.

Balance Transfers

To enable balance transfers, the community made a public proposal for a runtime upgrade that lifted the restriction on balance transfers. Transfer functionality was subsequently enabled on Polkadot at block number 1_205_128 on August 18, 2020, at 16:39 UTC.

Redenomination

On August 21, 2020, redenomination of DOT, the native token on Polkadot, occurred. From this date, one DOT (old) equals 100 new DOT.

Core Functionality

Polkadot is now moving to the next stage of opening up its core functionality, like parachain slot auctions, parathreads, and cross-chain message passing. Polkadot is now on track to launch several parachains in 2021. These upgrades will require runtime upgrades that will pass through Polkadot's normal governance processes. The core functionality does not have to be unlocked sequentially — several features can be unlocked with a single proposal.

Parachains will first roll out on Kusama with a common good parachain, followed by the first slot auction and winner's onboarding.

Polkadot 2.0

Researchers are in the midst of research for the next version of the Polkadot network. With many questions yet to be answered, as of now, some big areas of research will be in:

- Economics and Networking (Zero-Knowledge): How will scalability work in Polkadot 2.0?
- Horizontal vs. Vertical scalability: What is the breaking point of the maximum number of parachains built with horizontal scalability?
- Nested Relay Chain: How can multiple Relay Chains exist connected through parachains? How many tiers of Relay Chains can be nested? How will validators work together to validate blocks on various Relay Chains? How does XCM work in the nested setup? How is AnV going to work there?

Edit this page

*Last updated on **10/6/2021** by **Danny Salman***

Architecture

Polkadot is a heterogeneous multichain with shared security and interoperability.

Components

Relay Chain

The Relay Chain is the central chain of Polkadot. All validators of Polkadot are staked on the Relay Chain in DOT and validate for the Relay Chain. The Relay Chain is composed of a relatively small number of transaction types that include ways to interact with the governance mechanism, parachain auctions, and participating in NPoS. The Relay Chain has deliberately minimal functionality - for instance, smart contracts are not supported. The main responsibility is to coordinate the system as a whole, including parachains. Other specific work is delegated to the parachains, which have different implementations and features.

Parachain and Parathread Slots

Polkadot can support a number of execution slots. These slots are like cores on a computer's processor (a modern laptop's processor may have eight cores, for example). Each one of these cores can run one process at a time. Polkadot allows these slots using two subscription models: parachains and parathreads. Parachains have a dedicated slot (core) for their chain and are like a process that runs constantly. Parathreads share slots amongst a group, and are thus more like processes that need to be woken up and run less frequently.

Most of the computation that happens across the Polkadot network as a whole will be delegated to specific parachain or parathread implementations that handle various use cases. Polkadot places no constraints over what parachains can do besides that they must be able to generate a proof that can be validated by the validators assigned to the parachain. This proof verifies the state transition of the parachain. Some parachains may be specific to a particular application, others may focus on specific features like smart contracts, privacy, or scalability — still, others might be experimental architectures that are not necessarily blockchain in nature.

Polkadot provides many ways to secure a slot for a parachain for a particular length of time. Parathreads are part of a pool that shares slots and must-win auctions for individual blocks. Parathreads and parachains have the same API; their difference is economic. Parachains will have to reserve DOT for the duration of their slot lease; parathreads will pay on a per-block basis. Parathreads can become parachains, and vice-versa.

Shared Security

Parachains connected to the Polkadot Relay Chain all share in the security of the Relay Chain. Polkadot has a shared state between the Relay Chain and all of the connected parachains. If the Relay Chain must revert for any reason, then all of the parachains would also revert. This is to ensure that the validity of the entire system can persist and no individual part is corruptible.

The shared state ensures that the trust assumptions when using Polkadot parachains are only those of the Relay Chain validator set and no other. Since the validator set on the Relay Chain is expected to be secure with a large amount of stake put up to back it, parachains should benefit from this security.

Bridges

A blockchain bridge is a connection that allows for arbitrary data to transfer from one network to another. These chains are interoperable through the bridge but can exist as standalone chains with different protocols, rules, and governance models. In Polkadot, bridges connect to the Relay Chain and are secured through the Polkadot consensus mechanism, maintained by collators.

Polkadot uses bridges to bridge the future of Web 3.0, as bridges are fundamental to Polkadot's interoperable architecture by acting as a [secure and robust] communication channel for chains in isolation.

Main Actors

Validators

Validators, if elected to the validator set, produce blocks on the Relay Chain. They also accept proofs of valid state transition from collators. In return, they will receive staking rewards.

Nominators

Nominators bond their stake to particular validators in order to help them get into the active validator set and thus produce blocks for the chain. In return, nominators are generally rewarded with a portion of the staking rewards from that validator.

Collators

Collators are full nodes on both a parachain and the Relay Chain. They collect parachain transactions and produce state transition proofs for the validators on the Relay Chain. They can also send and receive messages from other parachains using XCMP.

Whiteboard Series

For a video overview of the architecture of Polkadot watch the video below for the whiteboard interview with W3F researcher Alistair Stewart:

Polkadot Accounts

This document covers the basics of Polkadot and Kusama account addresses and how they exist on-chain. For a more in-depth explanation of the cryptography behind them, please see the cryptography page.

Address Format

The address format used in Substrate-based chains is SS58. SS58 is a modification of Base-58-check from Bitcoin with some minor changes. Notably, the format contains an *address type* prefix that identifies an address as belonging to a specific network.

For example:

- Polkadot addresses **always start with** the number **1**.
- Kusama addresses always start with a capital letter, such as **C D, F, G, H, J**.
- Generic Substrate addresses **always start with** the number **5**.

These prefixes are broken down on the Substrate GitHub Wiki page on SS58.

It's important to understand that different network formats are **merely other representations of the same public key in a private-public keypair** generated by an address generation tool. As a result, the addresses across Substrate-based chains are compatible as long as the format is converted correctly.

As of Runtime 28, the default address format is the `MultiAddress` type.

This `enum` is a multi-format address wrapper for on-chain accounts and allows us to describe Polkadot's default address format to represent many different address types. This includes **20 byte**, **32 byte**, and **arbitrary raw byte** variants. It also allows an enhancement to the original `indices` lookup.

> Many wallets allow you to convert between formats. Stand-alone tools exist as well; you can find them in the address conversion tools section.

Address Generation, Derivation, and Portability

A valid account requires a private key that can sign on to one of the supported curves and signature schemes.

Most wallets take many steps from a mnemonic phrase to an account key, which affects the ability to use the same mnemonic phrase in multiple wallets. Wallets that use different measures will arrive at a different set of addresses from the exact mnemonic phrase.

Seed Generation

Most wallets generate a mnemonic phrase for users to back up their wallets and generate a private key from the mnemonic. Not all wallets use the same algorithm to convert from mnemonic phrase to private key.

A typical mnemonic phrase generated by the Subkey tool is shown below.

```
'caution juice atom organ advance problem want pledge someone senior holiday
very'
```

Its corresponding *private/public keypair* is also shown.

```
Secret seed (Private key):
0x056a6a4e203766ffbea3146967ef25e9daf677b14dc6f6ed8919b1983c9bebbc
Public key (SS58): 5F3sa2TJAWMqDhXG6jhV4N8ko9SxwGy8TpaNS1repo5EYjQX
```

Subkey and Polkadot-JS based wallets use the BIP39 dictionary for mnemonic generation, but use the entropy byte array to generate the private key, while full BIP39 wallets (like Ledger) use 2048 rounds of PBKDF2 on the mnemonic. The same mnemonic may generate different private keys on other wallets due to the various cryptographic algorithms used.

See Substrate BIP39 Repo for more information.

Derivation Paths

If you would like to create and manage several accounts on the network using the same seed, you can use derivation paths. We can think of the derived accounts as child accounts of the root account created using the original mnemonic seed phrase. Many Polkadot key generation tools support hard and soft derivation. For instance, if you intend to create an account to be used on the Polkadot chain, you can derive a **hard key** child account using **//** after the mnemonic phrase.

```
'caution juice atom organ advance problem want pledge someone senior holiday
very//0'
```

and a **soft key** child account using **/** after the mnemonic phrase

```
'caution juice atom organ advance problem want pledge someone senior holiday
very/0'
```

If you would like to create another account for using the Polkadot chain using the same seed, you can change the number at the end of the string above. For example, `/1`, `/2`, and `/3` will create different derived accounts.

You can use any letters or numbers in the derivation path as long as they make sense to you; they do not have to follow any specific pattern. You may combine multiple derivations in your path, as well. For instance, `//bill//account//1` and `//john/polkadot/initial` are both valid. To recreate a derived account, you must know both the seed and the derivation path, so you should either use a well-defined sequence (e.g. //0, //1, //2...) or be sure to write down any derivation paths you use.

> It is not possible to generate a derived account without also knowing the derivation path.

Soft vs. Hard Derivation

A soft derivation allows someone to potentially "go backwards" to figure out the initial account's private key if they know the derived account's private key. It is also possible to determine that different accounts generated from the same seed are linked to that seed.

A hard derivation path does not allow either of these - even if you know a derived private key, it's not feasible to figure out the private key of the root address, and it's impossible to prove that the first account is linked with the second. These derivation methods have their use cases, given that the private keys for all the derived accounts are fully secure. Unless you have a specific need for a soft derivation, it is recommended to generate the account using a hard derivation path.

See the Subkey documentation for details and examples of derivation path formats. The Polkadot-JS Apps and Extension and Parity Signer support custom derivation paths using the same syntax as Subkey.

Some wallets will automatically add derivation paths to the end of the generated mnemonic phrase. This will generate separate seeds for separate paths, allowing separate signing keys with the same mnemonic, e.g. `<mnemonic phrase>//polkadot` and `<mnemonic phrase>//kusama`. Although you may correctly save the mnemonic phrase, using it in another wallet will not generate the same addresses unless both wallets use the same derivation paths.

Polkadot and Kusama both have paths registered in the BIP44 registry.

> **Warning:** You must have both the *parent* private key and the derivation path to arrive at the key for an address. Do not use custom derivation paths unless you are comfortable with your understanding of this topic.

Portability

The above information brings us to portability: the ability to use a mnemonic phrase or seed across multiple wallets. Portability depends on several factors:

- Derivation path
- Mnemonic format
- Seed derivation
- Signature scheme

If you want to use the exact mnemonic across multiple wallets, make sure that they follow compatible methods for generating keys and signing messages. If you cannot find understandable documentation, reach out to the project maintainers.

	Mnemonic Format	**Derivation Path**	**Seed Derivation**	**Signature Support**
Polkadot{.js} Extension	Standard	User-Defined	BIP32	sr25519
Polkadot-JS Apps	Standard*	User-Defined	BIP32	sr25519, ed25519, secp256k
Ledger	BIP39	BIP44†	BIP32‡	ed25519§
Subkey	Standard*	User-Defined	BIP32	sr25519, ed25519, secp256k1

* Ed25519 keys have limited compatibility with BIP39.

† BIP44 Registry

‡ Ed25519 and BIP32 based on Khovratovich

§ Sr25519 planned

For the Curious: How Prefixes Work

The SS58 document states that:

- Polkadot has an address type of `00000000b`, so `0` is in decimal.
- Kusama (Polkadot Canary) has an address type of `00000010b`, so `2` is in decimal.
- Generic Substrate has `00101010b` as the address type, `42` is in decimal.

Because the `Base58-check` alphabet has no number 0, the lowest value is indeed 1. So `00000000b` is 1 in Base58-check. If we try to decode a Polkadot address like `1FRMM8PEiWXYax7rpS6X4XZX1aAAxSWx1CrKTyrVYhV24fg`, the result is `000aff6865635ae11013a83835c019d44ec3f865145943f487ae82a8e7bed3a66b29d7`. The first byte is `00`, which is indeed `00000000` in binary and `0` in decimal and thus matches the address type of Polkadot.

Let's take a look at Substrate addresses. If we decode `5CK8D1sKNwF473wbuBP6NuhQfPaWUetNsWUNAAzVwTfxqjfr`, we get `2a0aff6865635ae11013a83835c019d44ec3f865145943f487ae82a8e7bed3a66b77e5`. The first byte is `2a` which when converted from hex to decimal is 42. 42 is `00101010` in binary, just as the SS58 document states.

Finally, let's look at Kusama addresses. Decoding `CpjsLDC1JFyrhm3ftC9Gs4QoyrkHKhZKtK7YqGTRFtTafgp` gives us `020aff6865635ae11013a83835c019d44ec3f865145943f487ae82a8e7bed3a66b0985` with the first byte being `02`, just as specified. If we try a Kusama address that starts with a completely different letter, like `J4iggBtsWsb61RemU2TDWDXTNHqHNfBSAkGvVZBtn1AJV1a`, we still get `02` as the first byte: `02f2d606a67f58fa0b3ad2b556195a0ef905676efd4e3ec62f8fa1b8461355f1142509`. It seems counterintuitive that some addresses always have the same prefix and others like Kusama can vary wildly, but it's just a quirk of Base58-check encoding.

Obtaining and Managing an Address

The **most user-friendly** way to create a Polkadot or Kusama address is through the Polkadot-JS UI. Remember to back up the seed phrase used to generate your account - the accounts are stored only in your browser, so purging the cache will wipe your accounts as well. You would then have to recreate them using the seed phrase given to you by the UI - this will also restore all your previously held balances.

A **more convenient and recommended** method of keeping the accounts stored on your computer is using the Polkadot{.js} extension. This extension remembers your accounts and allows you to clear your browser cache without fear. Still, don't forget to back up your seed phrase - if you lose access to this computer or the extension somehow crashes beyond repair, the phrase will come in handy.

Please note that as this keeps your accounts in the browser, it is not safe to keep significant holdings. By definition, a browser is a "hot wallet" and susceptible to a wide range of attacks, so keep your funds in cold storage when dealing with non-trivial amounts. For improved security, you can securely stash away the seed phrase for your accounts and remove all traces of the accounts from your computer after creating them.

Besides the extension and the default UI, Polkadot and Kusama addresses can also be created with the Subkey tool. Subkey is intended for users comfortable with using the command line and can seem intimidating but is quite approachable. Follow the instructions in the Subkey documentation. When used properly, Subkey is the **most secure** available method of creating an account.

There is also the very secure Parity Signer. This keeps your keys on an air-gapped mobile phone. However, it does require obtaining an old Android or iOS-compatible phone that you are comfortable using only for Parity Signer.

Hardware wallet integration is possible with Ledger. A full guide is available here.

Alternatively, you might find other wallets on the Wallet page, but bear in mind that some of these are **unaudited** and are not officially affiliated with Web3 Foundation or the Polkadot project unless otherwise stated.

Balance Types

On Polkadot, **four different balance types** indicate whether your balance can be used for transfers, to pay fees, or must remain frozen and unused due to an on-chain requirement.

The `AccountData` struct defines the balance types in Substrate. The four types of balances include `free`, `reserved`, `misc_frozen` (`miscFrozen` in camel-case), and `fee_frozen` (`feeFrozen` in camel-case).

In general, the **usable** balance of the account is the amount that is `free` minus any funds that are considered frozen (either `misc_frozen` or `fee_frozen`) and depend on the reason for which the funds are to be used. If the funds are to be used for transfers, then the usable amount is the *free* amount minus any `misc_frozen` funds. However, if the funds are to be used to pay transaction fees, the usable amount would be the *free* funds minus `fee_frozen`.

The **total** balance of the account is considered to be the sum of `free` and `reserved` funds in the account. Reserved funds are held due to on-chain requirements and can usually be freed by taking some on-chain action. For example, the "Identity" pallet reserves funds while an on-chain identity is registered, but by clearing the identity, you can unreserve the funds and make them free again.

Existential Deposit and Reaping

When you generate an account (address), you only generate a *key* that lets you access it. The account does not exist yet on-chain. For that, it needs the existential deposit: 0.0000333333 KSM (on Kusama) or 1 DOT (on Polkadot mainnet).

Having an account go below the existential deposit causes that account to be *reaped*. The account will be wiped from the blockchain's state to conserve space, along with any funds in that address. You do not lose access to the reaped address - as long as you have your private key or recovery phrase, you can still use the address - but it needs a top-up of another existential deposit to be able to interact with the chain.

Transaction fees cannot cause an account to be reaped. Since fees are deducted from the account before any other transaction logic, accounts with balances *equal to* the existential deposit cannot construct a valid transaction. Additional funds will need to be added to cover the transaction fees.

Here's another way to think about existential deposits. Ever notice those `Thumbs.db` files on Windows or `.DS_Store` files on Mac? Those are junk; they serve no specific purpose other than making previews a bit faster. If a folder is empty saved for such a file, you can remove the folder to clear the junk off your hard drive. That does not mean you lose access to this folder forever - you can always recreate it. You have the *key*, after all - you're the computer's owner. It just means you want to keep your computer clean until you maybe end up needing this folder again and then recreate it. Your address is like this folder - it gets removed from the chain when nothing is in it but gets put back when it has the existential deposit.

Indices

A Kusama or Polkadot address can have an index. An index is like a short and easy-to-remember version of an address. Claiming an index requires a deposit that is released when the index is cleared.

Indices are populated in order. Think of them like slots going from 0 to any arbitrary number:

`[0][1][2][3][4][5][6]...`

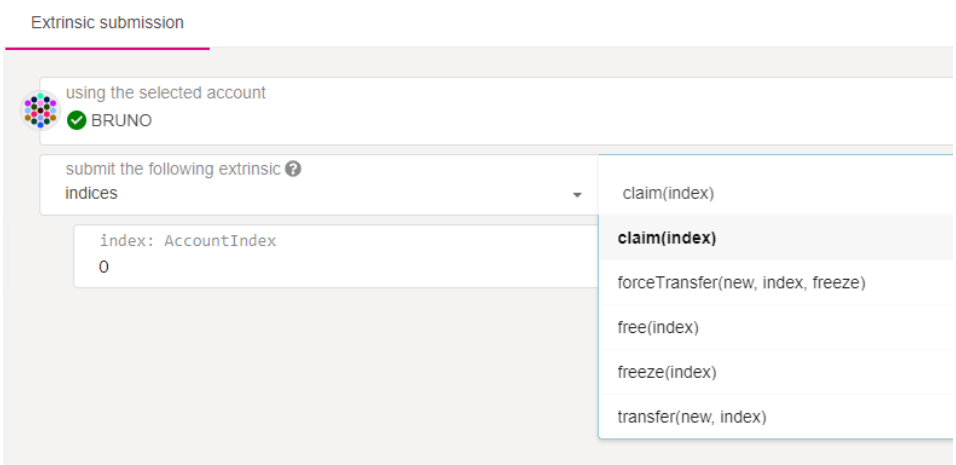
If slots 0-2 are populated by addresses A, B, and C, respectively, and I add an existential deposit to address X, that address will automatically be put into slot 3. Henceforth, you can send me money by just sending to `[3]` rather than remembering my complete address.

```
[0][1][2][3][4][5][6]...
[A][B][C][X][ ][ ][ ]...
```

But what if an account gets reaped as explained above? In that case, the index is emptied. In other words, the slot frees up again. If someone creates a new account, they may use the same index another address was using before.

It is possible to *freeze* an index and permanently assign it to an address. This action consumes a deposit but makes sure that the index can never be reclaimed unless released by the holding account.

To register an index, submit a `claim` extrinsic to the `indices` pallet, and follow up with a `freeze` extrinsic. The easiest way to do this is via PolkadotJS UI through the *Developer -> Extrinsics* menu:



To find available indices to claim, this helper tool may come in handy.

Identities

The *Identities pallet* built into Polkadot allows users to attach on-chain metadata to their accounts. Independent registrars can verify this metadata to provide trustworthiness. To learn more about how to set or release an identity, how to define sub-accounts, or how to become a registrar, please read this guide.

Proxy Accounts

Polkadot comes with a generalized proxy account system that allows users to keep keys in cold storage while proxies act on their behalf with restricted (or unrestricted) functionality. See the proxies page for more information.

Multi-signature Accounts

It is possible to create a multi-signature account in Substrate-based chains. A multi-signature account is composed of one or more addresses and a threshold. The threshold defines how many signatories (participating addresses) need to agree on submitting an extrinsic for the call to be successful.

For example, Alice, Bob, and Charlie set up a multi-sig with a threshold of 2. This means Alice and Bob can execute any call even if Charlie disagrees with it. Likewise, Charlie and Bob can execute any call without Alice. A threshold is typically a number smaller than the total number of members but can also be equal to it, which means they all have to agree.

> Learn more about multisig accounts from our technical explainer video.

Multi-signature accounts have several uses:

- securing your own stash: use additional signatories as a 2FA mechanism to secure your funds. One signer can be on one computer, and another can be on another or in cold storage. This slows down your interactions with the chain but is orders of magnitude more secure.
- board decisions: legal entities such as businesses and foundations use multi-sigs to govern over the entity's treasury collectively.
- group participation in governance: a multi-sig account can do everything a regular account can. A multi-sig account could be a council member in Kusama's governance, where a set of community members could vote as one entity.

Multi-signature accounts **cannot be modified after being created**. Changing the set of members or altering the threshold is not possible and instead requires the dissolution of the current multi-sig and creation of a new one. As such, multi-sig account addresses are **deterministic**, i.e. you can always calculate the address of a multi-sig by knowing the members and the threshold, without the account existing yet. This means one can send tokens to an address that does not exist yet, and if the entities designated as the recipients come together in a new multi-sig under a matching threshold, they will immediately have access to these tokens.

Generating Addresses of Multi-signature Accounts

> NOTE: Addresses that are provided to the multi-sig wallets must be sorted. The below methods for generating sort the accounts for you, but if you are implementing your own sorting, then be aware that the public keys are compared byte-for-byte and sorted ascending before being inserted in the payload that is hashed.

Addresses are deterministically generated from the signers and threshold of the multisig wallet. For a code example (in TypeScript) of generating you can view the internals of `@w3f/msig-util` here.

The `@w3f/msig-util` is a small CLI tool that can determine the multi-signature address based on your inputs.

```
$ npx @w3f/msig-util@1.0.7 derive --addresses
15o5762QE4UPrUaYcM83HERK7Wzbmgcsxa93NJjkHGH1unvr,1TMxLj56NtRg3scE7rRo8H9GZJMFXdsJk1
--threshold 1
npx: installed 79 in 7.764s
--------------------------------
Addresses: 15o5762QE4UPrUaYcM83HERK7Wzbmgcsxa93NJjkHGH1unvr
1TMxLj56NtRg3scE7rRo8H9GZJMFXdsJk1GyxCuTRAxTTzU
Threshold: 1
Multisig Address (SS58: 0): 15FKUKXC6kwaXxJ1tXNywmFy4ZY6FoDFCnU3fMbibFdeqwGw
--------------------------------
```

The Polkadot-JS Apps UI also supports multi-sig accounts, as documented in the Account Generation page. This is easier than generating them manually.

Making Transactions with a Multi-signature Account

There are three types of actions you can take with a multi-sig account:

- Executing a call.
- Approving a call.
- Cancelling a call.

In scenarios where only a single approval is needed, a convenience method `as_multi_threshold_1` should be used. This function takes only the other signatories and the raw call as its arguments.

However, in anything but the simple one approval case, you will likely need more than one of the signatories to approve the call before finally executing it. When you create a new call or approve a call as a multi-sig, you will need to place a small deposit. The deposit stays locked in the pallet until the call is executed. The deposit is to establish an economic cost on the storage space that the multi-sig call takes up on the chain and discourage users from creating dangling multi-sig operations that never get executed. The deposit will be reserved in the caller's accounts, so participants in multi-signature wallets should have spare funds available.

The deposit is dependent on the `threshold` parameter and is calculated as follows:

```
Deposit = DepositBase + threshold * DepositFactor
```

Where `DepositBase` and `DepositFactor` are chain constants set in the runtime code.

Currently, the **DepositBase** equals `deposit(1, 88)` (key size is 32; value is size 4+4+16+32 = 56 bytes) and the **DepositFactor** equals `deposit(0, 32)` (additional address of 32 bytes).

The deposit function in JavaScript is defined below, cribbed from the Rust source.

```
// Polkadot
const DOLLARS = 10000000000; // planck
const MILLICENTS = 100000; // planck

// Kusama
// const DOLLARS = 1666666666666.67;
// const MILLICENTS = 1666666.66;

const deposit = (items, bytes) => {
  return items * 20 * DOLLARS + bytes * 100 * MILLICENTS;
};

console.log('DepositBase', deposit(1, 88));
console.log('DepositFactor', deposit(0, 32));
```

Thus the deposit values can be calculated as shown in the table below.

	Polkadot (DOT)	**Kusama (KSM)**	**Polkadot (planck)**	**Kusama (planck)**
DepositBase	20.088	3.3401	200880000000	3340100000000
DepositFactor	.032	0.005333333312	320000000	5333333312

Let's consider an example of a multi-sig on Polkadot with a threshold of 2 and 3 signers: Alice, Bob, and Charlie. First, Alice will create the call on-chain by calling `as_multi` with the raw call. When doing this Alice will have to deposit `DepositBase + (2 * DepositFactor) = 20.152 DOT` while she waits for either Bob or Charlie also to approve the call. When Bob comes to approve the call and execute the transaction, he will not need to place the deposit, and Alice will receive her deposit back.

Example with Polkadot.JS

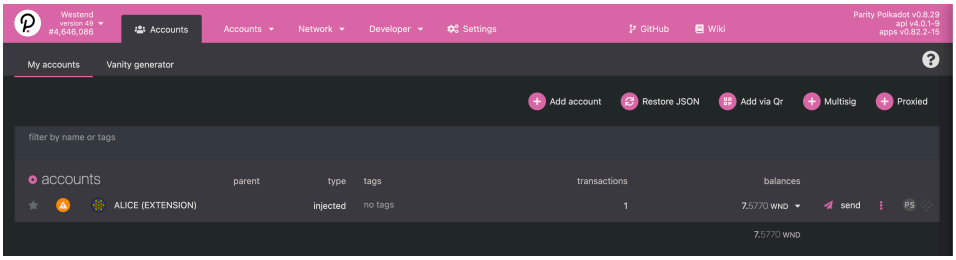
For this example, we will be using the Westend testnet and Polkadot.JS Apps to create a 2-of-3 multisig address and send a transaction with it.

> While Westend is meant to replicate the Polkadot mainnet as closely as possible, there are a few notable differences:
>
> - Existential deposit is equal to 0.01 WND (Westies; Westend's native coin) instead of 1 DOT.
> - The multi-signature transaction deposit is equal to ~1 WND instead of ~20.2 DOT.
>
> The photos below reflect values in WND, but instructions are the same for DOT.

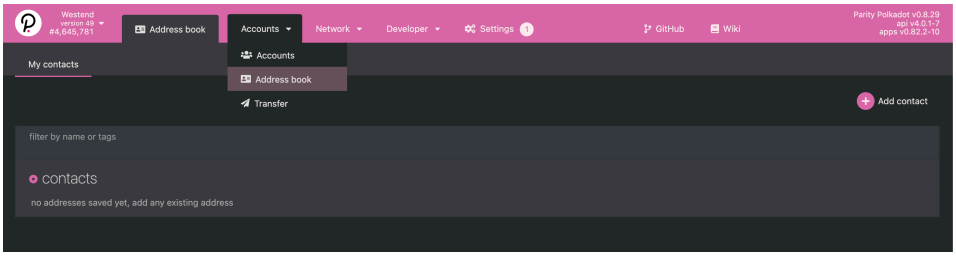
To create a multisig address and send a transaction using it, you will need the following:

- List of the multisig member's addresses. We will use Alice, Bob, and Charlie.
- This must be in the address that initiates a multi-signature transaction (in this example, Alice).- DOT to deposit into the multisig address.
- ~20.2 DOT refundable deposit to send a multisig transaction.

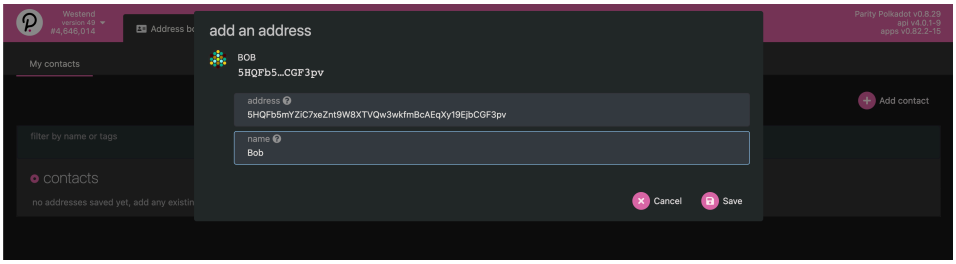
You should already have your account with some coins in it.



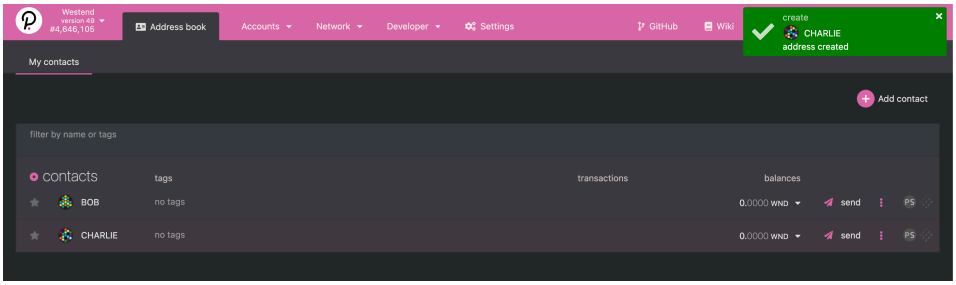
To generate the multisig address, we need **to add the multisig member addresses to the contact book** under "Accounts > Address book".



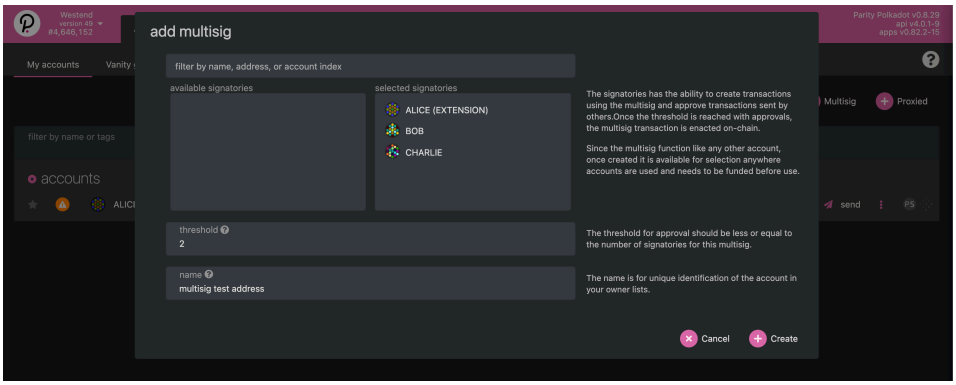
Click "Add Contact" in the upper right and provide the address and a name for each address.



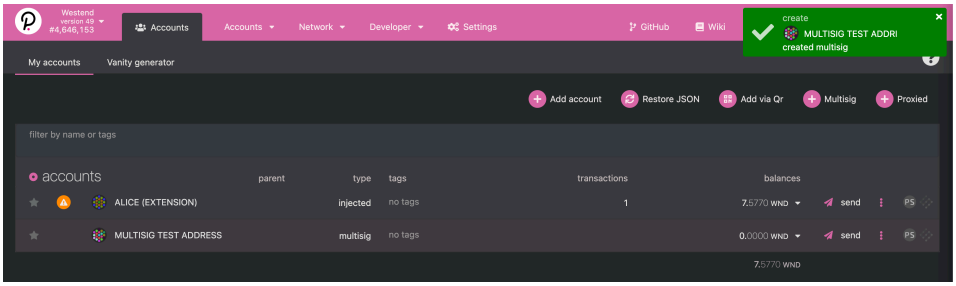
Here, Bob and Charlie have been added.



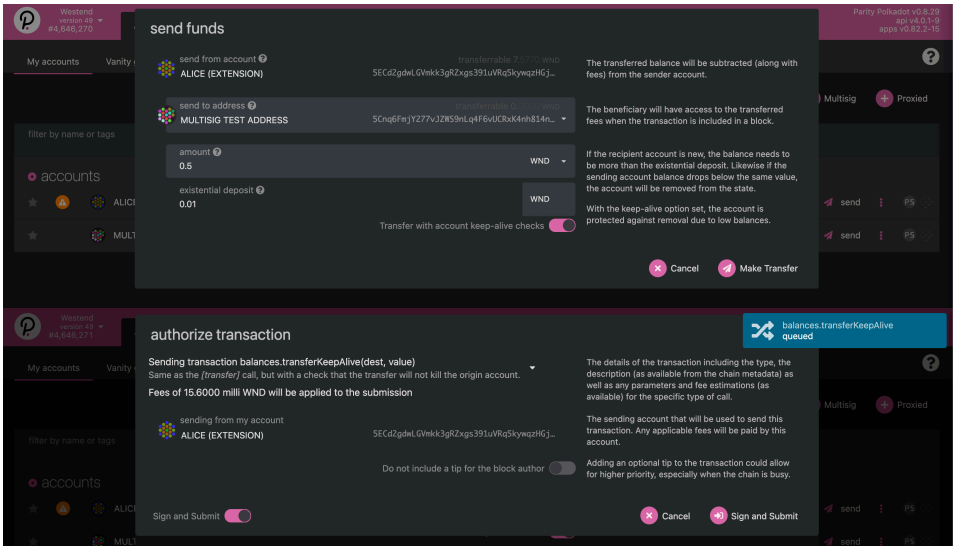
Next, we need to create the new multi-signature address. Navigate to the Accounts page (from the toolbar, "Accounts > Accounts") and click the "+ Multisig" button. We will supply the three multisig member addresses with a value '2' for the threshold.



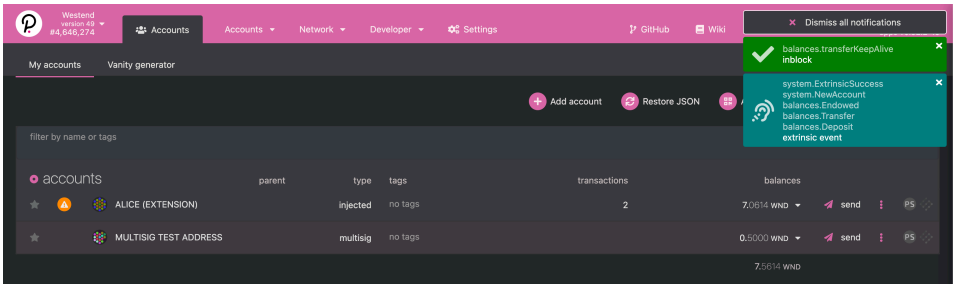
Click 'Create', and you should see the new multisig address appear on this Accounts page.



Let's fund the address now. For this example, we will transfer some coins from Alice's account to the multisig address. Under Alice's address, click 'Send', select the multisig wallet as the destination, and provide an amount. Then, click 'Make Transfer', and then 'Sign and Submit'.

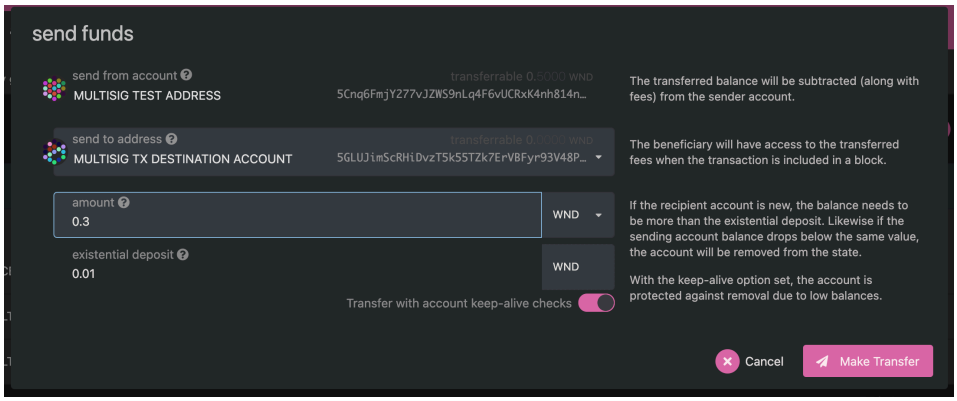


We can see that the multisig account now has a balance.

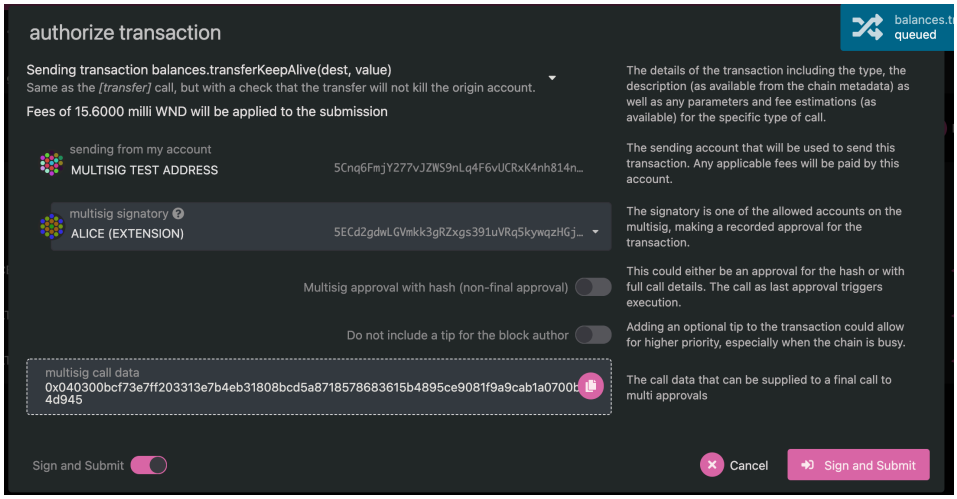


To send a transaction, we need one of the members to initiate it. Let's use Alice to initiate the transaction.

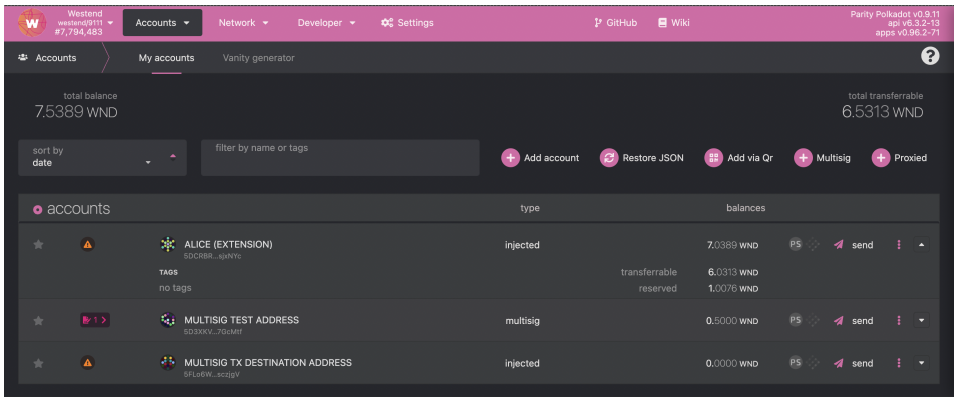
Make sure Alice has enough coins to cover the multisig transaction deposit and the transaction fees. Then, click 'Send' under the "Multisig Test Address", select a destination address (we generated an address locally) and a transfer amount, and click 'Make Transfer'.



To sign as Alice, make sure she is selected as the 'multisig signatory', click 'Sign and Submit', and sign the transaction.



You will now see a pending transaction the 'Multisig Test Address' (the purple icon next to the account name), and if you click the dropdown arrow right of Alice's balance - you will see that a value equivalent to the multisig deposit has been 'reserved'; rendering that value untransferable until the multisig transaction completes.

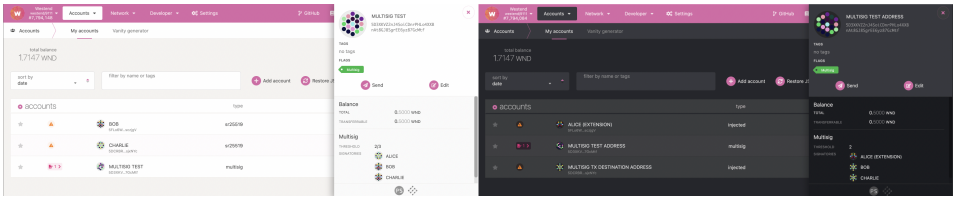


Next, we need a second signature. Let's get it from Bob. In Bob's browser, repeat the following from the above steps.

1. Add Alice, Charlie, and the multisig transaction destination addresses to Bob's Address book.
2. Create a new multisig address with the same parameters (Bob, Alice, and Charlie's addresses, and a threshold value of '2').

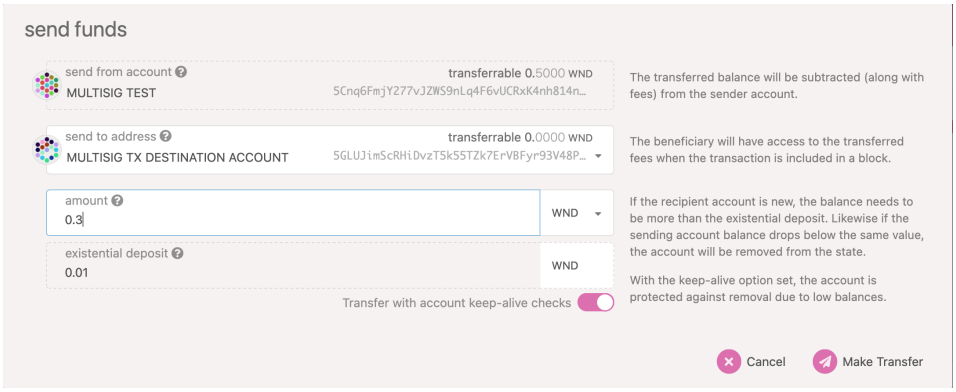
> NOTE: Since multisig address generation is deterministic, if Bob (or any other member), on his computer was to generate a multisig address using Alice's, Charlie's, and his addresses, with a threshold value of '2', he would produce the **same** multisig address that Alice has here.

If done correctly, we should see that the **same** multisig address is produced in Bob's browser, and that a pending transaction is displayed, too.



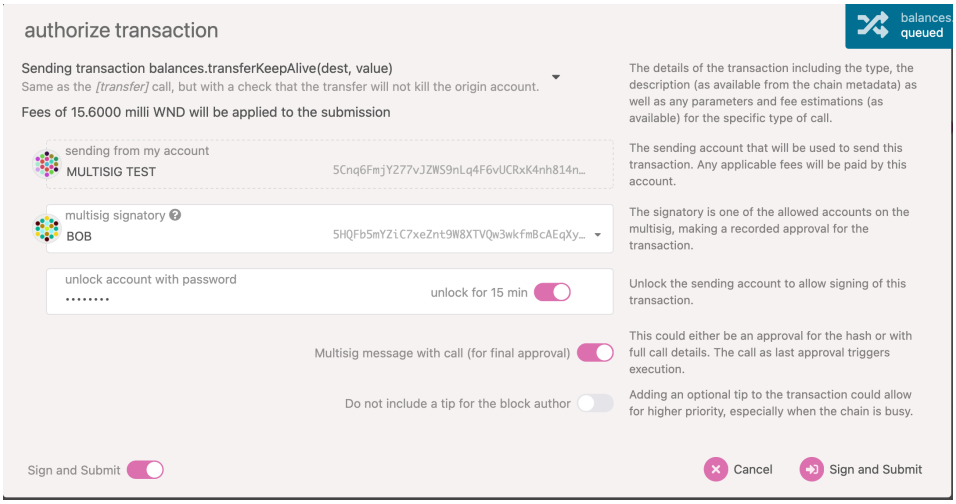
> Alice initiated the transaction by uploading a signature of the hash of the transaction and the hash.

Next, to get Bob's signature, he must craft the same multisig transaction that Alice did by providing the same destination address and transfer amount (together, transaction parameters), signing and submitting it. These transaction parameters will allow Bob to produce and sign the same transaction (the same hash) that Alice signed earlier.

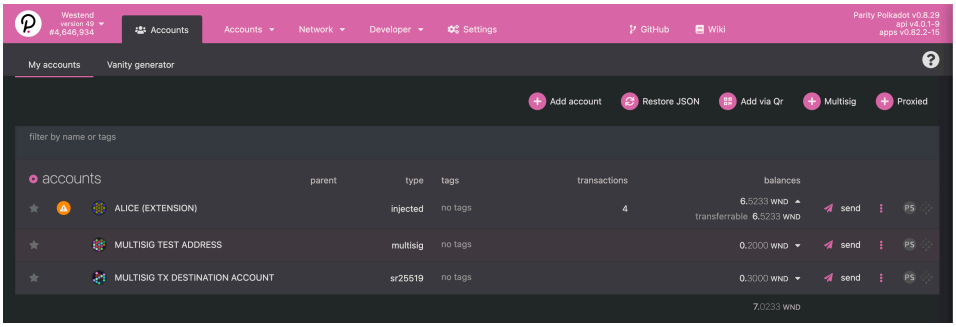


Click 'Make Transfer' - ensure that Bob is the 'multisig signatory', and click 'Sign and Submit'.

> NOTE: 'Multisig message with the call (for final approval)' is automatically enabled; this means that, since the transaction will reach the signature threshold, it will execute the actual transaction on the chain after adding the second signature.



Assuming no errors, 'Multisig Destination Account' has a balance of 0.3 WND, and Alice's account has released the multisig transaction deposit.



Address Conversion Tools

You can use the tools below to convert any SS58 address for any network for use on different networks

- handy subscan tool
- simple address converter

How to Verify a Public Key's Associated Address

You can verify your public key's associated address through a series of inspection steps, where the key is a base-16 (hexadecimal) address.

Using Subkey to Retrieve Public Key from SS58 Address

This is to showcase that the **SS58 address is based on the public key (aka "Account ID")**

The Subkey Tool's The Inspecting Keys section explains how to use the `inspect` command to recalculate your key pair's public key and address.

Start by inspecting your account's Polkadot address by running the inspect command against your account's address:

```
$ subkey inspect 1a1LcBX6hGPKg5aQ6DXZpAHCCzWjckhea4sz3P1PvL3oc4F

Public Key URI `1a1LcBX6hGPKg5aQ6DXZpAHCCzWjckhea4sz3P1PvL3oc4F` is account:
  Network ID/version: polkadot
  Public key (hex):
0x192c3c7e5789b461fbf1c7f614ba5eed0b22efc507cda60a5e7fda8e046bcdce
  Account ID:
0x192c3c7e5789b461fbf1c7f614ba5eed0b22efc507cda60a5e7fda8e046bcdce
  SS58 Address:       1a1LcBX6hGPKg5aQ6DXZpAHCCzWjckhea4sz3P1PvL3oc4F
```

Take note of the hexadecimal string for "Public key (hex)". This is your account's public key.

Running the `inspect` command on your public key, the default SS58 address that is returned the associated Substrate address.

```
$ subkey inspect
0x192c3c7e5789b461fbf1c7f614ba5eed0b22efc507cda60a5e7fda8e046bcdce

Secret Key URI
`0x192c3c7e5789b461fbf1c7f614ba5eed0b22efc507cda60a5e7fda8e046bcdce` is account:
  Secret seed:
0x192c3c7e5789b461fbf1c7f614ba5eed0b22efc507cda60a5e7fda8e046bcdce
  Public key (hex):
0x5a095388156e3a68d4fb7cbaef981478e1a0be6d4998f00dffc3e4e9c60c104c
  Account ID: 0x5a095388156e3a68d4fb7cbaef981478e1a0be6d4998f00dffc3e4e9c60c104c
  SS58 Address: 5E6kwKEhrpVMnZvkBRFCzCcRnMXcft4HSaogYQtgtaw6QJ5s
```

Using the `--network` flag, you can define the network that you would like to inspect, where the SS58 address will be based on that network. Now, running the `inspect` command with `--network polkadot` return your original Polkadot address, thus verifying the public key.

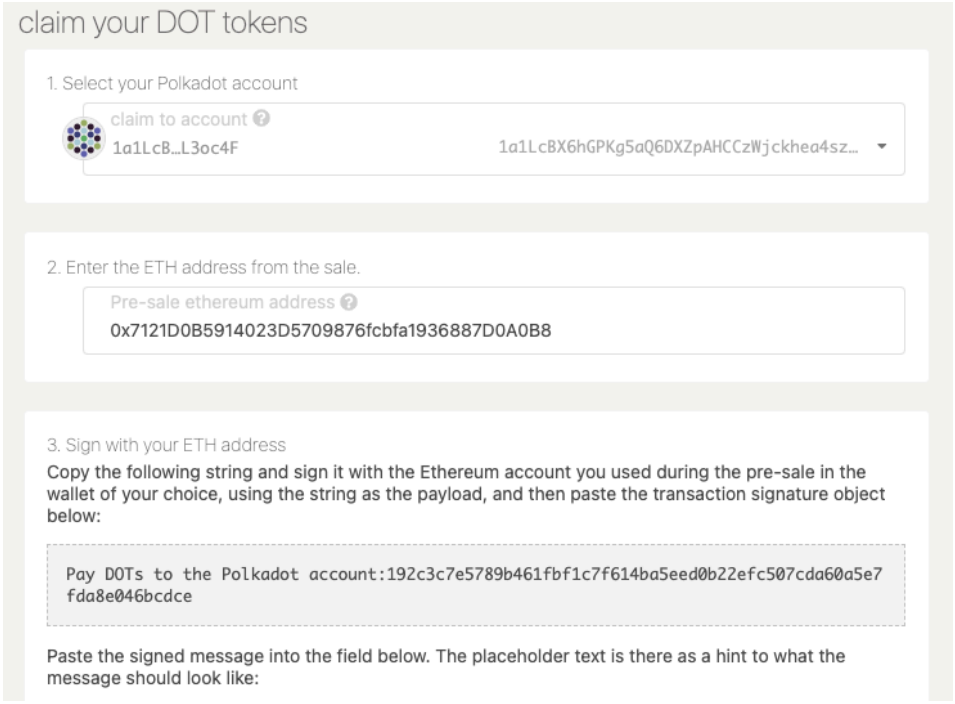
```
subkey inspect
0x192c3c7e5789b461fbf1c7f614ba5eed0b22efc507cda60a5e7fda8e046bcdce --network
polkadot

Secret Key URI
`0x192c3c7e5789b461fbf1c7f614ba5eed0b22efc507cda60a5e7fda8e046bcdce` is account:
  Secret seed:
0x192c3c7e5789b461fbf1c7f614ba5eed0b22efc507cda60a5e7fda8e046bcdce
  Public key (hex):
0x5a095388156e3a68d4fb7cbaef981478e1a0be6d4998f00dffc3e4e9c60c104c
  Account ID: 0x5a095388156e3a68d4fb7cbaef981478e1a0be6d4998f00dffc3e4e9c60c104c
  SS58 Address: 13345eVmibkqE6wG94JD8MSadyXGNBcRX5YAhht3Sfxcaw9U
```

You will notice that the Subkey Tool recognizes the correct address network and returns the associated public key. The public key is returned as a hexadecimal string (i.e. prefixed with **"0x"**). **For both SS58 addresses, the same public key is returned.**

Address Verification

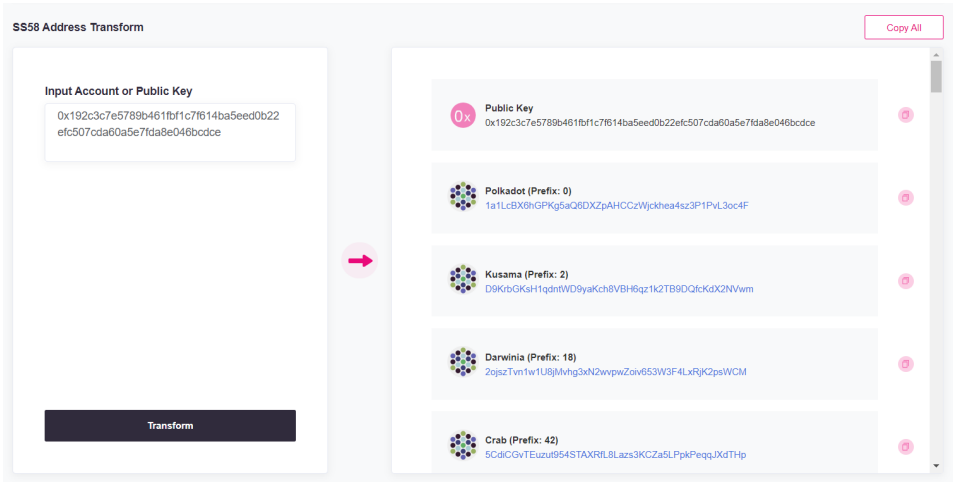
Consider the following example:



If you are comfortable enough to distinguish between each account parameter, you can prefix the public-key string with **"0x"** on your own:

From: `Pay DOTs to the Polkadot account:192c3c7e5789b461fbf1c7f614ba5eed0b22efc507cda60a5e7fda8e046bcdce`, we prefix the address by "0x" -> `0x192c3c7e5789b461fbf1c7f614ba5eed0b22efc507cda60a5e7fda8e046bcdce`.

Using the handy subscan tool, you can verify both address associations to your public key. Copy your public key into the "Input Account or Public Key" textbox and click "Transform" at the bottom. On the right-hand side, the addresses for Polkadot and Substrate that are returned based on your public key should match the ones you inspected.



> NOTE: You may have to scroll down to the bottom of the menu to find the Substrate address based on the menu listings. You will notice that many networks that also use the same Substrate address.

You can verify your public key verification by recalling that Polkadot addresses start with a '1', whereas Substrate addresses generally start with a '5' (Kusama addresses start with a capital letter). See Addresses for more details.

Furthermore, the Utility Scripts can be referenced for how the verification is performed: pubkeyToAddress.js demonstrates how a single public key interprets a Polkadot, Substrate, or Kusama address.

Resources

- Understanding Accounts and Keys in Polkadot - An explanation of what the different kinds of accounts and keys are used for in Polkadot, with Bill Laboon and Chinmay Patel of BlockX Labs.

Edit this page

*Last updated on **11/7/2021** by **Radha***

Account Generation

An address is the public part of a Polkadot account. The private part is the key used to access this address. The public and private parts together make up a Polkadot account.

There are several ways to generate a Polkadot account:

- Polkadot{.js} Browser Extension **RECOMMENDED FOR MOST USERS**
- Subkey **ADVANCED and MOST SECURE**
- Polkadot-JS Apps
- Parity Signer
- Vanity Generator
- Ledger Hardware Wallet

> If you prefer video instructions for creating an account using Polkadot JS, we have an easy to follow guide for beginners on YouTube

DISCLAIMER: Key Security

The *only* ways to get access to your account are via your secret seed or your account's JSON file in combination with a password. You must keep them both secure and private. If you share them with anyone they will have full access to your account, including all of your funds. This information is a target for hackers and others with bad intentions - see also How to Recognize Scams.

On this page, we recommend a variety of account generation methods that have various convenience and security trade-offs. Please review this page carefully before making your account so that you understand the risks of the account generation method you choose and how to properly mitigate them in order to keep your funds safe.

Storing your key safely

The seed is your **key** to the account. Knowing the seed allows you, or anyone else who knows the seed, to re-generate and control this account.

It is imperative to store the seed somewhere safe, secret, and secure. If you lose access to your account (i.e. you forget the password for your account's JSON file), you can re-create it by entering the seed. This also means that somebody else can have control over your account if they have access to your seed.

For maximum security, the seed should be written down on paper or another non-digital device and stored in a safe place. You may also want to protect your seed from physical damage, as well (e.g. by storing in a sealed plastic bag to prevent water damage, storing it in a fireproof safe, etching it in metal, etc.) It is recommended that you store multiple copies of the seed in geographically separate locations (e.g., one in your home safe and one in a safety deposit box at your bank).

You should **not store your seed on any kind of computer that has or may have access to the internet in the future.**

Storing your account's JSON file

The JSON file is encrypted with a password, which means you can import it into any wallet which supports JSON imports, but to then use it, you need the password. You don't have to be as careful with your JSON file's storage as you would with your seed (i.e. it can be on a USB drive near you), but remember that in this case, your account is only as secure as the password you used to encrypt it. Do not use easy to guess or hard to remember passwords. It is good practice to use a mnemonic password of four to five words. These are nearly impossible for computers to guess due to the number of combinations possible, but much easier for humans to remember.

Polkadot{.js} Browser Extension

The Polkadot{.js} Extension provides a reasonable balance of security and usability. It provides a separate local mechanism to generate your address and interact with Polkadot.

This method involves installing the Polkadot{.js} plugin and using it as a "virtual vault," separate from your browser, to store your private keys. It also allows the signing of transactions and similar functionality.

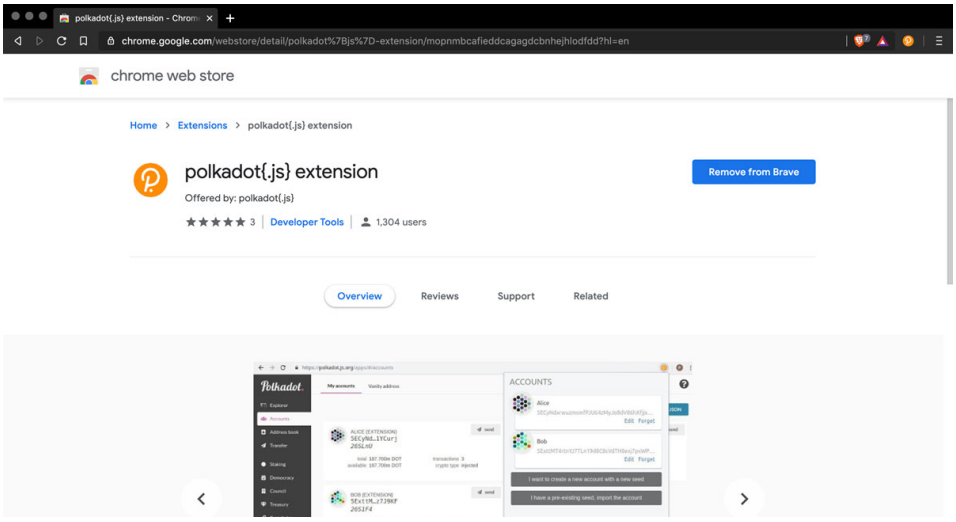
It is still running on the same computer you use to connect to the internet with and thus is less secure than using Parity Signer or other air-gapped approaches.

Install the Browser Extension

The browser extension is available for both Google Chrome (and Chromium-based browsers like Brave) and FireFox.

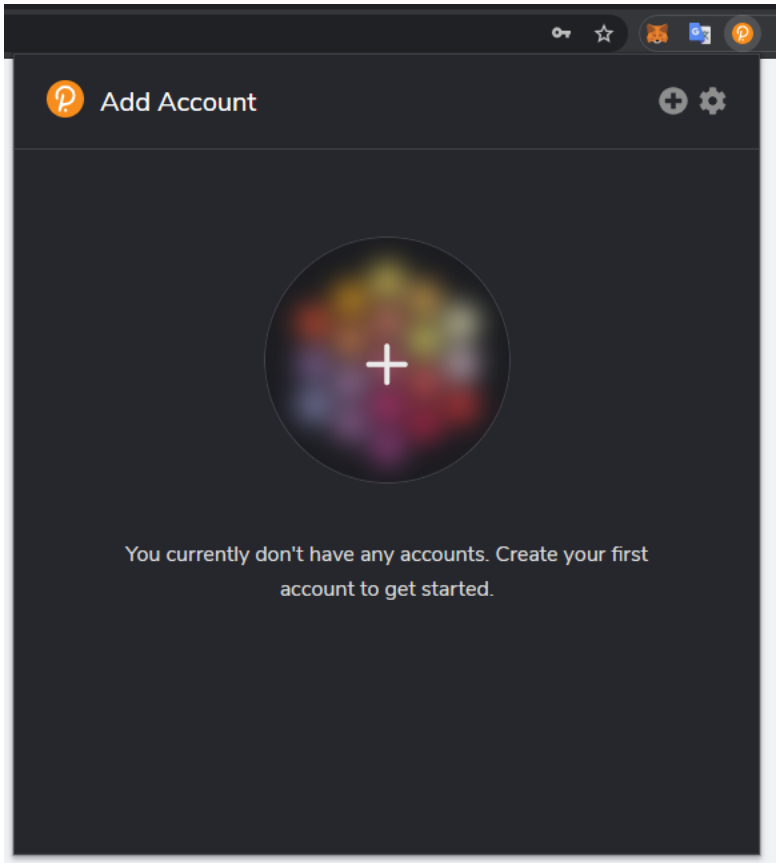
If you would like to know more or review the code of the plugin yourself, you can visit the GitHub source repository.

After installing the plugin, you should see the orange and white Polkadot{.js} logo in the menu bar of your browser.

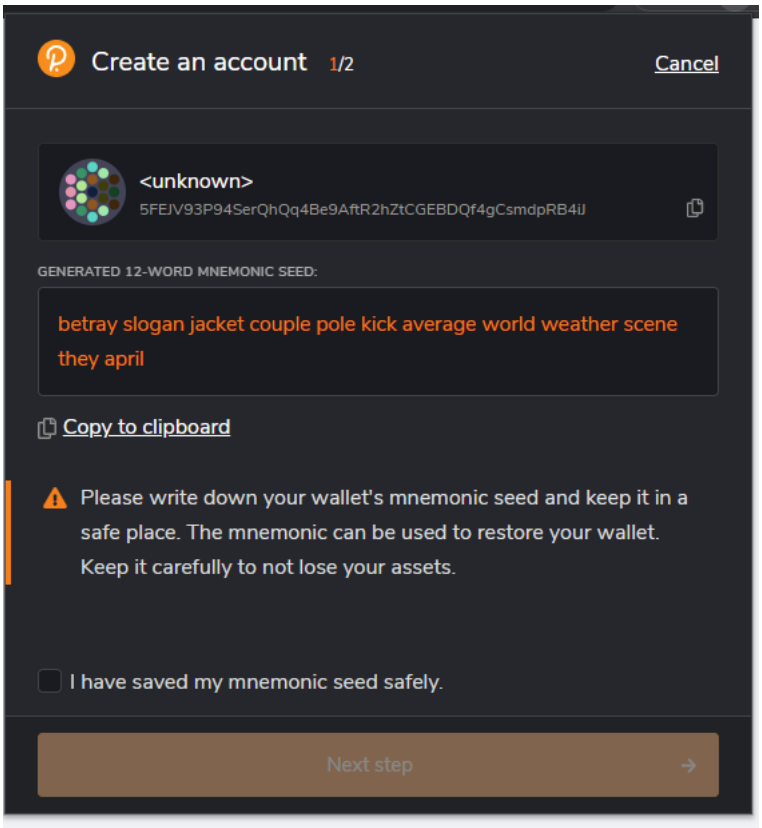


Create Account

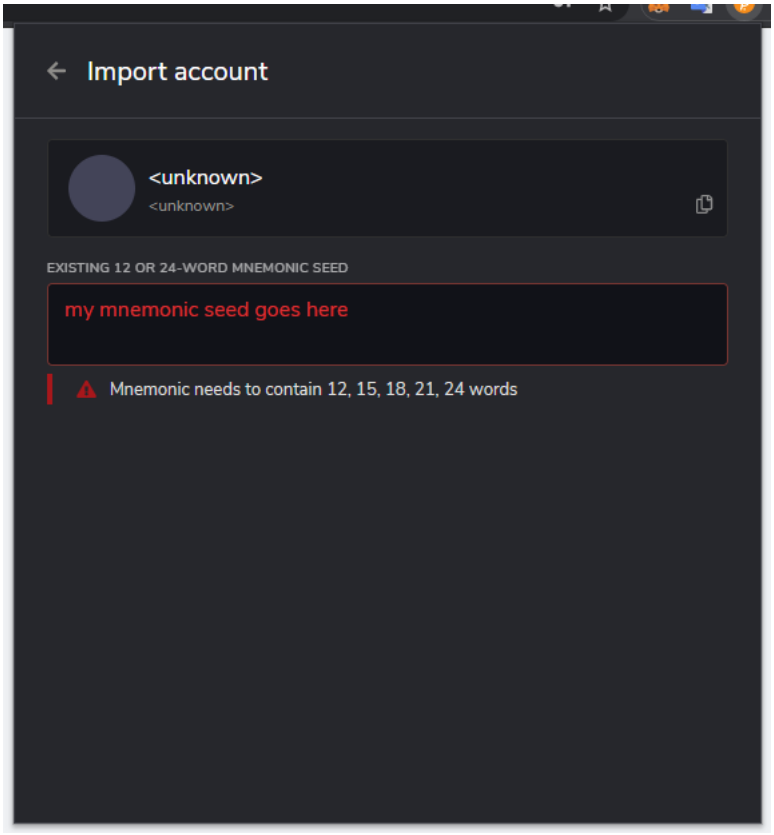
Open the Polkadot{.js} browser extension by clicking the logo on the top bar of your browser. You will see a browser popup, not unlike the one below.



Click the big plus button or select "Create new account" from the small plus icon in the top right. The Polkadot{.js} plugin will then use system randomness to make a new seed for you and display it to you in the form of twelve words.



You should back up these words as explained above. It is imperative to store the seed somewhere safe, secret, and secure. If you cannot access your account via Polkadot{.js} for some reason, you can re-enter your seed through the "Add account menu" by selecting "Import account from pre-existing seed".



Name Account

The account name is arbitrary and for your use only. It is not stored on the blockchain and will not be visible to other users who look at your address via a block explorer. If you're juggling multiple accounts, it helps to make this as descriptive and detailed as needed.

Enter Password

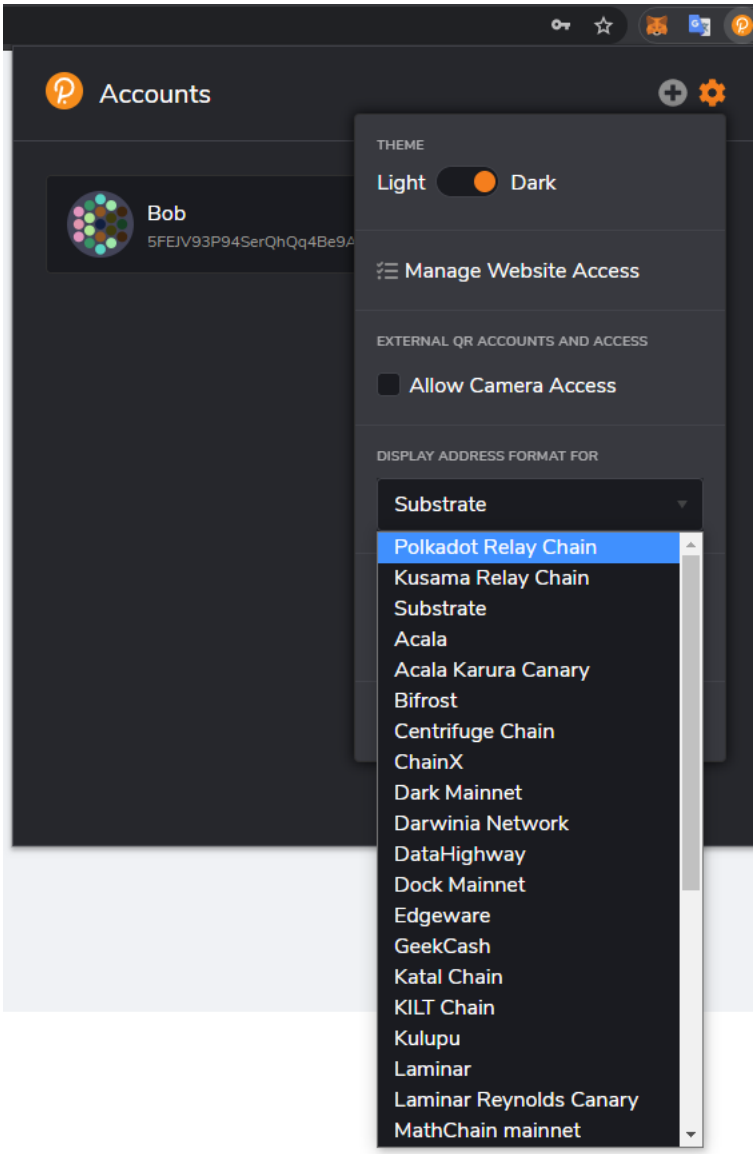
The password will be used to encrypt this account's information. You will need to re-enter it when using the account for any kind of outgoing transaction or when using it to cryptographically sign a message.

Note that this password does NOT protect your seed phrase. If someone knows the twelve words in your mnemonic seed, they still have control over your account even if they do not know the password.

Set Address for Polkadot Mainnet

Now we will ensure that the addresses are displayed as Polkadot mainnet addresses.

Click on "Options" at the top of the plugin window, and under "Display address format for" select "Polkadot Relay Chain".



Your address' format is only visual - the data used to derive this representation of your address are the same, so you can use the same address on multiple chains. However, for privacy reasons, we recommend creating a new address for each chain you're using.

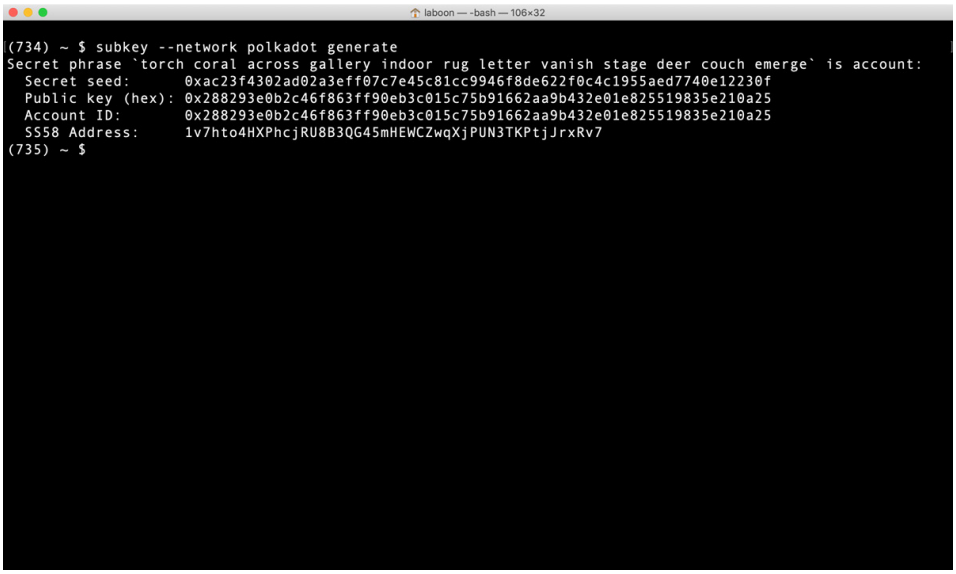
Our Accounts page also has a tool you can use to convert your address between the different chain formats.

You can copy your address by clicking on the account's icon while the desired chain format is active. E.g. selecting "Substrate" as the format will change your address to start with the number 5, and clicking the colorful icon of your account will copy it in that format. While in Polkadot mode (starts with 1), that format will be copied, and so on.

Subkey

Subkey is recommended for technically advanced users who are comfortable with the command line and compiling Rust code. Subkey allows you to generate keys on any device that can compile the code. Subkey may also be useful for automated account generation using an air-gapped device. It is not recommended for general users.

For detailed build and usage instructions of subkey, please see here.



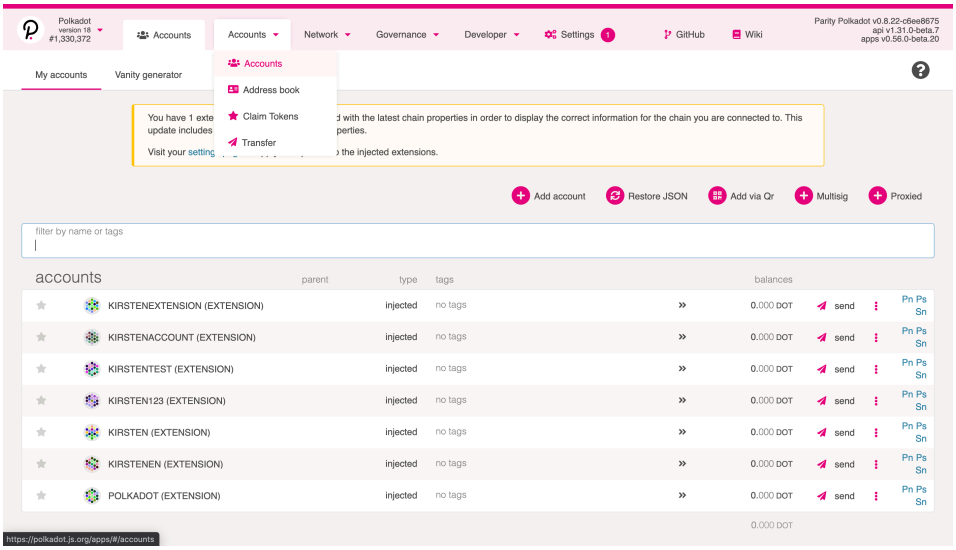
Polkadot-JS Apps

> Please note! If you use this method to create your account and clear your cookies in your browser, your account will be lost forever if you do not back it up. Make sure you store your seed phrase in a safe place, or download the account's JSON file if using the Polkadot{.js} browser extension. Learn more about account backup and restoration here.

Using the Polkadot-JS user interface without the plugin is **not recommended**. It is the least secure way of generating an account. It should only be used if all of the other methods are not feasible in your situation.

Go to Polkadot-JS Apps

Navigate to Polkadot-JS Apps and click on "Accounts" underneath the Accounts tab. It is located in the navigation bar at the top of your screen.



> To create an account on a different network than Polkadot, you'll need to click on the network selection in the top left corner of the navigation menu. A pop-up sidebar will appear listing live, testing, and custom node to choose from. Do remember to hit the "Switch" button when you want to switch your network.

Start Account Generation

Click on the "Add Account" button. You should see a pop-up similar to the process encountered when using the Polkadot JS Extension method above. Follow the same instructions and remember to store your seed safely!

Create and Back-Up Account

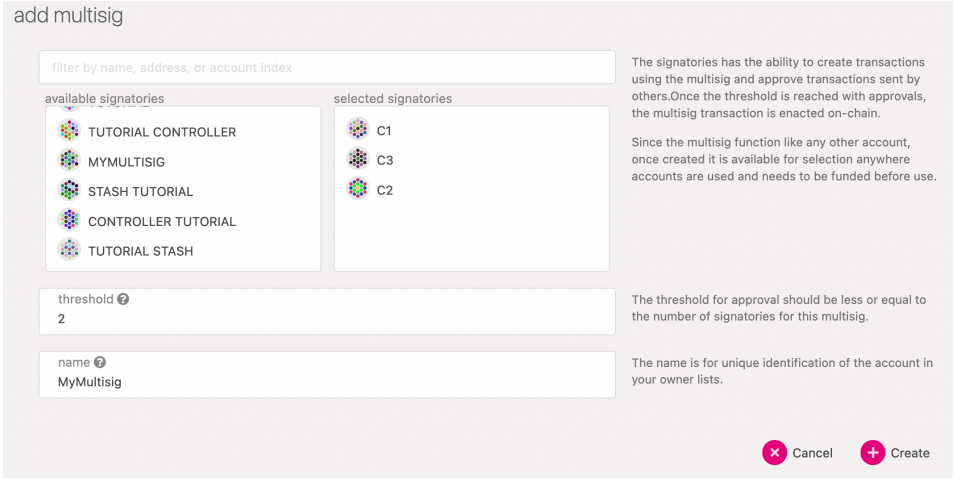
Click “Save” and your account will be created. It will also generate a backup JSON file that you should safely store, ideally on a USB off the computer you're using. You should not store it in cloud storage, email it to yourself, etc.

You can use this backup file to restore your account. This backup file is not readable unless it is decrypted with the password.

Multi-signature Accounts

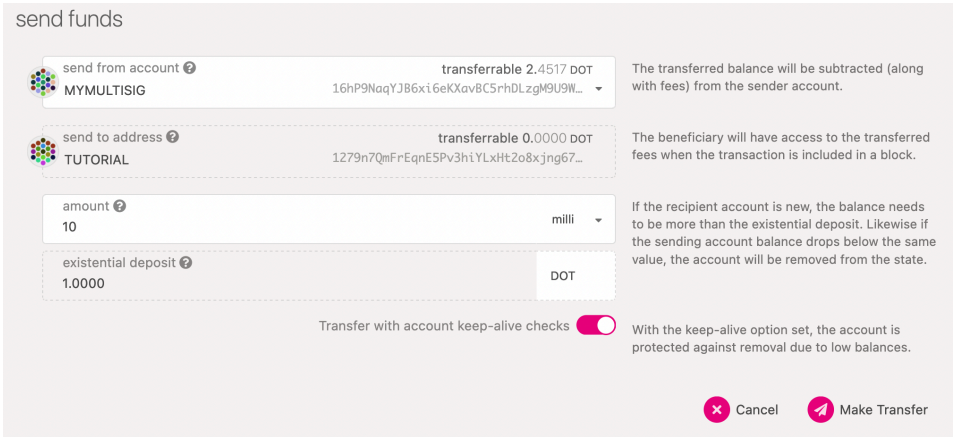
Multi-signature accounts are accounts created from several standard accounts (or even other multi-sig accounts). For a full explanation, please see the Accounts Explainer section on multi-sigs.

On the Accounts tab, click the `Multisig` button. Enter the threshold and add signatories. The threshold must be less than or equal to the number of signatories. The threshold indicates how many members must agree for an extrinsic submission to be successful. Click `Create` when done.

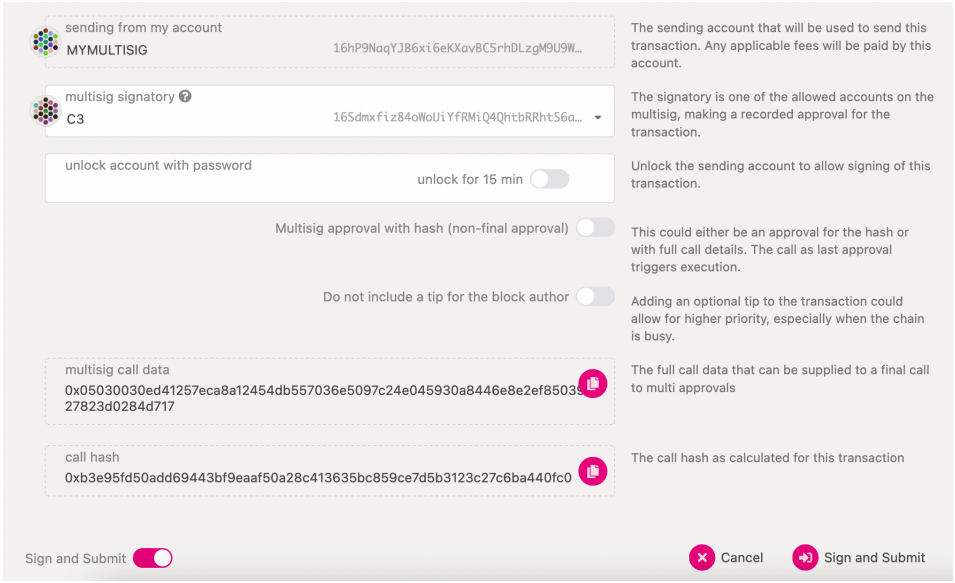


This merely calculates the multi-signature's address and adds it to your UI. The account does not exist yet, and is subject to the same Existential Deposit and Reaping rules as regular accounts.

Suppose we funded it with some tokens, and now want to send from the multi-sig to another account.



The next step is to sign the transaction from with enough accounts to meet the threshold; in the above case, two out of three signatories must sign.



There is currently no indication of a pending transaction from a multi-sig in the UI. This means the second signatory must **repeat the call in full** in order to sign it. In other words:

- if Alice initiates a transaction from the multi-sig to Ferdie for 150 tokens, there will be a pending transaction in the chain.
- if Bob initiates a transaction from the multi-sig to Ferdie for 250 tokens, there will be **another** pending transaction in the chain, and the first one will not complete.
- because the threshold is 2/3, Charlie can now finalize either or both of these by repeating the desired transaction.

Other calls work the same - if a multi-sig wants to become a Council member, the candidacy request has to come from the multi-sig, but be signed (re-requested) from each signatory until the threshold is reached.

Signatories should communicate off-chain to prevent many pending transactions and crossed communication lines on-chain.

> The bigger the multisig, the more of a deposit an account needs to put down when initiating a multi-sig call. This is to prevent chain storage spam with pending but never-resolved multi-sig transactions. Once a call is resolved (canceled or executed) the deposit is returned to the initiator. The deposit is not taken from the multi-sig's balance but from the initiator.

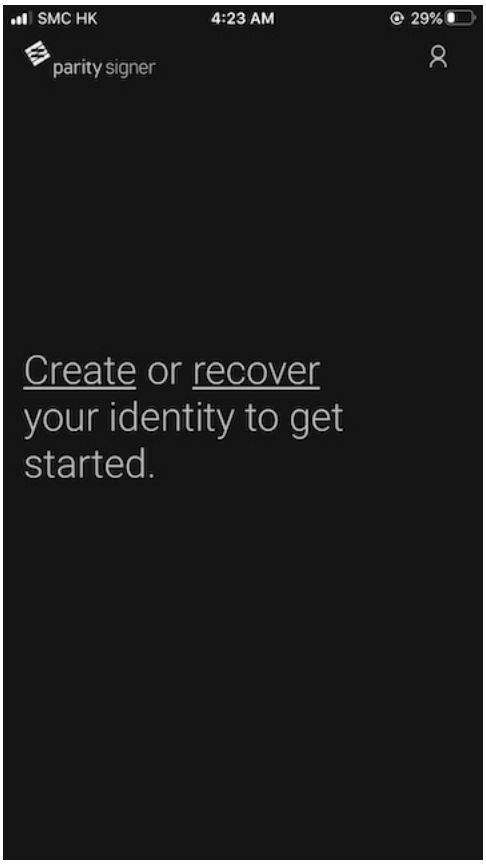
For a more in-depth introduction into multi-signature accounts on Polkadot, please see the accounts page section on Multi-sigs.

Parity Signer

Parity Signer is a secure way of storing your DOT on an air-gapped device. It is highly recommended that you turn off wifi, cellular network, Bluetooth, NFC, and any other communications methods after installing it. The device needs to be offline and only you should be viewing the device screen. If you have any communications methods turned on, you will see an "unshielded" icon in red in the top-right corner that indicates your connection may not be secure.

Create Account

Click "Create" to create an identity, or "recover" if you have previously backed up the recovery phrase. You can have multiple identities on one device. Each identity can manage multiple addresses on different networks.



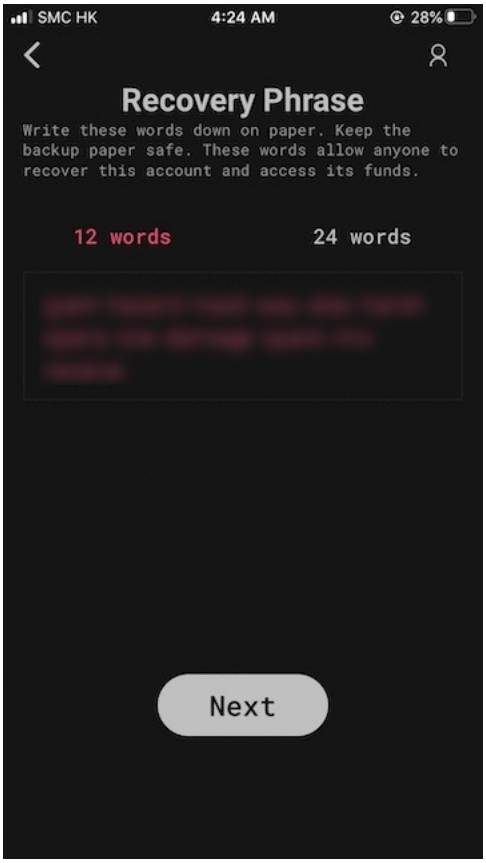
Name Account

Input the name for your identity and then click "Create".

Back Up Account

Parity Signer will then generate a recovery phrase for you and display it in the form of 12 or 24 words.

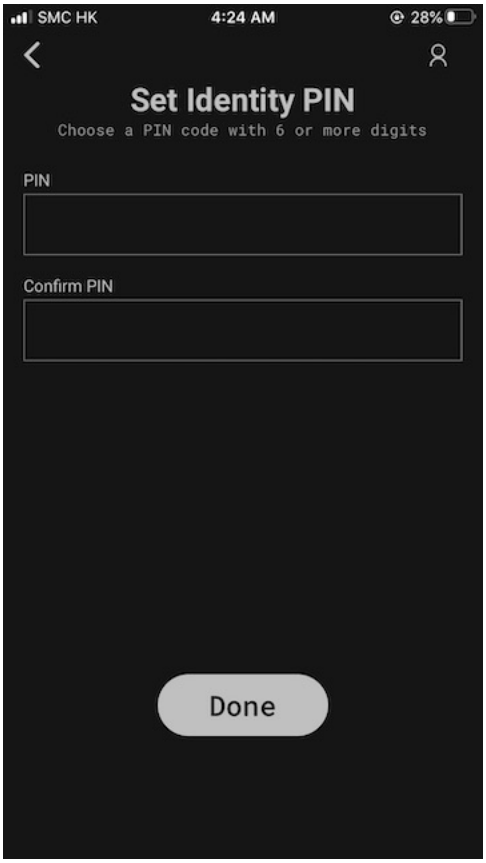
You should write down this recovery phrase on paper and store it somewhere safe.



Set PIN

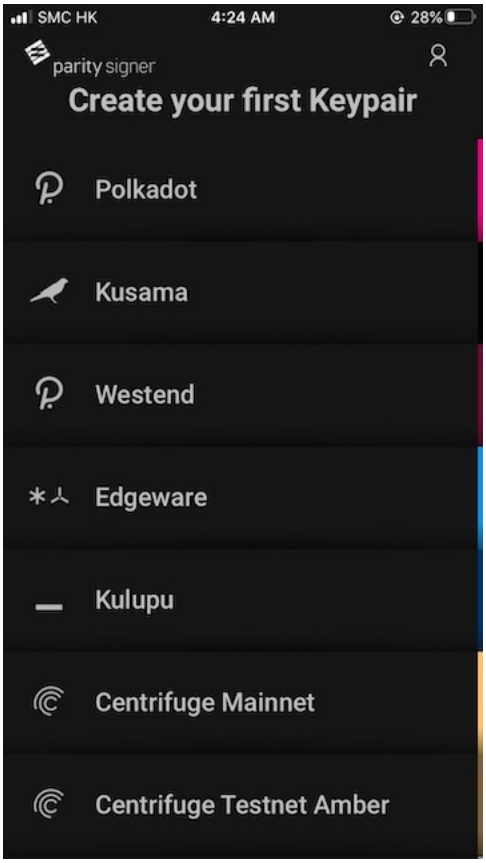
After confirming that you have backed up your seed, a new textbox will appear in which you can set a PIN. The PIN code should contain at least 6 digits. If the PIN codes do not match, it will not allow you to create an account. The PIN code will be used when signing any transaction, or to protect sensitive operations such as deleting an identity or revealing the recovery phrase.

Note that if someone knows the 12/24 words in your recovery phrase, they will still have control over your account, even if they do not know the PIN.



Get Address

Choose which network you would like to create an address for by clicking the name of the network. For example, if you select "Polkadot", a new Polkadot address will be created for you under this identity.



There is currently no way to copy your address from Parity signer in plain text in order to send it via text or email. You must use the QR method.

Your Address

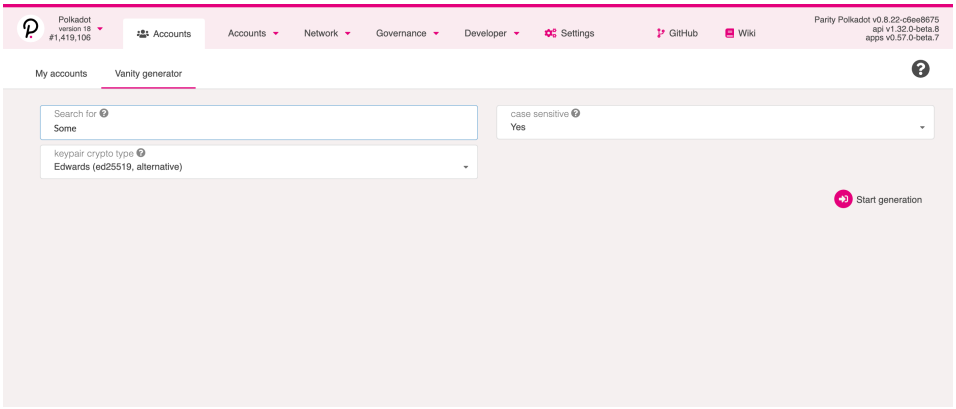
The address will be shown as a QR code. You can import your address to the Polkadot-JS Apps by going to the Accounts page on an Internet-connected computer and click "Add via QR", and following the instructions to add the account. An account created this way will always require you to sign messages with your Parity Signer device. It will do this only by scanning and displaying QR codes, leaving even someone with total control of your internet-connected computer has a very small and limited scope for interacting with the Parity Signer device, which can continue to keep your key safe.

Ledger Hardware Wallet

To use a Ledger hardware wallet to create your address and keep your tokens in cold storage, follow the instructions on our Ledger hardware wallet guide page.

Vanity Generator

The vanity generator is a tool on Polkadot-JS UI that lets you generate addresses that contain a specific substring.

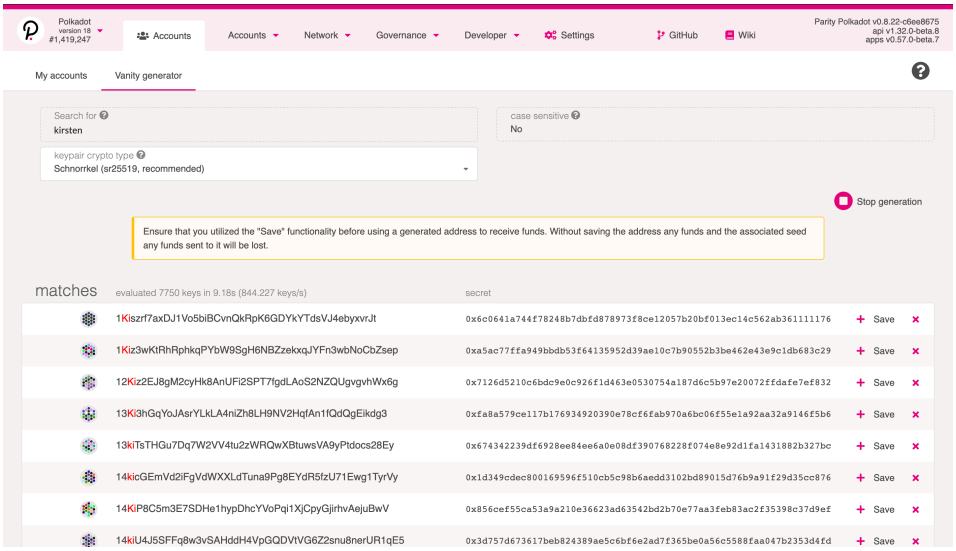


The vanity generator takes the following parameters:

- "Search for": The substring that you would like to include in your new address.

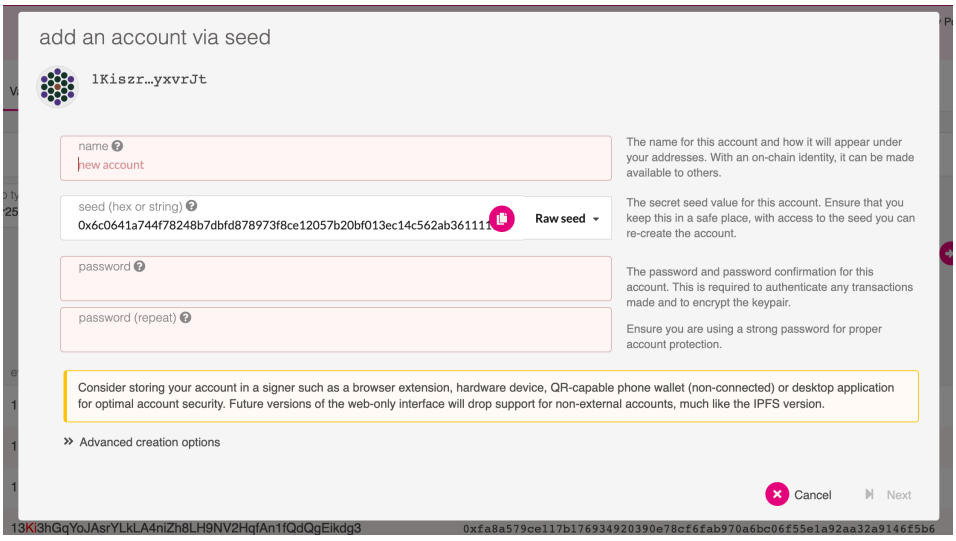
- "case sensitive": "Yes" if the search is case sensitive; "no" if not.
- "keypair crypto type": Specifies the type of account you'd like to generate; Schnorrkel is recommended in most cases for its security.

If you've filled out all details above and hit the "Start generation" button, a list of accounts will start generating on your screen. Note that depending on the length of the substring and the processing speed of your computer, it may take some time before any accounts appear.



The "Save" button will allow you to save the generated accounts - they are not saved if you do not choose to do this. The next steps are identical to the steps above on creating an account on the UI, where a password and name need to be filled in, and a backup file of your account will be downloaded.

Note that the Subkey tool also has vanity generation built-in, and is orders of magnitude faster than the web version. If you need to generate addresses with longer strings, or need plenty of them, we recommend using Subkey instead.



Edit this page

*Last updated on **11/11/2021** by **Zak Greant***

General

About

FAQ

Contact

Build

Grants and Bounties

Careers

Technology

Technology

Token

Telemetry

Substrate

Whitepaper

Lightpaper

Community

Community

Documentation

Brand Assets

Blog

Element Chat

Medium

Subscribe to the newsletter to hear about Polkadot updates and events.

Subscribe

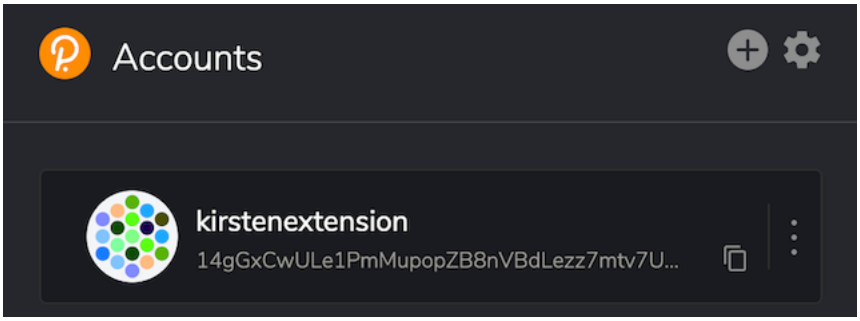
Backing up and Restoring Accounts

Depending on what software you are using to access your account, there are various ways to back up and restore your account. It is a good idea to back your information up and keep it in a secure place.. Note that in order to recover an account, you should create your account according to the instructions here. In general, however, as long as you know how you created your account, and have the seed phrase (mnemonic phrase) or JSON file (and password) stored securely, you will be able to restore your account.

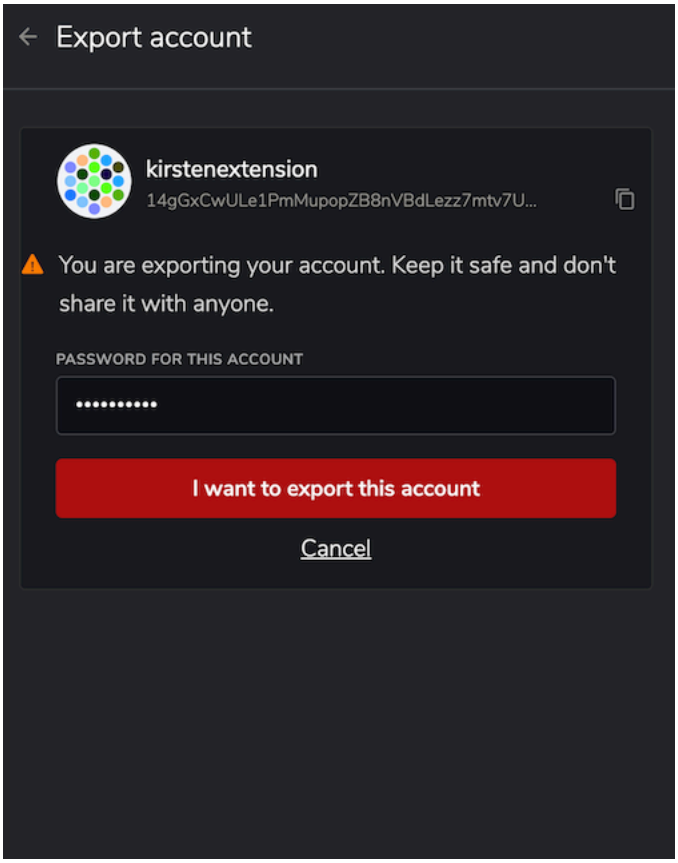
This page covers backing up and restoring accounts in Polkadot{.js} Browser Plugin, Polkadot-JS UI, and Parity Signer. For other wallet applications, please see their specific documentation.

Polkadot{.js} Browser Plugin

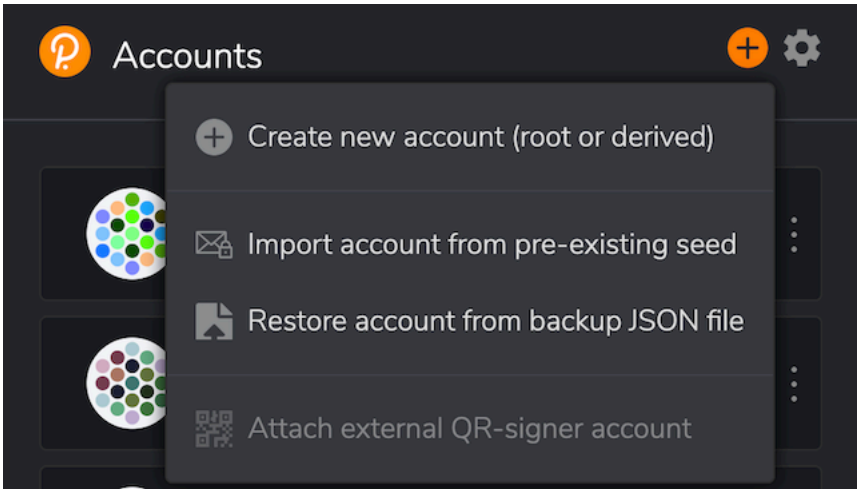
To back up an account using the Polkadot{.js} browser plugin, open the extension and select the desired account to back up. Click on the three dots beside the address to open up the account options menu.



Click on the Export Account button, which will then ask you for the password for that specific account. Once you enter the correct password, the browser will automatically download a `.json` file that will hold all the account restoration details that will be used when you restore your account. You should store this file securely. Note that you will need the password for this account to restore it.



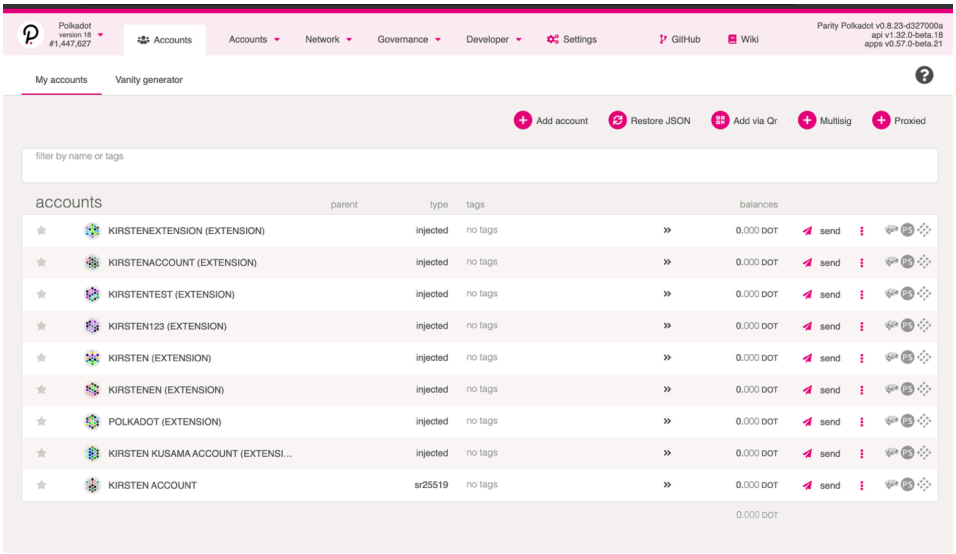
To restore the account from this JSON file, once again open the Polkadot{.js} browser plugin. Click on the + button at the top. This will open up a menu with several choices - select "Restore account from backup JSON file". The program then prompts you for the `.json` file which was download earlier and the password for that account.



Once these are filled out, and the "Restore" button has been pressed, you'll be taken back to the main page of the plugin. This account will now be listed with the rest of your accounts.

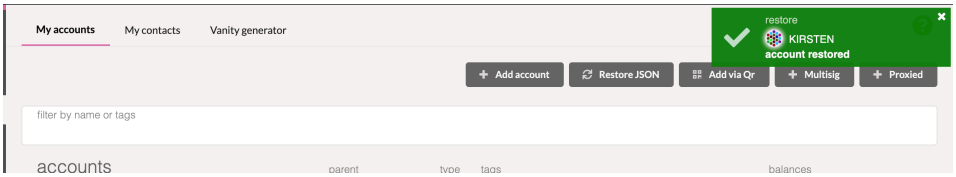
Polkadot-JS

If you're using the main Polkadot-JS UI, restoring an account will feel similar to restoring an account on Polkadot-JS browser plugin. Navigate to the Accounts page of Polkadot-JS.



Click on the "Restore JSON" button, which will let you upload your `.json` file that you downloaded and enter your password for that account. This `.json` file holds all relevant data about the account to be used in account restoration. Note that you will need to enter your password here; the file cannot be unencrypted without it.

After you press the "Restore" button, you should see a green notification letting you know that your account has been restored. It will now be included in your accounts list on this browser.

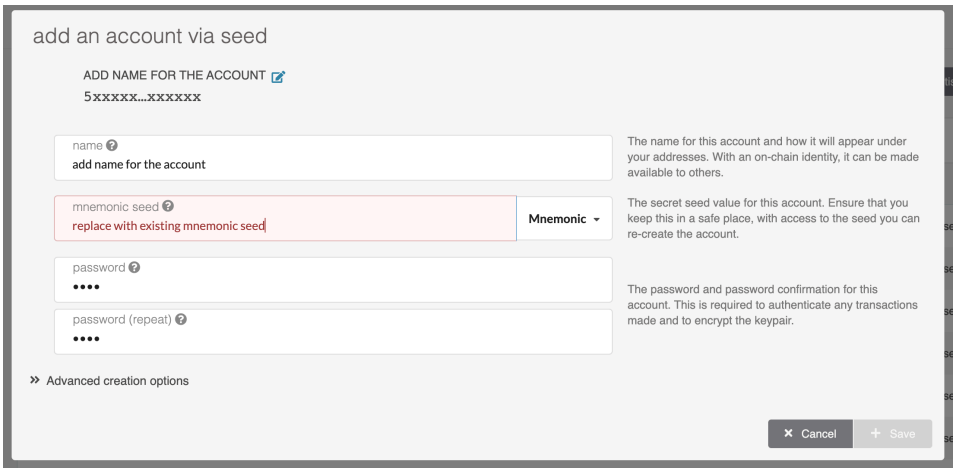


Using an Existing Mnemonic Seed to Restore an Account

You can also always restore an account by using the mnemonic phrase (seed words).

To do this with Polkadot-JS, navigate to the Polkadot-JS Accounts Page. Click on the "Add Account" button, and enter a name and password for the account. The name and the password of this added account can be set to whatever you'd like, it does not need to be the same name and password as when this account was initially created.

After this, delete the generated mnemonic phrase (seed words) and replace them with your *existing seed words*. When you replace the generated mnemonic seed with your existing one, you are not creating a new account, rather adding that account onto the Polkadot-JS UI. Any account using the same seed words will have control over that account on-chain. This is why it is so important to keep your seed words secret and safe.

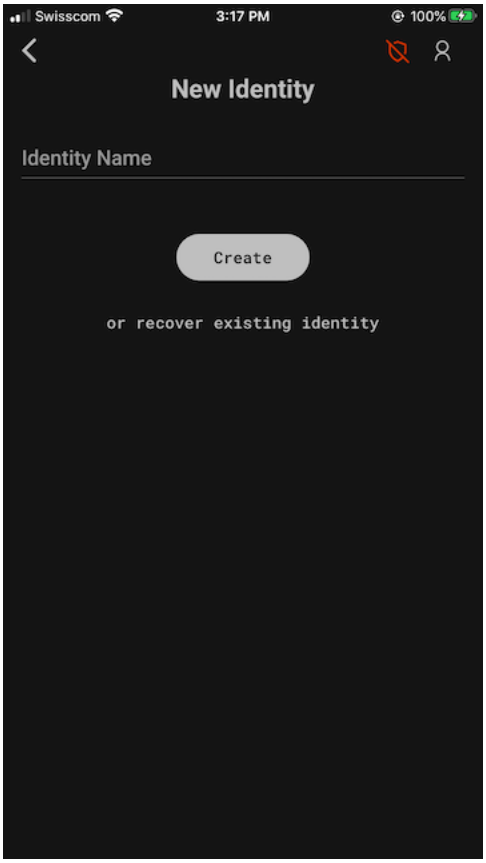


Finally, click the "Save" button, then click the "Create and backup account" button. This will download the `.json` file which contains the data to be used in account restoration. You can use this JSON file to restore this account in the future using the instructions above, or simply delete the file and continue to use the mnemonic phrase to restore the account if necessary.

Parity Signer

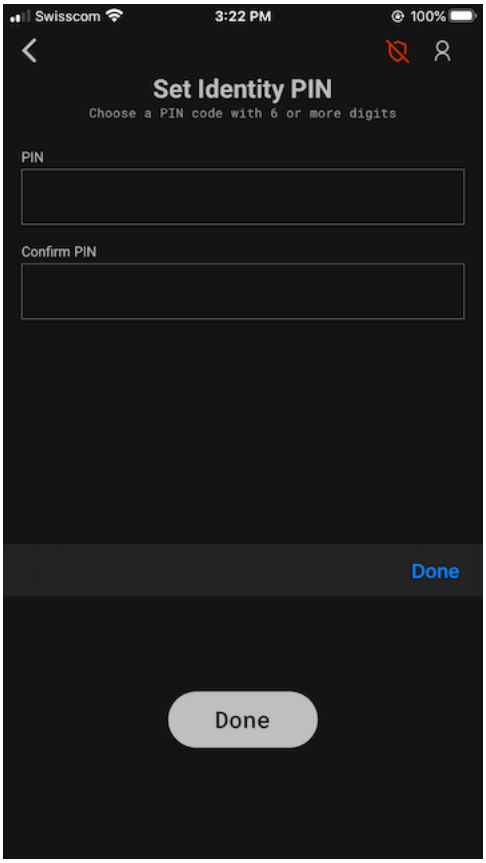
If you've created an account with Parity Signer, you can recover that account with your seed words. If you generated the account with another wallet, there may be additional steps necessary, including setting the derivation path: see this document for details.

On Parity Signer, click on the top-right user icon on the screen. Proceed to "+ Add Identity". On this screen, tap on the "recover existing identity" button.



Enter in the identity name and the mnemonic seed phrase from the account you'd like to restore.

Set an identity PIN that will be used to unlock this account when you need to.



The identity has now been recovered and you can select a network to create the first account.

Edit this page

*Last updated on **10/23/2021** by **Danny Salman***

Assets

Assets in the Polkadot network can be represented on several chains. They can also take many forms, from a parachain's native token to on-chain representations of off-chain reserves. This page focuses on the latter, namely assets that would be issued by a creator (e.g. rights to audited, off-chain reserves held by the creator, or art issued as an NFT).

The Statemint parachain hosts data structures and logic that specialize in the creation, management, and use of assets in the Polkadot network. Although other parachains can host applications dealing with assets on Statemint, Statemint can be thought of as the "home base" of assets in the network.

Statemint uses DOT as its native token. The chain yields its governance to its parent Relay Chain, and has no inflation or era-based rewards for collators (although collators do receive a portion of transaction fees). As a common good parachain, Statemint has a trusted relationship with the Relay Chain, and as such, can teleport DOT between itself and the Relay Chain. That is, DOT on Statemint is just as good as DOT on the Relay Chain.

Statemint does not support smart contracts. See the Advanced section at the bottom for discussion on using proxy and multisig accounts to replicate oft used contract logic.

Fungible Assets

Fungible assets are those that are interchangeable, i.e. one unit is equivalent to any other unit for the purposes of claiming the underlying item. Statemint represents fungible assets in the Assets pallet. For those familiar with the ERC20 standard, this pallet presents a similar interface. However, the logic is encoded directly in the chain's runtime. As such, operations are not gas metered and instead are benchmarked upon every release, leading to efficient execution and stable transaction fees.

Creation and Management

Anyone on the network can create assets on Statemint, as long as they can reserve the required deposit of 100 DOT.. The network reserves the deposit on creation. The creator also must specify a unique `AssetId`, an integer of type `u32`, to identify the asset. The `AssetId` should be the canonical identifier for an asset, as the chain does not enforce the uniqueness of metadata like "name" and "symbol". The creator must also specify a minimum balance, which will prevent accounts from having dust balances.

An asset class has a number of privileged roles. The creator of the asset automatically takes on all privileged roles, but can reassign them after creation. These roles are:

- Owner
- Issuer
- Admin
- Freezer

The owner has the ability to set the accounts responsible for the other three roles, as well as set asset metadata (e.g. name, symbol, decimals). The issuer can mint and burn tokens to/from addresses of their choosing. The freezer can freeze assets on target addresses or the entire asset class. The admin can make force transfers as well as unfreeze accounts of the asset class. **Always refer to the reference documentation for certainty on privileged roles.**

An asset's details contain one field not accessible to its owner or admin team, that of asset sufficiency. Only the network's governance mechanism can deem an asset as *sufficient*. A balance of a non-sufficient asset (the default) can only exist on already-existing accounts. That is, a user could not create a new account on-chain by transferring an insufficient asset to it; the account must already exist by having more than the existential deposit in DOT (or a sufficient asset). However, assets deemed *sufficient* can instantiate accounts. In the future, *sufficient* assets will be able to pay transaction fees, such that users can transact on Statemint without the need for DOT .

Using

Users have a simple interface, namely the ability to transfer asset balances to other accounts on-chain. As mentioned before, if the asset is not *sufficient*, then the destination account must already exist for the transfer to succeed.

The chain also contains a `transfer_keep_alive` function, similar to that of the Balances pallet, that will fail if execution would kill the sending account.

Statemint also sweeps dust balances into transfers. For example, if an asset has a minimum balance of 10 and an account has a balance of 25, then an attempt to transfer 20 units would actually transfer all 25.

Application Development

Statemint provides an `approve_transfer`, `transfer_approved`, and `cancel_approval` interface. Application developers can use this interface so that users can authorize the application to effectuate transfers up to a given amount on behalf of an account.

Cross-Chain Accounting

Statemint uses a reserve-backed system to manage asset transfers to other parachains. It tracks how much of each asset has gone to each parachain and will not accept more back from a particular parachain.

As a result of this, asset owners can use Statemint to track information like the total issuance of their asset in the entire network, as parachain balances would be included in the reserve-backed table. Likewise, for the minting and burning of tokens, an asset's team can perform all operations on Statemint and propagate any minted tokens to other parachains in the network.

Parachains that want to send assets to other parachains should do so via instructions to Statemint so that the reserve-backed table stays up to date. For more info, see the "Moving Assets between Chains in XCM" section of the article on the XCM format.

Non-Fungible Assets

Unlike fungible assets, the particular instance of a non-fungible asset (NFT) has meaning separate from another instance of the same class. Statemint represents NFTs in the Uniques pallet.

Similar to the Assets pallet, this functionality is encoded into the chain. Operations are benchmarked prior to each release in lieu of any runtime metering, ensuring efficient execution and stable transaction fees.

Creation and Management

Anyone on the network can create an asset class, as long as they reserve the required deposit of 100 DOT on Statemint. Creating instances of a class also requires a per-instance deposit, unless the chain's governance designates the class as "free holding", allowing the class to mint more instances without deposit. The creator must specify a `ClassId`, which, like its cousin `AssetId`, should be the canonical identifier for the class.

The creator can also specify the same privileged roles of Owner, Admin, Issuer, and Freezer.

Asset classes and instances can have associated metadata. The metadata is an array of data that the class Owner can add on-chain, for example, a link to an IPFS hash or other off-chain hosting service. The Uniques pallet also supports setting key/value pairs as attributes to a class or instance.

Using

Users can transfer their NFTs to other accounts. The chain also provides an `approve_transfer`, `transfer_approved`, and `cancel_approval` interface that application developers can use to allow users to authorize an application to transfer an instance on their behalf.

Advanced Techniques

Many asset creators on other networks use smart contracts to control privileged functions like minting and burning. Although Statemint does not have a smart contract interface, it contains the Multisig, Proxy, and Utility pallets, which will meet most account management needs.

For example, if a team wants sign-off from two groups to perform a privileged operation, it could create a 2-of-2 multisig from two anonymous proxies, and then set members from each group as proxies to those two accounts.

Edit this page

*Last updated on **10/20/2021** by **Danny Salman***

NFTs

This page is a high level overview of NFTs in general, and the various approaches to NFTs within the Polkadot network.

Fungibility

NFT stands for *non fungible token*. Fungibility means interchangeability inside of a group. In theory, a $20 bill is always worth $20 in a store, and identical in value to any other $20 bill. It is not, however, fungible with a $1 or $100 dollar bill (outside its group).

A Pokemon™ trading card of a Charizard is non-fungible with a card of Squirtle, but editions of Charizard are fungible with each other.

Fungibility is a spectrum - what's fungible to some might not be fungible to others. In reality, Pokemon™ cards, the canonical example of non fungible assets, are more fungible than US dollar bills each of which has a unique serial number which may be of importance to a government agency. The cards have no serial numbers.

source: Investopedia

Additionally, a digital item like a "purple magic sword" in a game may be fungible with another visually identical sword if all the player cares about is the looks of their character. But if the other sword has a different function, and that function influences the outcome of an adventure the player is about to embark on, then visually identical swords are absolutely non-fungible.

Bearing that in mind, the simplest explanation of NFTs is that **NFTs are rows of arbitrary, project-specific, and non-interchangeable data that can be cryptographically proven to "belong" to someone**. This data can be anything - concert tickets, attendance badges, simple words, avatars, plots of land in a metaverse, audio clips, house deeds and mortgages, and more.

NFT Standards

A general-purpose blockchain is not built to natively understand the concept of NFTs. It is only natively aware and optimized for its own native tokens, but implementations built on top of such a chain are essentially "hacks".

As an example, Ethereum is a general purpose blockchain which does not have the concept of "tokens" (fungible or not) built-in. Tokens in Ethereum are essentially spreadsheets of information that are to be interpreted and read in a certain way by various user interfaces. This *way* in which they should read them is called a *standard*.

The most widespread fungible token standard you may have heard of is ERC20, while the most widespread NFT standard is ERC721, followed closely by ERC1155. The downside of having to define these standards is that they are always instructions for how to read a spreadsheet pretending to serve information in a certain way, which by definition cannot be optimized. For this reason, even on a good day of extremely low network congestion, interactions with NFTs on any EVM chain will cost a few dollars, but were on average around $100 per interaction (transer, mint, sale) in 2021 on Ethereum.

This prevents use cases that go beyond the current craze of *digital dust gathering NFTs* on Ethereum - profile pictures, generative "look once and then put away" art, ENS addresses, and proof of attendance badges.

a typical NFT on Ethereum

For the sake of comparison, we can refer to these as NFTs 1.0: static NFTs that are almost exclusively image-based collectibles of varying rarity.

NFTs 2.0: NFTs in Polkadot and Kusama

This is where Polkadot's technology shines and where NFTs 2.0 come into play.

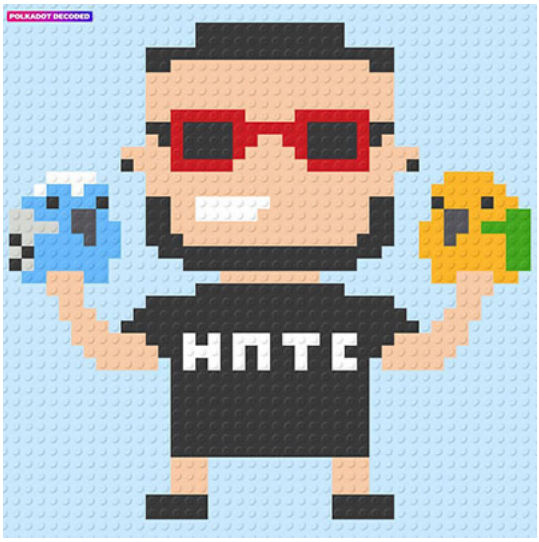
By allowing heterogeneous application-specific shards to exist, builders are able to natively optimize for complex NFT use cases without tradeoffs that would make interacting with the system prohibitively inefficient and expensive in other environments.

The following NFT solutions exist and/or are under development in the Polkadot ecosystem:

Unique network

Unique network, an NFT-specific blockchain offering innovations such as sponsored transactions, bundling fungible tokens with non-fungibles, and spliting NFTs into fungible tokens for partial ownership.

Unique Network have launched two NFT projects to date: Substrapunks as part of Hackusama, and Chelobricks as a recent promotion during Polkadot Decoded. They are currently running a betanet which is bridged to Kusama, and on which these NFTs are already tradable: see Unqnft.io.



Source: Unique network's Chelobrick

Users can send KSM into their Unique Network escrow account, trade with it there, and then send any earned or leftover KSM back.

Unique Network aims to make their marketplace tech open source and whitelabel-friendly. In theory, it should be trivial to set up a new marketplace for your project using Unique's technology. Unique network aims to be a parachain on Polkadot, and Quartz is their Kusama counterpart.

Unique Network works closely with RMRK (see below).

RMRK

RMRK is a "hack", a forced standard directly on top of the Kusama relay chain. Since Kusama is meant to be light-weight so it can process the various parachains connected to it, it is not meant to have any other complex chain logic like native NFTs or smart contracts to enable them. However, because of market demand and Kusama's "chaotic" nature, the RMRK team decided to take the "colored coins" approach from Bitcoin and implement NFTs as graffiti on the Kusama chain.

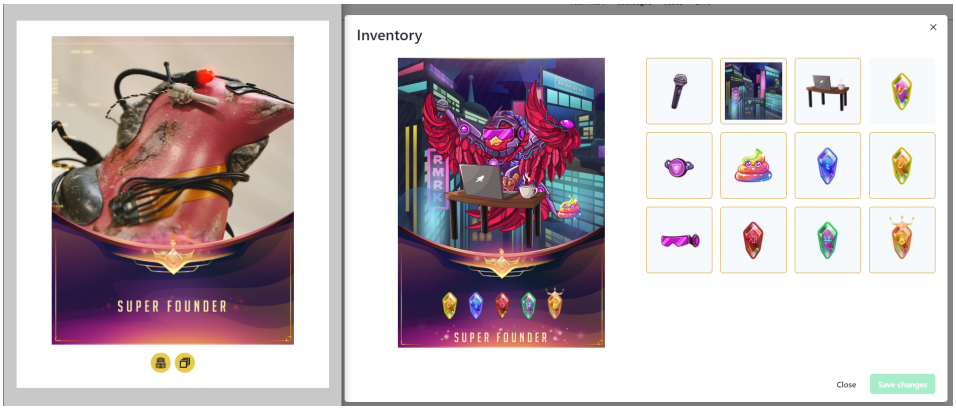
The RMRK standard is a set of rules and specifications for how to interpret special graffiti on Kusama called "remarks", accessible via the core `system` pallet in any Substrate chain.

The RMRK team has just launched the 2.0 version of the protocol, a set of "NFT legos", primitives that when put together allow a builder to compose an NFT system of arbitrary complexity without smart contracts. The "legos" are:

1. NFTs can own other NFTs, NFTs can equip other NFTs for visual change
2. NFTs can have multiple resources (different outputs based on context and resource priority)
3. NFTs can have on-chain emotes (reactions) for price discovery and social mechanics

4. NFTs have conditional rendering (e.g. show Mona Lisa as blushing if she got 50 kissy 😘 emoji)
5. NFTs can be governed by the community via fungible shareholder-tokens (fractionalization of NFTs)

The upcoming version 3.0 (Q1 2022) will be pallet and smart contract (EVM) versions of all RMRK 2.0 logic, and integration into partner chains for cheap and easy teleportation of non-fungibles across dozens of chains.



a multi resource NFT (gif of statue, and SVG-composable dynamic NFT in one) that can also equip other NFTs from withing its "inventory", from Kanaria

The RMRK team is collaborating closely with Unique network. RMRK's logic and technology will be deployed on Unique Network in the form of runtime upgrades (FRAME pallets).

Two marketplaces for RMRK-based NFTs exist with hundreds of projects already launched:

- Singular, the official marketplace
- Kodadot, a third party marketplace

Additionally, RMRK 2.0 functionality featuring composable, nested, multi-resource NFTs can be accessed and tested on the Kanaria platform.

For a full introduction into RMRK see the video explainer of RMRK, the video explainer of Kanaria (RMRK 2), and read the docs.

Efinity

Spearheaded by Enjin, the authors of Ethereum's ERC1155 standard and makers of the Enjin wallet and Unity plugin which allows easy implementation of NFTs into 3D games, Efinity is an NFT bridging chain coming to Kusama and Polkadot in 2022.

Their plan is to build a *paratoken* which would be a standard for token migration across different parachains in the Polkadot ecosystem, but also into and out of Ethereum and other EVM systems.

Moonbeam

Moonbeam and its Kusama counterpart Moonriver are full EVM deployments with Ethereum RPC endpoints.

This means that the entire toolkit offered to other EVM chains (stacks like Hardhat, Remix, Truffle, Metamask...) is available to Moonriver / Moonbeam users and developers, giving it a noticeable head start in attracting existing userbases.

There are several dozen high profile teams launching their products (or re-launching) on Moonriver / Moonbeam, however, it is important to note that Moonbeam is an EVM chain and will therefore suffer from the same limitations as any other EVM chain in regards to customization and feature-richness of NFTs.

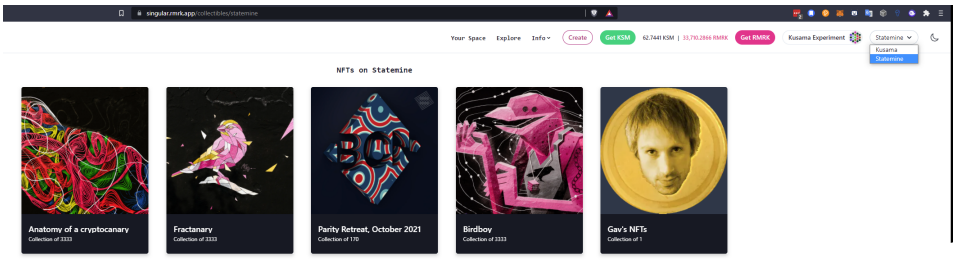
A notable advantage, however, is that Moonriver / Moonbeam is still a Substrate chain, meaning integration of custom pallets into the runtime is still possible, making NFT specific optimizations at the chain runtime level a good way to keep EVM compatibility of tools while at the same time optimizing storage and interactions for rich NFTs.

Uniques

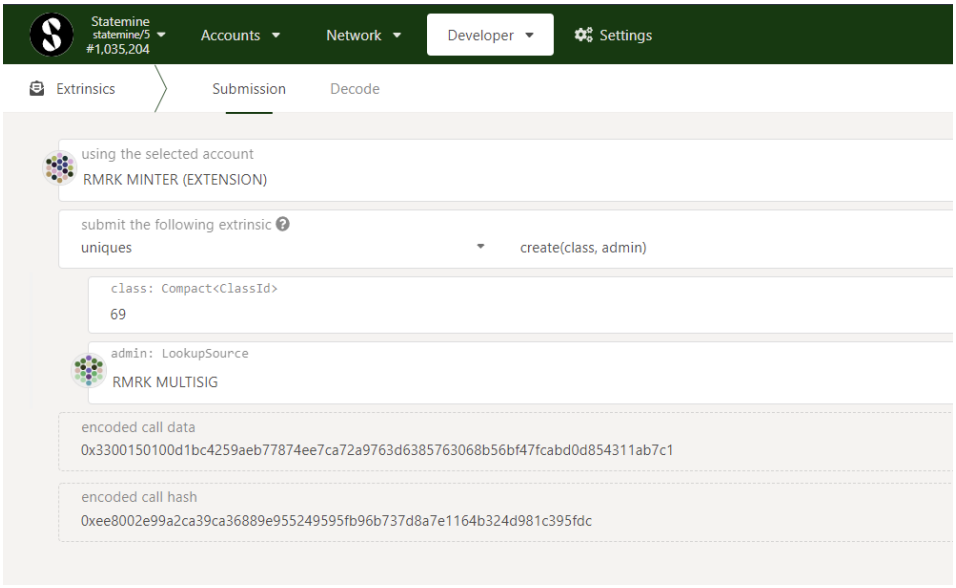
Uniques is a FRAME pallet deployed on the Statemint common good parachain. It is an implementation of the most basic kind of NFT - a data record referencing some metadata. This metadata reference is mutable until frozen, so NFTs and their classes (entities they are derived from) are mutable unless specifically made immutable by the issuer.

Uniques takes a very bare-bones approach on purpose, to keep the Statemint chain a simple balance-keeping chain for both fungible and non-fungibles.

Uniques NFTs can be viewed and interacted with on RMRK's Singular platform, by switching the top right menu from Kusama to Statemine.



The can also be interacted with directly through the extrinsics tab of Statemine:



More UIs are already being developed.

Bridging

Bridging to and from Substrate chains and EVM chains takes a lot of effort, but is a highly desired featured in the NFT industry. Merging the collector and customer base has big implications, so multiple projects are focusing on making this possible.

Apart from the aforementioned RMRK (Substrate-to-Substrate seamless teleportation natively with XCMP) and Efinity (Paratoken), the following efforts are underway:

- MyNFT: an EVM to EVM bridging effort.
- RMRK<->EVM Simplification bridge: a bridge developed during the RMRK hackathon for porting RMRK NFTs into simplified IOUs on EVM chains, primary deployment pending November 2022 on Moonriver
- RMRK<->EVM Full bridge: EVM version of RMRK 2.0 should be ready in December 2021, meaning a full migration of RMRK 2.0 NFTs from RMRK (Kusama) to Moonriver (and other EVMs) will become possible

Edit this page

*Last updated on **11/1/2021** by **Bruno Škvorc***

DOT

What is DOT?

DOT is the native token of the Polkadot network in a similar way that BTC is the native token of Bitcoin or Ether is the native token of the Ethereum blockchain.

The smallest unit of account in a Substrate network (Polkadot, Kusama, etc.) is the Planck (a reference to Planck Length, the smallest possible distance in the physical Universe). You can compare the Planck to Satoshis or Wei, while the DOT is like a bitcoin or an ether. Kusama tokens (KSM) are equal to 1e12 Planck, and Polkadot mainnet DOT is equal to 1e10 Planck.

Polkadot

Unit	Decimal Places	Example
Planck	0	0.0000000001
Microdot (uDOT)	4	0.0000010000
Millidot (mDOT)	7	0.0010000000
Dot (DOT)	10	1.0000000000
Million (MDOT)	16	1,000,000.00

> Note: This changed at block #1248_328. *Previously, DOT was denominated as equal to 1e12 Planck, just like Kusama. This denomination is deprecated, and, if necessary, referred to as "DOT (old)". See Redenomination of DOT for more details.*

Kusama

Unit	Decimal Places	Example
Planck	0	0.000000000001
Point	3	0.000000001000
MicroKSM (uKSM)	6	0.000001000000
MilliKSM (mKSM)	9	0.001000000000
KSM	12	1.000000000000

What are the uses of DOT?

DOT serves three key functions in Polkadot:

- to be used for governance of the network,

- to be staked for the operation of the network,
- to be bonded to connect a chain to Polkadot as a parachain.

DOT can also serve ancillary functions by virtue of being a transferrable token. For example, DOT stored in the Treasury can be sent to teams working on relevant projects for the Polkadot network.

> These concepts have been further explained in the video Usage of DOT and KSM on Polkadot and Kusama.

DOT for Governance

The first function of DOT is to entitle holders to control the governance of the platform. Some functions that are included under the governance mechanism include determining the fees of the network, the addition or removal of parachains, and exceptional events such as upgrades and fixes to the Polkadot platform.

Polkadot will enable any holder of DOT to participate in governance. For details on how holders can participate in governance, as well as their rights and responsibilities, see the governance page.

DOT for Consensus

DOT will be used to facilitate the consensus mechanism that underpins Polkadot. For the platform to function and allow for valid transactions to be carried out across parachains, Polkadot will rely on holders of DOT to play active roles. Participants will put their DOT at risk (via staking) to perform these functions. The staking of DOT acts as a disincentive for malicious participants who will be punished by the network by getting their DOT slashed. The DOT required to participate in the network will vary depending on the activity that is being performed, the duration the DOT will be staked for, and the total number of DOT staked.

DOT for Parachain Slot Acquisition

DOT will have the ability to be locked for a duration in order to secure a parachain slot in the network. The DOT will be reserved during the slot lease and will be released back to the account that reserved them after the duration of the lease has elapsed and the parachain is removed. You can learn more about this aspect by reading about the auctions that govern parachain slots.

Vesting

DOT may have a lock placed on them to account for vesting funds. Like other types of locks, these funds cannot be transferred but can be used in other parts of the protocol such as voting in governance or being staked as a validator or nominator.

Vesting funds are on a linear release schedule and unlock a constant number of tokens at each block. Although the tokens are released in this manner, it does not get reflected on-chain automatically since locks are lazy and require an extrinsic to update.

There are two ways that vesting schedules can be created.

- One way is as part of the genesis configuration of the chain. In the case of Polkadot and Kusama, the chain specification genesis script reads the state of the Polkadot Claims contract that exists on the Ethereum blockchain and creates vesting schedules in genesis for all the allocations registered as being vested.

- A second way is through an extrinsic type available in the Vesting pallet, `vested_transfer`. The vested transfer function allows anyone to create a vesting schedule with a transfer of funds, as long as the account for which the vesting schedule will be created does not already have one and the transfer moves at least `MinVestedTransfer` funds, which is specified as a chain constant.

Vesting schedules have three parameters, `locked`, `per_block`, and `starting_block`. The configuration of these three fields dictate the amount of funds that are originally locked, the slope of the unlock line and the block number for when the unlocking begins.

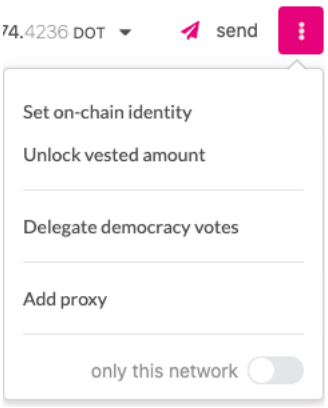
Lazy Vesting

Like simple payouts, vesting is *lazy*, which means that someone must explicitly call an extrinsic to update the lock that is placed on an account.

- The `vest` extrinsic will update the lock that is placed on the caller.
- The `vest_other` will update the lock that is placed on another "target" account's funds.

These extrinsics are exposed from the Vesting pallet.

If you are using Polkadot-JS, when there are DOT available to vest for an account, then you will have the ability to unlock DOT which has already vested from the Accounts page.



Calculating When Vesting DOT Will Be Available

Generally, you should be able to see from the Accounts by looking at your accounts and seeing when the vesting will finish. However, some DOT vest with "cliffs" - a single block where all the DOT are released, instead of vesting over time. In this case, you will have to query the chain state directly to see when they will be available (since technically, the vesting has not yet started - all of the vesting will occur in a single block in the future).

1. Navigate to the Chain State page on Polkadot-JS.
2. Query chain state for `vesting.vesting(ACCOUNT_ID)`
3. Note the `startingBlock` where the unlock starts, and how much DOT is unlocked per block (`perBlock`).
4. You will have to calculate the result into “human time". To do this, remember that there are approximately 14’400 blocks per day, and you can see what the latest block is shown on the Explorer page.

Obtaining Testnet DOT

DOT are required to make transactions on the Polkadot network. Testnet DOT do not have any value beside allowing you to experiment with the network.

Getting Westies

The current testnet is called Westend and you can obtain its native tokens by posting `!drip <WESTEND_ADDRESS>` in the Matrix chatroom #westend_faucet:matrix.org.

You can also make your own WNDs (testnet DOT) by becoming a validator.

Watch this video on how to get started on Westend

Westend

Unit	Decimal Places	Example
Planck	0	0.000000000001
Point	3	0.000000001000
MicroWND (uWND)	6	0.000001000000
MilliWND (mWND)	9	0.001000000000
WND	12	1.000000000000

Getting Rococo Tokens

Rococo is a parachain testnet. Tokens are given directly to teams working on parachains or exploring the cross consensus message passing aspects of this testnet. General users can obtain ROC by posting `!drip <ROCOCO_ADDRESS>` in the Matrix chatroom #rococo-faucet:matrix.org.

Learn more about Rococo on its dedicated wiki section.

Kusama Tokens

Unlike testnet DOT, Kusama tokens are not freely given away. Kusama tokens are available via the claims process (if you had DOT at the time of Kusama genesis) or through the Treasury. Alternatively, they can be obtained on the open market.

Polkadot Mainnet DOT

Polkadot Mainnet DOT are not freely given away. If you purchased DOT in the original 2017 offering, you may claim them via the Polkadot claims process. Alternatively, they are available on the open market.

Edit this page

*Last updated on **9/28/2021** by **Radha***

Security of the network

Shared security

Shared security, sometimes referred in documentation as *pooled security*, is one of the unique value propositions for chains considering to become a parachain and join the Polkadot network. On a high level, shared security means that all parachains that are connected to the Polkadot Relay Chain by leasing a parachain slot will benefit from the economic security provided by the Relay Chain validators.

The notion of shared security is different from interchain protocols that build on an architecture of bridges. For bridge protocols, each chain is considered sovereign and must maintain its own validator set and economic security. One concern in these protocols is on the point of scalability of security. For example, one suggestion to scale blockchains is that of *scale by altcoins,* which suggests that transaction volumes will filter down to lower market cap altcoins as the bigger ones fill their blocks. A major flaw in this idea is that the lower market cap coins will have less economic security attached, and be easier to attack. A real life example of a 51% attack occurred recently (Ethereum Classic attack on January 10), in which an unknown attacker double spent 219_500 ETC (~1.1 million USD). This was followed by two more 51% attacks on ETC.

Polkadot overcomes security scalability concerns since it gravitates all the economic incentives to the Relay Chain and allows the parachains to tap into stronger guarantees at genesis. Sovereign chains must expend much more effort to grow the value of their coin so that it is sufficiently secure against well-funded attackers.

Example

Let's compare the standard sovereign security model that exists on current proof-of-work (PoW) chains to that of the shared security of Polkadot. Chains that are secured by their own security model like Bitcoin, Zcash, Ethereum, and their derivatives all must bootstrap their own independent network of miners and maintain a competitive portion of honest hashing power. Since mining is becoming a larger industry that increasingly centralizes on key players, it is becoming more real that a single actor may control enough hash power to attack a chain.

This means that smaller chains that cannot maintain a secure amount of hash power on their networks could potentially be attacked by a large mining cartel at the simple whim of redirecting its hash power away from Bitcoin and toward a new and less secure chain. 51% attacks are viable today with attacks having been reported on Ethereum Classic (see above), Verge, Bitcoin Gold, and other cryptocurrencies.

On Polkadot, this disparity between chain security will not be present. When a parachain connects to Polkadot, the Relay Chain validator set become the securers of that parachain's state transitions. The parachain will only have the overhead of needing to run a few collator nodes to keep the validators informed with the latest state transitions and proofs/witness. Validators will then check these for the parachains to which they are assigned. In this way, new parachains instantly benefit from the overall security of Polkadot even if they have just been launched.

FAQ

Is security correlated to the number of validators? What about the number of parachains?

Security is independent of the number of parachains that are connected to the Polkadot Relay Chain. The correlation of security and the number of validators exists as the higher number of validators will give the network stronger decentralization properties and make it harder to try to take down. However, the biggest indicator of the security of the network is the economic signal of the number of DOT that are bonded and staked. The greater the number of DOT staked by honest validators and nominators, the higher the minimum amount of DOT an attacker would need to acquire a validator slot.

Will parachains ever need their own security? In what scenarios do parachains need their own security?

Most parachains will not need to worry about their own security, since all state transitions will be secured by the Polkadot Relay Chain validator set. However, in some cases (which are considered more experimental), parachains may require their own security. In general, these cases will revolve around lack of data available to Relay Chain validators.

One example is if the state transition function is some succinct or zero-knowledge proof, the parachain would be responsible for keeping its data available as the Relay Chain won't have it. Additionally, for chains with their own consensus, like the one that enables fast payments on Blink Network, there would probably need to be a Byzantine agreement between stakers before a parachain block is valid. The agreement would be necessary because the data associated with the fast consensus would be unknown to Relay Chain validators.

Edit this page

*Last updated on **10/23/2021** by **Danny Salman***

Polkadot Consensus

Why do we need Consensus?

Consensus is a method for coming to agreement over a shared state. In order for the state of the blockchain to continue to build and move forward, all nodes in the network must agree and come to consensus. It is the way that the nodes in a decentralized network are able to stay synced with each other. Without consensus for the decentralized network of nodes in a blockchain, there is no way to ensure that the state one node believes is true will be shared by the other nodes. Consensus aims to provide the *objective* view of the state amid participants who each have their own *subjective* views of the network. It is the process by which these nodes communicate and come to agreement, and are able to build new blocks.

What are PoW and PoS?

Proof of Work (PoW) and Proof of Stake (PoS) have been inaccurately used as short hand to refer to consensus mechanisms of blockchains, but that does not capture the full picture. PoW is the method for agreeing on a block author and part of the fuller Nakamoto consensus that also encompasses a chain selection algorithm (longest chain rule in Bitcoin). Similarly, PoS is a set of rules for selecting the validator set and does not specify a chain selection rule or how a chain might reach finality. PoS algorithms have traditionally been paired with an algorithm for coming to Byzantine agreement between nodes. For example, Tendermint is a practical Byzantine fault tolerant algorithm that uses PoS as its validator set selection method.

Why not Proof of Work?

Although simple and effective in coming to a decentralized consensus on the next block producer, proof of work with Nakamoto consensus consumes an incredible amount of energy, has no economic or provable finality, and has no effective strategy in resisting cartels.

Nominated Proof of Stake

In traditional PoS systems, block production participation is dependent on token holdings as opposed to computational power. While PoS developers usually have a proponent for equitable participation in a decentralized manner, most projects end up proposing some level of centralized operation, where the number of validators with full participation rights is limited. These validators are often seen to be the most wealthy, and, as a result, influence the PoS network as they are the most staked. Usually, the number of candidates to maintain the network with the necessary knowledge (and equipment) is limited; this can directly increase operational costs as well. Systems with a large number of validators tend to form pools to decrease the variance of their revenue and profit from economies of scale. These pools are often off-chain.

A way to alleviate this is to implement pool formation on-chain and allow token holders to vote [with their stake] for validators to represent them.

Polkadot uses NPoS (Nominated Proof-of-Stake) as its mechanism for selecting the validator set. It is designed with the roles of **validators** and **nominators**, to maximize chain security. Actors who are interested in maintaining the network can run a validator node.

Validators assume the role of producing new blocks in BABE, validating parachain blocks, and guaranteeing finality. Nominators can choose to back select validators with their stake. Nominators can approve candidates that they trust and back them with their tokens.

Probabilistic vs. Provable Finality

A pure Nakamoto consensus blockchain that runs PoW is only able to achieve the notion of *probabilistic finality* and reach *eventual consensus*. Probabilistic finality means that under some assumptions about the network and participants, if we see a few blocks building on a given block, we can estimate the probability that it is final. Eventual consensus means that at some point in the future, all nodes will agree on the truthfulness of one set of data. This eventual consensus may take a long time and will not be able to be determined how long it will take ahead of time. However, finality gadgets such as GRANDPA (GHOST-based Recursive ANcestor Deriving Prefix Agreement) or Ethereum's Casper FFG (the Friendly Finality Gadget) are designed to give stronger and quicker guarantees on the finality of blocks - specifically, that they can never be reverted after some process of Byzantine agreements has taken place. The notion of irreversible consensus is known as *provable finality.*

In the GRANDPA paper, it is phrased in this way:

> We say an oracle A in a protocol is *eventually consistent* if it returns the same value to all participants after some unspecified time.

Hybrid Consensus

There are two protocols we use when we talk about the consensus protocol of Polkadot, GRANDPA and BABE (Blind Assignment for Blockchain Extension). We talk about both of these because Polkadot uses what is known as *hybrid consensus*. Hybrid consensus splits up the finality gadget from the block production mechanism.

This is a way of getting the benefits of probabilistic finality (the ability to always produce new blocks) and provable finality (having a universal agreement on the canonical chain with no chance for reversion) in Polkadot. It also avoids the corresponding drawbacks of each mechanism (the chance of unknowingly following the wrong fork in probabilistic finality, and a chance for "stalling" - not being able to produce new blocks - in provable finality). By combining these two mechanisms, Polkadot allows for blocks to be rapidly produced, and the slower finality mechanism to run in a separate process to finalize blocks without risking slower transaction processing or stalling.

Hybrid consensus has been proposed in the past. Notably, it was proposed (now defunct) as a step in Ethereum's transition to proof of stake in EIP 1011, which specified Casper FFG.

Block Production: BABE

BABE (Blind Assignment for Blockchain Extension) is the block production mechanism that runs between the validator nodes and determines the authors of new blocks. BABE is comparable as an algorithm to Ouroboros Praos, with some key differences in chain selection rule and slot time adjustments. BABE assigns block production slots to validators according to stake and using the Polkadot randomness cycle.

Validators in Polkadot will participate in a lottery in every slot that will tell them whether or not they are the block producer candidate for that slot. Slots are discrete units of time, nominally 6 seconds in length. Because of this randomness mechanism, multiple validators could be candidates for the same slot. Other times, a slot could be empty, resulting in inconsistent block time.

Multiple Validators per Slot

When multiple validators are block producer candidates in a given slot, all will produce a block and broadcast it to the network. At that point, it's a race. The validator whose block reaches most of the network first wins. Depending on network topology and latency, both chains will continue to build in some capacity, until finalization kicks in and amputates a fork. See Fork Choice below for how that works.

No Validators in Slot

When no validators have rolled low enough in the randomness lottery to qualify for block production, a slot can remain seemingly blockless. We avoid this by running a secondary, round-robin style validator selection algorithm in the background. The validators selected to produce blocks through this algorithm always produce blocks, but these *secondary* blocks are ignored if the same slot also produces a primary block from a VRF-selected validator. Thus, a slot can have either a *primary* or a *secondary* block, and no slots are ever skipped.

For more details on BABE, please see the BABE paper.

BADASS BABE: SASSAFRAS

SASSAFRAS (Semi Anonymous Sortition of Staked Assignees For Fixed-time Rhythmic Assignment of Slots) (aka SASSY BABE or BADASS BABE), is an extension of BABE and acts as a constant-time block production protocol. This approach tries to address the shortcomings of BABE by ensuring that exactly one block is produced with time-constant intervals. The protocol utilizes zk-SNARKs to construct a ring-VRF and is a work in progress. This section will be updated as progress ensues.

Finality Gadget: GRANDPA

GRANDPA (GHOST-based Recursive ANcestor Deriving Prefix Agreement) is the finality gadget that is implemented for the Polkadot Relay Chain.

It works in a partially synchronous network model as long as 2/3 of nodes are honest and can cope with 1/5 Byzantine nodes in an asynchronous setting.

A notable distinction is that GRANDPA reaches agreements on chains rather than blocks, greatly speeding up the finalization process, even after long-term network partitioning or other networking failures.

In other words, as soon as more than 2/3 of validators attest to a chain containing a certain block, all blocks leading up to that one are finalized at once.

Protocol

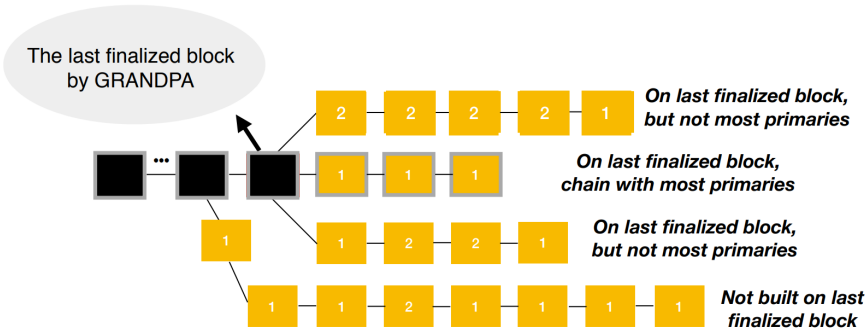
Please refer to the GRANDPA paper for a full description of the protocol.

Implementation

The Substrate implementation of GRANDPA is part of Substrate Frame.

Fork Choice

Bringing BABE and GRANDPA together, the fork choice of Polkadot becomes clear. BABE must always build on the chain that has been finalized by GRANDPA. When there are forks after the finalized head, BABE provides probabilistic finality by building on the chain with the most primary blocks.



Longest chain with most primaries on last finalized GRANDPA block

In the above image, the black blocks are finalized, and the yellow blocks are not. Blocks marked with a "1" are primary blocks; those marked with a "2" are secondary blocks. Even though the topmost chain is the longest chain on the latest finalized block, it does not qualify because it has fewer primaries at the time of evaluation than the one below it.

Comparisons

Nakamoto consensus

Nakamoto consensus consists of the longest chain rule using proof of work as its Sybil resistance mechanism and leader election.

Nakamoto consensus only gives us probabilistic finality. Probabilistic finality states that a block in the past is only as safe as the number of confirmations it has, or the number of blocks that have been built on top of it. As more blocks are built on top of a specific block in a Proof of Work chain, more computational work has been expended behind this particular chain. However, it does not guarantee that the chain containing the block will always remain the agreed-upon chain, since an actor with unlimited resources could potentially build a competing chain and expend enough computational resources to create a chain that did not contain a specific block. In such a situation, the longest chain rule employed in Bitcoin and other proof of work chains would move to this new chain as the canonical one.

PBFT / Tendermint

Please see the relevant section in the Cosmos comparison article.

Casper FFG

The two main differences between GRANDPA and Casper FFG are:

- in GRANDPA, different voters can cast votes simultaneously for blocks at different heights
- GRANDPA only depends on finalized blocks to affect the fork-choice rule of the underlying block production mechanism

Resources

- BABE paper - The academic description of the BABE protocol.
- GRANDPA paper - The academic description of the GRANDPA finality gadget. Contains formal proofs of the algorithm.
- Rust implementation - The reference implementation and the accompanying Substrate pallet.
- Block Production and Finalization in Polkadot - An explanation of how BABE and GRANDPA work together to produce and finalize blocks on Kusama, with Bill Laboon.
- Block Production and Finalization in Polkadot: Understanding the BABE and GRANDPA Protocols - An academic talk by Bill Laboon, given at MIT Cryptoeconomic Systems 2020, describing Polkadot's hybrid consensus model in-depth.

Edit this page

*Last updated on **10/23/2021** by **Danny Salman***

Nominator

Nominators secure the Relay Chain by selecting good validators and staking DOT.

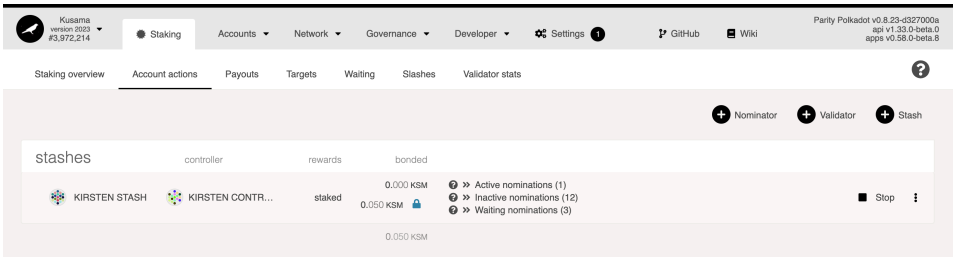
You may have an account with DOT and want to earn fresh DOT. You could do so as validator, which requires a node running 24/7. If you do not have such node or do not want to bother, you can still earn DOT by nominating one or more validators.

By doing so, you become a nominator for the validator(s) of your choice. Pick your validators carefully - if they do not behave properly, they will get slashed and you will lose DOT as well. However, if they do follow the rules of the network, then you can share in staking rewards that they generate.

While your DOT are staked by nominating a validator, they are 'locked' (bonded). You can receive new DOT in your account but you cannot stake as validator or transfer DOT away from your account. You can un-nominate at any time to stop staking your funds. Keep in mind that the un-nomination is effective in the next era, and that un-nominating does not automatically unbond your funds. There is an unbonding period of 7 days on Kusama and 28 days on Polkadot before bonded funds can be transferred after issuing an unbond transaction.

Active vs. Inactive Nomination

When you go to the Account actions under staking page, you should see your bonded accounts and nomination status. If not, you can follow this guide to configure it first. Your nominations will be effective in the next era; eras are roughly 6 hours on Kusama and 24 hours on Polkadot.



Suppose you have nominated five validator candidates, and three out of five were elected to the active validator set, then you should see two of your nominations as "waiting", and most likely one as "active" and the rest as "inactive". Active or inactive nomination means your nominated validators have been elected to be in the validator set, whereas waiting means they did not get elected. Generally, you will only have a single validator have an active nomination, which means that you are directly supporting it with your stake this era and thus potentially receiving staking rewards. Inactive nominators were validators that were elected for this era but which you are not actively supporting. Every era, a new election will take place and you may be assigned a different active nomination from among the validators you have selected.

If you are committing a very large amount of stake, then you may have more than one active nomination. However, the election algorithm attempts to minimize this situation, and it should not occur often, so you should almost always see only a single active nomination per era. See the section on Phragmén optimization for more details.

Required Minimum Stake

Due to the way the Phragmen algorithm generates the solution set, and due to the fact that the solution set must fit in a single block, in some eras, a minimum number of DOT will be required to nominate with in order to receive staking rewards.

This parameter can be updated via on-chain governance and the most recent and up to date version can be found on chain state (select **state query > staking > minimumNominatorBond**)

Oversubscribed Validators

Validators can only pay out to a certain number of nominators per era. This is currently set to 256, but can be modified via governance. If more than 256 nominators nominate the same validator, it is "oversubscribed", and only the top 256 staked nominators (ranked by amount of stake) are paid rewards. Other nominators will receive no rewards for that era, although their stake will still be used to calculate entry into the active validator set.

Although it is difficult to determine exactly how many nominators will nominate a given validator in the next era, one can estimate based on the current number of nominators. A validator with only 5 nominators in this era, for instance, is unlikely to have more than 256 in the next era. An already-oversubscribed validator with 1000 nominators this era, however, is very likely to be oversubscribed in the next era as well.

The Election Solution Set

Determining which validators are in the active set and which nominators are nominating them creates a very large graph mapping nominators to their respective validators. This "solution set" is computed off-chain and submitted to the chain, which means it must fit in a single block. If there are a large number of nominators, this means that some nominators must be eliminated. Currently, nominators are sorted by amount of DOT staked and those with more DOT are prioritized. This means that if you are staking with a small amount of DOT, you may not receive rewards. This minimal amount is dynamic based on the number of validators, number of nominators, amount nominated, and other factors.

Receiving Rewards

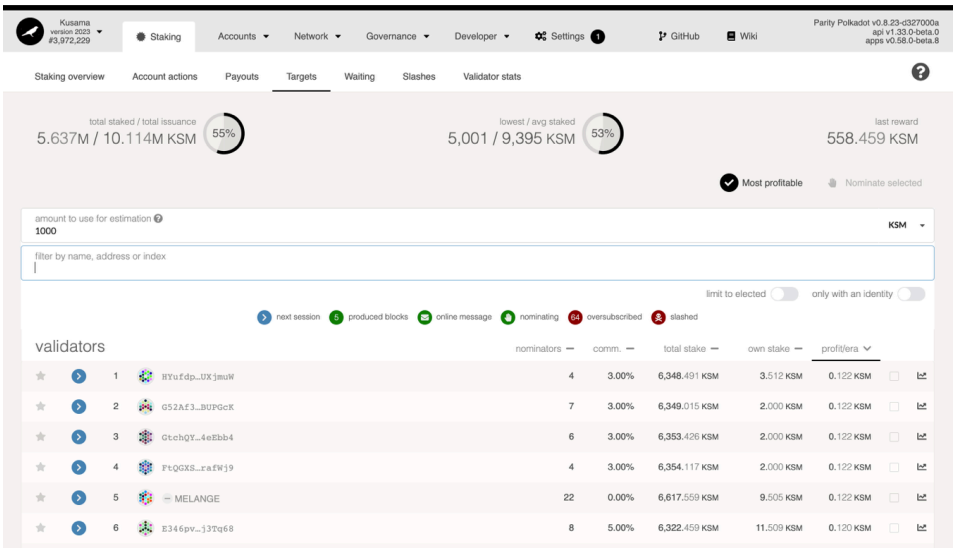
As long as you have nominated more than one validator candidate, at least one of them got elected, and you are nominating with enough stake to get into the solution set, your bonded stake will be fully distributed to one or more validators. That being said, you may not receive rewards if you nominated very few validator candidates and no one got elected, or your stake is small and you only selected oversubscribed validators, or the validator you are nominating has 100% commission. It is generally wise to choose as many trustworthy validators as you can (up to 16) to reduce the risk of none of your nominated validators being elected.

Rewards are *lazy* - somebody must trigger a payout for a validator for rewards to go all of the validator's nominators. Any account can do this, although in practice validator operators often do this as a service to their nominators. See the page on Simple Payouts for more information and instructions for claiming rewards.

What to Take Into Consideration When Nominating

One thing to keep in mind as a nominator is the validator's commission. The commission is the percentage of the validator reward which is taken by the validator before the rewards are split among the nominators. As a nominator, you may think that the lowest commission is best. However, this is not always true. Validators must be able to run at

break-even in order to sustainably continue operation. Independent validators that rely on the commission to cover their server costs help to keep the network decentralized. Commission is just one piece of the puzzle that you should consider when picking validators to nominate.



As a nominator, if you only want to know the profit each validator made for each era, you can go to the Targets section under the staking page by inputting the number of tokens you would like to stake to check it. Then, nominate those who have a higher profit. However, that does not guarantee the right way to evaluate the validators' overall performance.

It is worth taking into consideration "own stake" of a validator. This refers to the quantity of DOT the validator has put up at stake themselves. A higher "own stake" amount can be considered as having more "skin in the game". This can imply increased trustworthiness. However, a validator not having a large amount of "own stake" is not automatically untrustworthy, as the validator could be nominating from a different address.

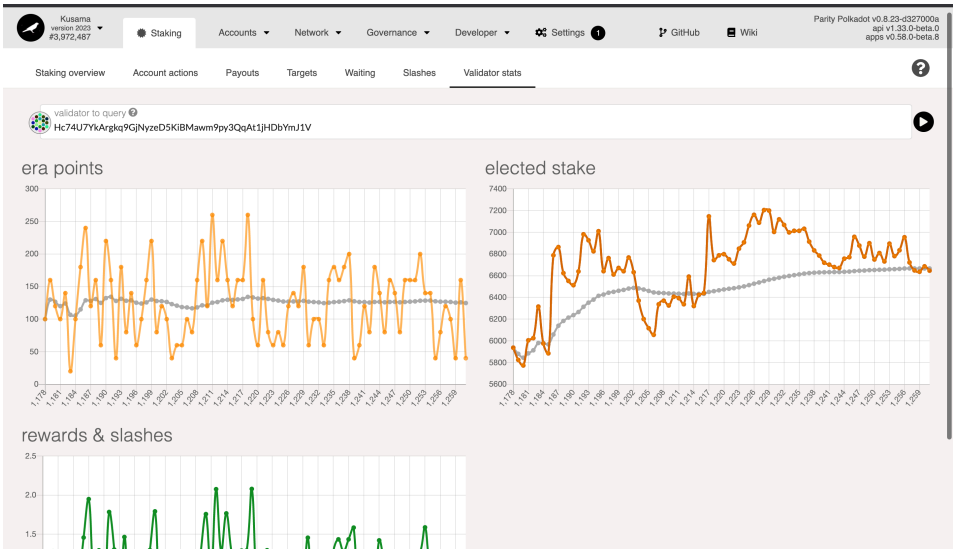
Filter Out Validators With Undesirable Traits

On the Targets page, you can filter out validators that have traits that may indicate an issue with you nominating them. You can turn these filters off and on to help narrow down which validators you should nominate. It is important to note that none of these traits are necessarily "bad"; however, depending on your validator selection methodology, they may be characteristics that you would be interested in filtering out.

- **Single from operator** - Do not show groups of validators run by a single operator.
- **No 20%+ comm** - Do not show any validators with a commission of 20% or higher.
- **No at capacity** - Do not show any validators who are currently operating at capacity (i.e., could potentially be oversubscribed).
- **Recent payouts** - Only show validators that have recently caused a payout to be issued. Note that anyone can cause a payout to occur; it does not have to be the operator of a validator.
- **Only elected** - Only show validators that are currently in the active set (i.e., they have been elected to produce blocks this era).
- **Only with an identity** - Only show validators that have set an identity. Note that this identity does not have to be verified by a registrar for the validator to show up in the list.

Review Your Validators' History

How the validator acted in the past may be a good indicator of how they will act in the future. An example of problematic behavior would be if a validator is regularly offline, their nominators most likely would get fewer rewards than others. More importantly, when many validators are unreachable, those nominators who staked with them will be slashed.



Thus, to be a smart nominator, it would be better to query their histories to see statistics such as blocks produced, rewards and slashes, and identity (if they have it set). Moreover, a nominator should do comprehensive research on their validator candidates - they should go over the validators' websites to see who they are, what kind of infrastructure setup they are using, reputation, the vision behind the validator, and more.

Be Aware of The Risks of Single Operators with Multiple Validators

Recall that slashing is an additive function; the more validators that are offline or equivocate in a given session, the harsher the penalties. Since validators that are controlled by a single entity are more at risk of a "synchronized" failure, nominating them implies a greater risk of having a large slash of your nominated funds. Generally, it is safer to nominate validators whose behavior is independent from others in as many ways as possible (different hardware, geographic location, owner, etc.).

Avoiding Oversubscribed Validators

If you are not nominating with a large number of DOTs, you should try to avoid oversubscribed validators. It is not always easy to calculate if the validator selected will be oversubscribed in the next session; one way to avoid choosing potentially oversubscribed validators is to filter out any that are at capacity on the Targets page.

Finally, if you have a very small amount of DOTs, you may not be able to have your nomination fit into the election set. The nominator to validator mapping has to fit in a single block, and if there are too many nominators, the lowest-staked nominations will be dropped. This value is obviously dynamic and will vary over time. If you review the lowest amount of nominations that are occurring on current validators, you can get a good idea of how many DOTs will likely be necessary to have your nomination earn you rewards. You can read the blog post "Polkadot Staking: An Update" for more details.

> These concepts have been further explained in the Why Nominate on Polkadot & Kusama video and What to Consider when Nominating Validators on Polkadot and Kusama and Nominating/Staking on Polkadot and Kusama

Guides

- Be a Nominator (Polkadot) - Guide on nominating on the Kusama canary network.
- Stop Being a Nominator (all networks) - Guide on stopping nominations and withdrawing tokens.

Edit this page

*Last updated on **10/1/2021** by **Radha***

Validator

Validators secure the Relay Chain by staking DOT, validating proofs from collators and participating in consensus with other validators.

These participants play a crucial role in adding new blocks to the Relay Chain and, by extension, to all parachains. This allows parties to complete cross-chain transactions via the Relay Chain. Parachain validators participate in some form of off-chain consensus, and submit candidate receipts to the tx pool for a block producer to include on-chain. The Relay Chain validators guarantee that each parachain follows its unique rules and can pass messages between shards in a trust-free environment.

Validators perform two functions:

1. **Verifying** that the information contained in an assigned set of parachain blocks is valid (such as the identities of the transacting parties and the subject matter of the contract).
2. **Participating** in the consensus mechanism to produce the Relay Chain blocks based on validity statements from other validators. Any instances of non-compliance with the consensus algorithms result in punishment by removal of some or all of the validator's staked DOT, thereby discouraging bad actors. Good performance, however, will be rewarded, with validators receiving block rewards (including transaction fees) in the form of DOT in exchange for their activities.

Guides

- How to Validate on Polkadot - Guide on how to set up a validator on the Polkadot live network.
- Validator Payout Overview - A short overview on how the validator payout mechanism works.
- How to run your validator as a systemd process - Guide on running your validator as a `systemd` process so that it will run in the background and start automatically on reboots.
- How to Upgrade your Validator - Guide for securely upgrading your validator when you want to switch to a different machine or begin running the latest version of client code.
- How to Use Validator Setup - Guide on how to use Polkadot / Kusama validator setup.

Other References

- How to run a Polkadot node (Docker)
- A Serverless Failover Solution for Web3.0 Validator Nodes - Blog that details how to create a robust failover solution for running validators.
- VPS list
- Polkadot Validator Lounge - A place to chat about being a validator.
- Slashing Consequences - Learn more about slashing consequences for running a validator node.
- Why You Should be A Validator on Polkadot and Kusama
- Roles and Responsibilities of a Validator
- Validating on Polkadot - An explanation of how to validate on Polkadot, with Joe Petrowski and David Dorgan of Parity Technologies, along with Tim Ogilvie from

Staked.

Security / Key Management

- Validator Security Overview

Monitoring Tools

- PANIC for Polkadot - A monitoring and alerting solution for Polkadot / Kusama node
- Polkadot Telemetry Service - Network information, including what nodes are running on a given chain, what software versions they are running, and sync status.
- Other Useful Links

Validator Stats

- HashQuark Staking Strategy - The HashQuark staking strategy dashboard helps you choose the optimal set-up to maximize rewards, and provides other useful network monitoring tools.
- Polkastats - Polkastats is a cleanly designed dashboard for validator statistics.
- YieldScan - Staking yield maximization platform, designed to minimize effort.
- Subscan Validators Page - Displays information on the current validators - not as tailored for validators as the other sites.

Edit this page

*Last updated on **9/17/2021** by **Danny Salman***

Collator

Collators maintain parachains by collecting parachain transactions from users and producing state transition proofs for Relay Chain validators. In other words, collators maintain parachains by aggregating parachain transactions into parachain block candidates and producing state transition proofs for validators based on those blocks.

Collators maintain a full node for the Relay Chain and a full node for their particular parachain; meaning they retain all necessary information to be able to author new blocks and execute transactions in much the same way as miners do on current PoW blockchains. Under normal circumstances, they will collate and execute transactions to create an unsealed block and provide it, together with a proof of state transition, to one or more validators responsible for proposing a parachain block.

Unlike validators, collator nodes do not secure the network. If a parachain block is invalid, it will get rejected by validators. Therefore the assumption that having more collators is better or more secure is not correct. On the contrary, too many collators may slow down the network. The only nefarious power collators have is transaction censorship. To prevent censorship, a parachain only needs to ensure that there exist some neutral collators - but not necessarily a majority. Theoretically, the censorship problem is solved with having just one honest collator.

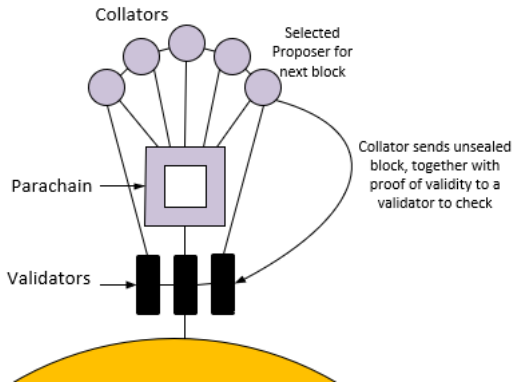
XCM

Collators are a key element of the XCM (Cross-Consensus Message Passing Format). By being full nodes of the Relay Chain, they are all aware of each other as peers. This makes it possible for them to send messages from parachain A to parachain B.

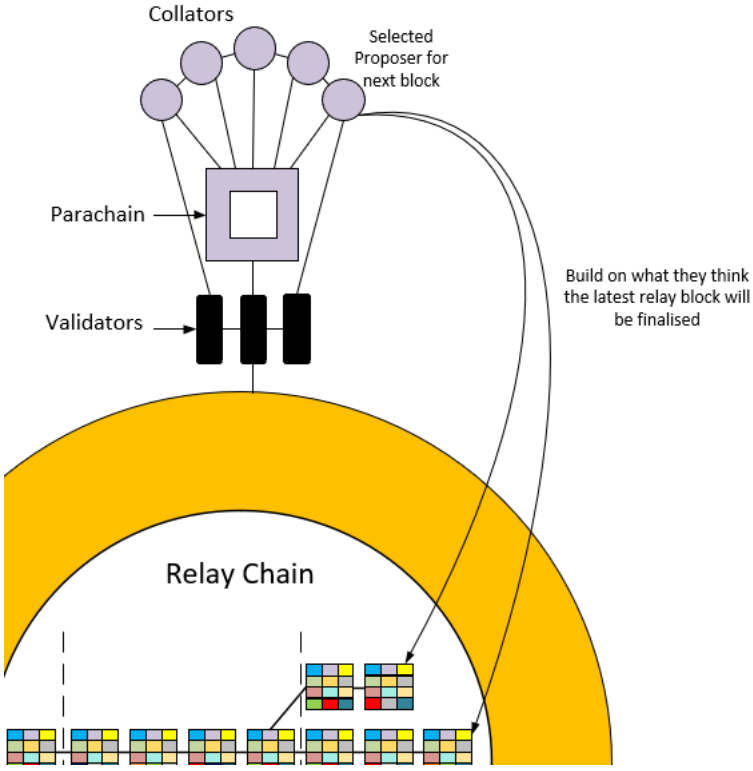
Taking the case for one Parachain

A start of a new block candidate is initiated with a block creation time. The collator aggregates all new transactions at the end of the process. When doing so, the collator signs the *parachain block candidate* and produces state transition proofs, which are a summary of the final account balances caused by the transactions in the candidate block. The collator relays the candidate block and state transition proofs to the validators on-chain. The validators verify the transactions within the parachain block candidate. Upon verification, and if all is well, the validator shares the candidate block with the Relay Chain.

Parachain block candidates are collected together and a *Relay Chain block candidate* is produced.



The validators on the network will try to reach a consensus on the Relay Chain block candidate. Upon reaching consensus, the now validated Relay Chain block candidate is shared with the validators and collators, and the process repeats for new transactions. A collator cannot continue building blocks on a parachain until the block candidate they proposed to the Relay Chain validators have been validated.



A block is produced every 6 seconds.

Collators in the Wild

Blockchains that are built using Substrate are unable to hook onto the Relay Chain on their own. The Parity team built the Cumulus library to address this. Collators are being used on the Rococo testnet, and you can learn more about how they are used with Cumulus via the Cumulus repository. More information can be found under the Cumulus section on the build parachain page.

Guides and Tools

- Workshop covering Cumulus and Collators
- Rococo tesnet guide
- polkadot-launch - a tool to quickly spin up a local Polkadot testnet based on some parameters like number of parachains, collator setup, etc.

Edit this page

*Last updated on **10/23/2021** by **Danny Salman***

Governance

Polkadot uses a sophisticated governance mechanism that allows it to evolve gracefully overtime 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

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 can 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 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 from 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 that has no proposals (is empty), and proposals are 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 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 the following table for reference.

Entity	**Metric**
Public	Positive Turnout Bias (Super-Majority Approve)
Council (Complete agreement)	Negative Turnout Bias (Super-Majority Against)
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 with lower turnout.

```
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 into 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

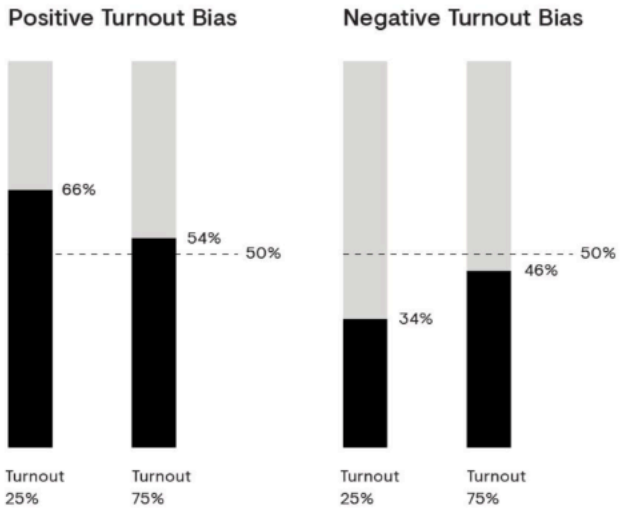
The maximum number of "doublings" of the lock period is set to 6 (and thus 32 lock periods in total), and one lock period equals 28 days on Polkadot and 8 days on Kusama. Only doublings are allowed; you cannot lock for, say, 24 periods and increase your conviction by 5.5, for instance.

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 a 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 a 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 the 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 several 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.

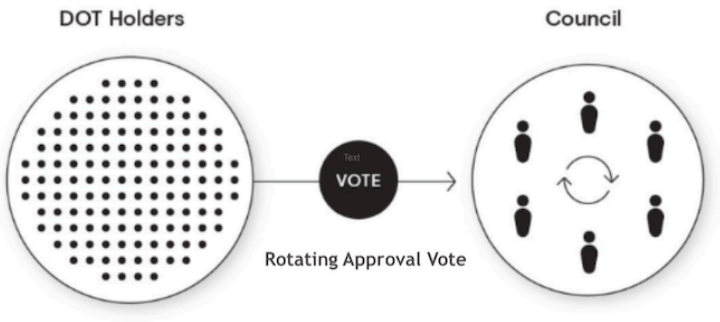
Blacklisting

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					
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 candidates 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 a 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.

Edit this page

*Last updated on **9/17/2021** by **Danny Salman***

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.

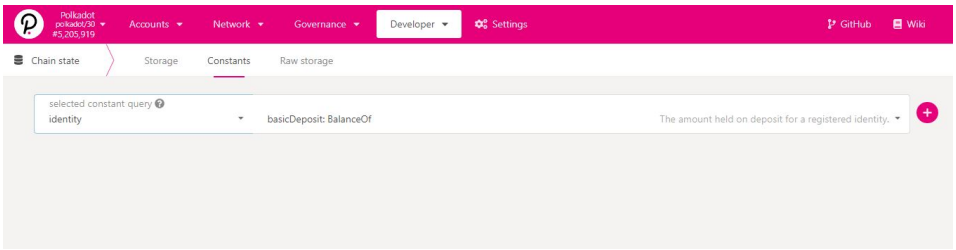
Setting an Identity

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).

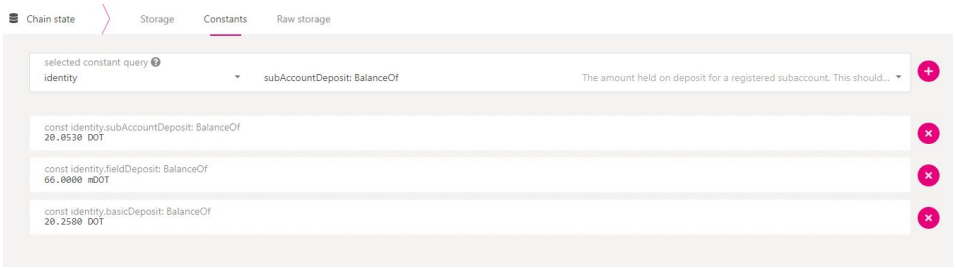
Users must reserve funds in a bond to store their information on chain: 20.258, and 0.066 per each field beyond the legal name. These funds are *locked*, not spent - they are returned when the identity is cleared.

These amounts can also be extracted by querying constants through the Chain state constants tab on polkadot.js/apps.

First, select `identity` as the `selected constant query`.

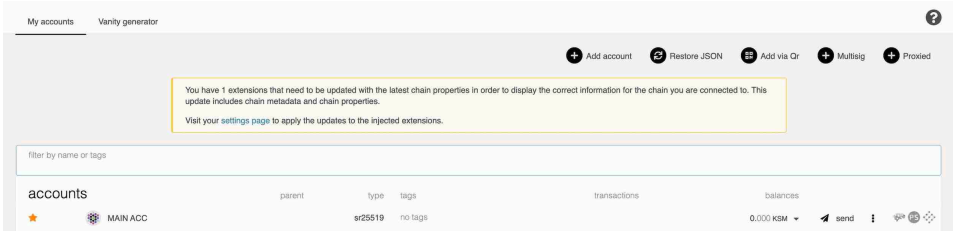


Then on the right-hand side, you can select the constants that you would like to view and add them onto the webpage by clicking the "plus" icon at the end of the bar.

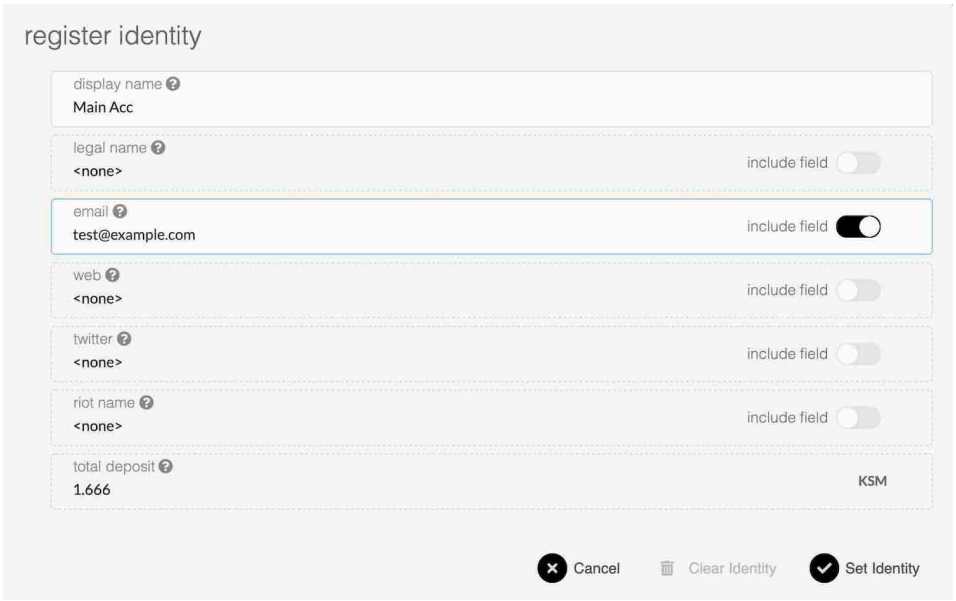


Each field can store up to 32 bytes of information, so the data must be less than that. When inputting the data manually through the Extrinsics UI, a UTF8 to bytes converter can help.

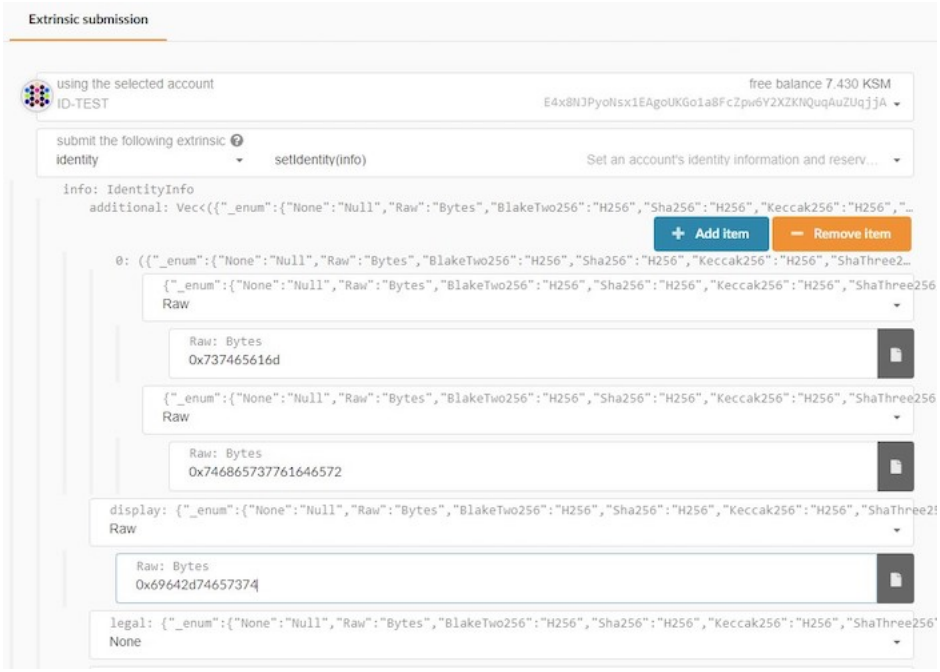
The easiest way to add the built-in fields is to click the gear icon next to your account and select "Set on-chain identity".



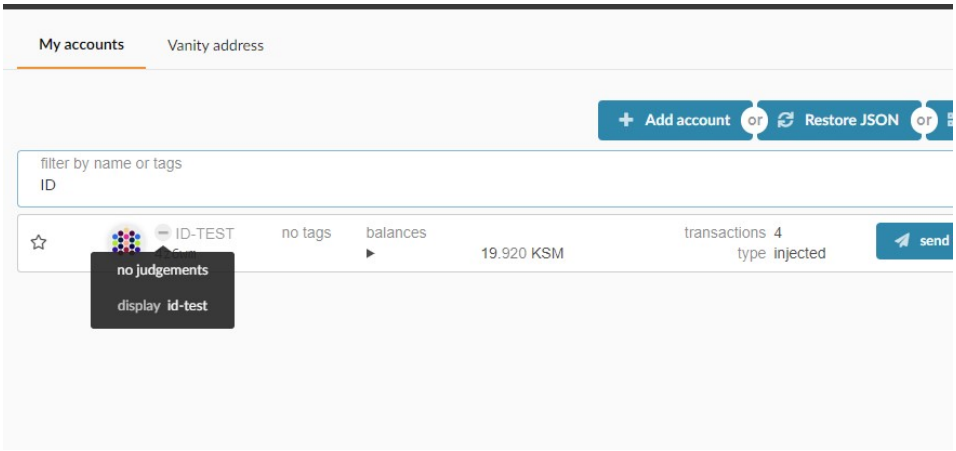
A popup will appear, offering the default fields.



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.

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 the complete removal of the identity.

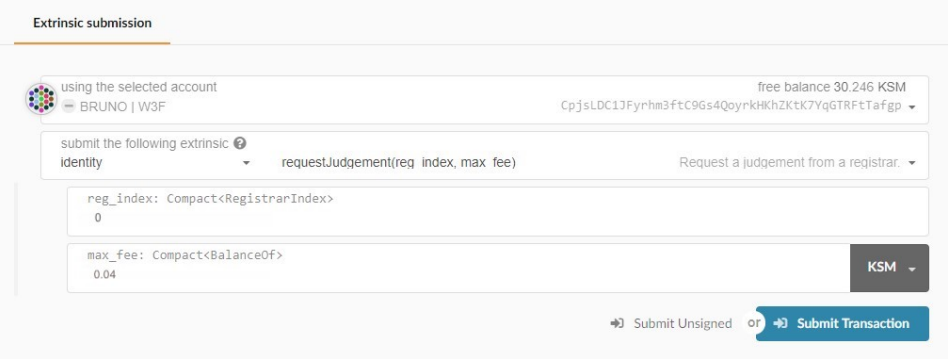
Registrars gain trust by performing proper due diligence and would presumably be replaced for issuing faulty judgements.

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.

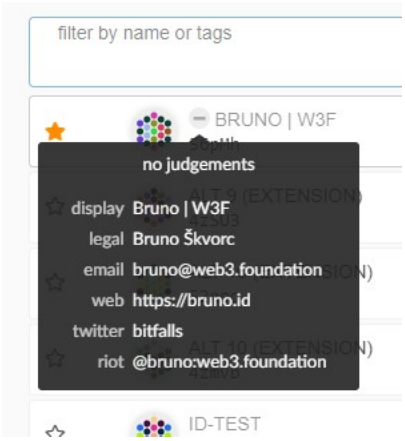
If you don't know which registrar to pick, first check the available registrars by going to "Chain State UI" and selecting `identity.registrars()` to get the full list.

Requesting a Judgement

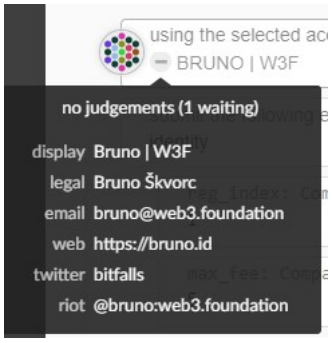
Requesting judgement follows the same process regardless of whether you're on the Kusama or Polkadot networks. Select one of the registrars from the query you made above.



This will make your identity go from unjudged:

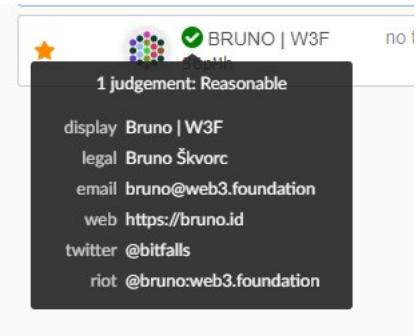


To "waiting":



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.

Cancelling a Judgement

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.

Registrars

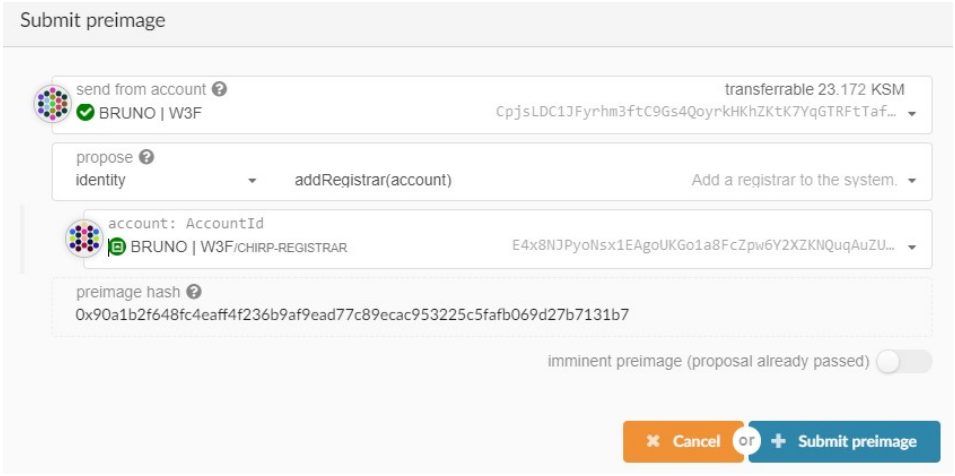
Registrars can set a fee for their services and limit their attestation to certain fields. For example, a registrar could charge 1 DOT to verify one's legal name, email, and GPG key. When a user requests judgement, they will pay this fee to the registrar who provides the judgement on those claims. Users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

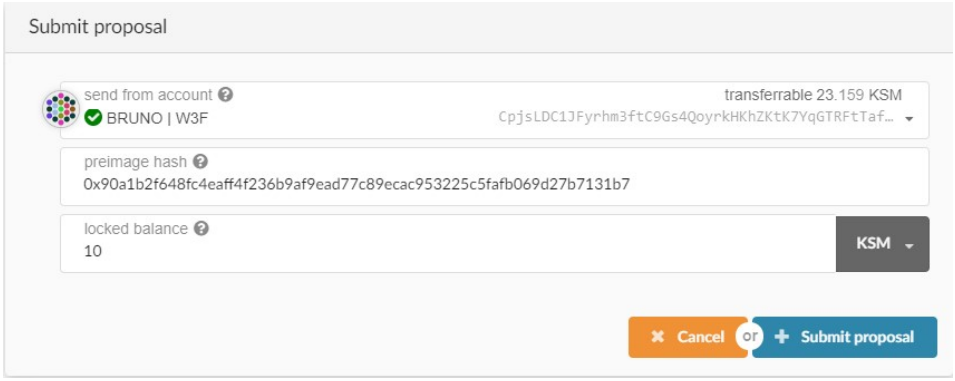
Here's how to submit a proposal to become a registrar:

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 20.258 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.

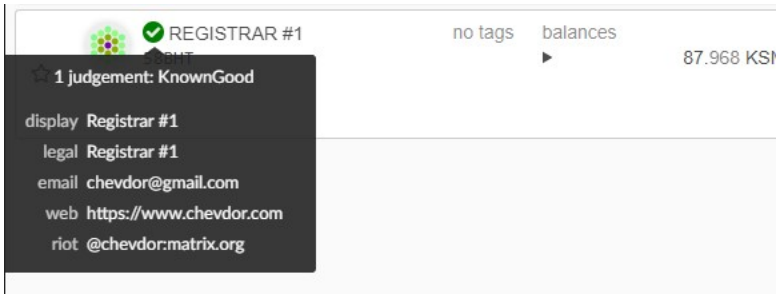
Kusama Registrars

There are multiple registrars on Kusama. Unless no additional information is available here, you must reach out to specific registrars individually if you want to be judged by those.

- Registrar 0:
  - URL: https://registrar.web3.foundation/
  - Account: H4XieK3r3dq3VEvRtqZR7wN7a1UEkXxf14orRsEfdFjmgkF,
  - Fee: 0.04 KSM
- Registrar 1:
  - URL: https://registrar.d11d.net/

    - Account: Fom9M5W6Kck1hNAiE2mDcZ67auUCiNTzLBUdQy4QnxHSxdn,
    - Fee: 0.65 KSM,
- Registrar 2:
    - Account: EK8veMNH6sVtvhSRo4q1ZRh6huCDm69gxK4eN5MFoZzo3G7,
    - Fee: 1 KSM,
- Registrar 3:
    - Account: GLiebiQp5f6G5vNcc7BgRE9T3hrZSYDwP6evERn3hEczdaM,
    - Fee: 1 KSM,
- Registrar 4:
    - Account: GhmpzxUyTVsFJhV7s2wNvD8v3Bgikb6WvYjj4QSuSScAUw6,
    - Fee: 0.04 KSM,

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

There are multiple registrars on Polkadot. Unless no additional information is available here, you must reach out to specific registrars individually if you want to be judged by those.

- Registrar 0:
    - URL: https://registrar.web3.foundation/
    - Account: 12j3Cz8qskCGJxmSJpVL2z2t3Fpmw3KoBaBaRGPnuibFc7o8,
    - Fee: 0 DOT,
- Registrar 1:
    - URL: https://registrar.d11d.net/
    - Account: 1Reg2TYv9rGfrQKpPREmrHRxrNsUDBQKzkYwP1UstD97wpJ,
    - Fee: 10 DOT,
- Registrar 2:
    - URL: https://polkaregistry.org/
    - Account: 1EpXirnoTimS1SWq52BeYx7sitsusXNGzMyGx8WPujPd1HB,
    - Fee: 0 DOT,

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 20.053 is required for every sub-account.

You can use polkadot.js/apps again to verify this amount by querying the `identity.subAccountDeposit` constant.

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.

Edit this page

*Last updated on **10/21/2021** by **Wei Tang***

General

About

FAQ

Contact

Build

Grants and Bounties

Careers

Technology

Technology

Roles

Telemetry

Substrate

Whitepaper

Lightpaper

Community

Community

Documentation

Brand Assets

Blog

Element Chat

Medium

Subscribe to the newsletter to hear about Polkadot updates and events.

Subscribe

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 are ready to be transferred.

Polkadot-JS Apps

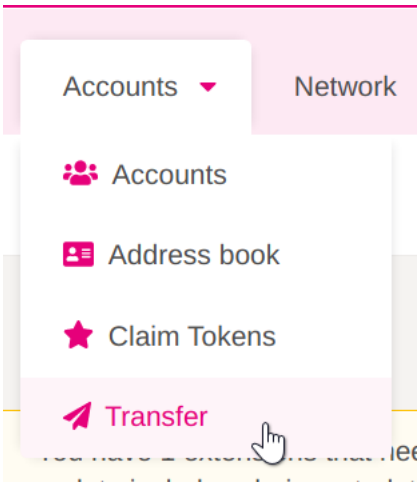
> NOTE: In this walkthrough we will be using the Polkadot network, but this is the same process for Kusama. If you would like to switch to a different network, you can change it by clicking the top left navigation dropdown and selecting a different network.

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

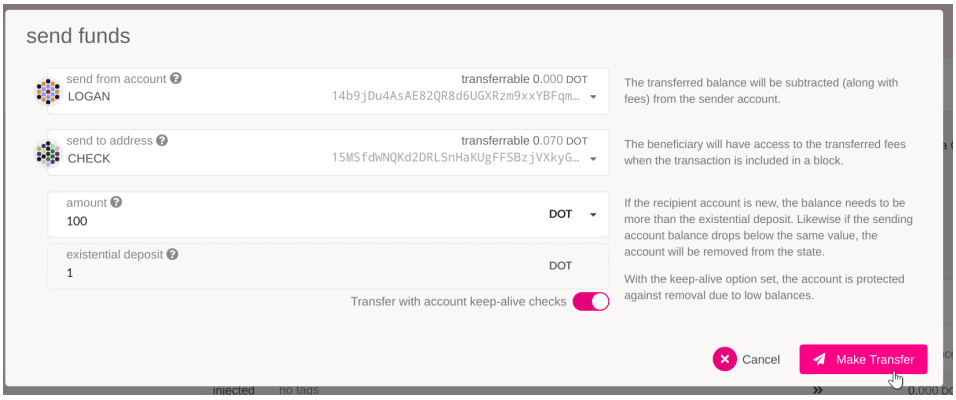
Click on the "Transfer" tab in the "Accounts" dropdown.



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.

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: 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 receive 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 previously 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

Currently, Polkadot does not use the Recovery Pallet, so this is probably not the reason for your tokens having existing references.

Existing Non-DOT Assets

Currently, Polkadot does not use the Assets Pallet, so this is probably not the reason for your tokens having 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.

Edit this page

*Last updated on **10/23/2021** by **Danny Salman***

Transaction Fees

Several resources in a blockchain network are limited, for example, storage and computation. Transaction fees prevent individual users from consuming too many resources. Polkadot uses a weight-based fee model as opposed to a gas-metering model. As such, fees are charged prior to transaction execution; once the fee is paid, nodes will execute the transaction.

Web3 Foundation Research designed the Polkadot fee system with the following objectives:

- Each Relay Chain block should be processed efficiently to avoid delays in block production.
- The growth rate of the Relay Chain should be bounded.
- Each block should have space for special, high-priority transactions like misconduct reports.
- The system should be able to handle spikes in demand.
- Fees should change slowly so that senders can accurately predict the fee for a given transaction.

Fee Calculation

Fees on the Polkadot Relay Chain are calculated based on three parameters:

- A per-byte fee (also known as the "length fee").
- A weight fee.
- A tip (optional).

The length fee is the product of a constant per-byte fee and the size of the transaction in bytes.

Weights are a fixed number designed to manage the time it takes to validate a block. Each transaction has a base weight that accounts for the overhead of inclusion (e.g. signature verification) as well as a dispatch weight that accounts for the time to execute the transaction. The total weight is multiplied by a per-weight fee to calculate the transaction's weight fee.

Tips are an optional transaction fee that users can add to give a transaction higher priority.

Together, these three fees constitute the inclusion fee. This fee is deducted from the sender's account prior to transaction execution. A portion of the fee will go to the block producer and the remainder will go to the Treasury. At Polkadot's genesis, this is set to 20% and 80%, respectively.

Block Limits and Transaction Priority

Blocks in Polkadot have both a maximum length (in bytes) and a maximum weight. Block producers will fill blocks with transactions up to these limits. A portion of each block - currently 25% - is reserved for critical transactions that are related to the chain's operation. Block producers will only fill up to 75% of a block with normal transactions. Some examples of operational transactions:

- Misbehavior reports
- Council operations
- Member operations in an election (e.g. renouncing candidacy)

Block producers prioritize transactions based on each transaction's total fee. Since a portion of the fee will go to the block producer, producers will include the transactions with the highest fees to maximize their reward.

Fee Adjustment

Transaction volume on blockchains is highly irregular, and therefore transaction fees need a mechanism to adjust. However, users should be able to predict transaction fees.

Polkadot uses a slow-adjusting fee mechanism with tips to balance these two considerations. In addition to block *limits*, Polkadot also has a block fullness *target.* Fees increase or decrease for the next block based on the fullness of the current block relative to the target. The per-weight fee can change up to 30% in a 24 hour period. This rate captures long-term trends in demand, but not short-term spikes. To consider short-term spikes, Polkadot uses tips on top of the length and weight fees. Users can optionally add a tip to the fee to give the transaction a higher priority.

Shard Transactions

The transactions that take place within Polkadot's shards - parachains and parathreads - do not incur Relay Chain transaction fees. Users of shard applications do not even need to hold DOT tokens, as each shard has its own economic model and may or may not have a token. There are, however, situations where shards themselves make transactions on the Relay Chain.

Parachains have a dedicated slot on the Relay Chain for execution, so their collators do not need to own DOT in order to include blocks. The parachain will make some transactions itself, for example, opening or closing an XCM channel, participating in an auction to renew its slot, or upgrading its runtime. Parachains have their own accounts on the Relay Chain and will need to use those funds to issue transactions on the parachain's behalf.

Parathreads will also make all the same transactions that a parachain might. In addition, the collators need to participate in an auction every block to progress their chain. The collators will need to have DOT to participate in these auctions.

Other Resource Limitation Strategies

Transaction weight must be computable prior to execution, and therefore can only represent fixed logic. Some transactions warrant limiting resources with other strategies. For example:

- Bonds: Some transactions, like voting, may require a bond that will be returned or slashed after an on-chain event. In the voting example, returned at the end of the election or slashed if the voter tried anything malicious.
- Deposits: Some transactions, like setting an identity or claiming an index, use storage space indefinitely. These require a deposit that will be returned if the user decides to free storage (e.g. clear their IDE).
- Burns: A transaction may burn funds internally based on its logic. For example, a

transaction may burn funds from the sender if it creates new storage entries, thus increasing the state size.
- Limits: Some limits are part of the protocol. For example, nominators can only nominate 16 validators. This limits the complexity of Phragmén.

Advanced

This page only covered transactions that come from normal users. If you look at blocks in a block explorer, though, you may see some "extrinsics" that look different from these transactions. In Polkadot (and any chain built on Substrate), an extrinsic is a piece of information that comes from outside the chain. Extrinsics fall into three categories:

- Signed transactions
- Unsigned transactions
- Inherents

This page only covered signed transactions, which is the way that most users will interact with Polkadot. Signed transactions come from an account that has funds, and therefore Polkadot can charge a transaction fee as a way to prevent spam.

Unsigned transactions are for special cases where a user needs to submit an extrinsic from a key pair that does not control funds. For example, when users claim their DOT tokens after genesis, their DOT address doesn't have any funds yet, so that uses an unsigned transaction. Validators also submit unsigned transactions in the form of "heartbeat" messages to indicate that they are online. These heartbeats must be signed by one of the validator's session keys. Session keys never control funds. Unsigned transactions are only used in special cases because, since Polkadot cannot charge a fee for them, each one needs its own, custom validation logic.

Finally, inherents are pieces of information that are not signed or included in the transaction queue. As such, only the block author can add inherents to a block. Inherents are assumed to be "true" simply because a sufficiently large number of validators have agreed on them being reasonable. For example, Polkadot blocks include a timestamp inherent. There is no way to prove that a timestamp is true the way one proves the desire to send funds with a signature. Rather, validators accept or reject the block based on how reasonable they find the timestamp. In Polkadot, it must be within some acceptable range of their own system clocks.

Learn More

- Web3 Foundation Research
- Substrate Weights
- Substrate Fees
- Extrinsics

Edit this page

*Last updated on **10/23/2021** by **Danny Salman***

Polkadot Host (PH)

The architecture of Polkadot can be divided into two different parts, the Polkadot *runtime* and the Polkadot *host*. The Polkadot runtime is the core state transition logic of the chain and can be upgraded over the course of time and without the need for a hard fork. In comparison, the Polkadot host is the environment in which the runtime executes and is expected to remain stable and mostly static over the lifetime of Polkadot.

The Polkadot host interacts with the Polkadot runtime in limited, and well-specified ways. For this reason, implementation teams can build an alternative implementation of the Polkadot host while treating the Polkadot runtime as a black box. For more details of the interactions between the host and the runtime, please see the specification.

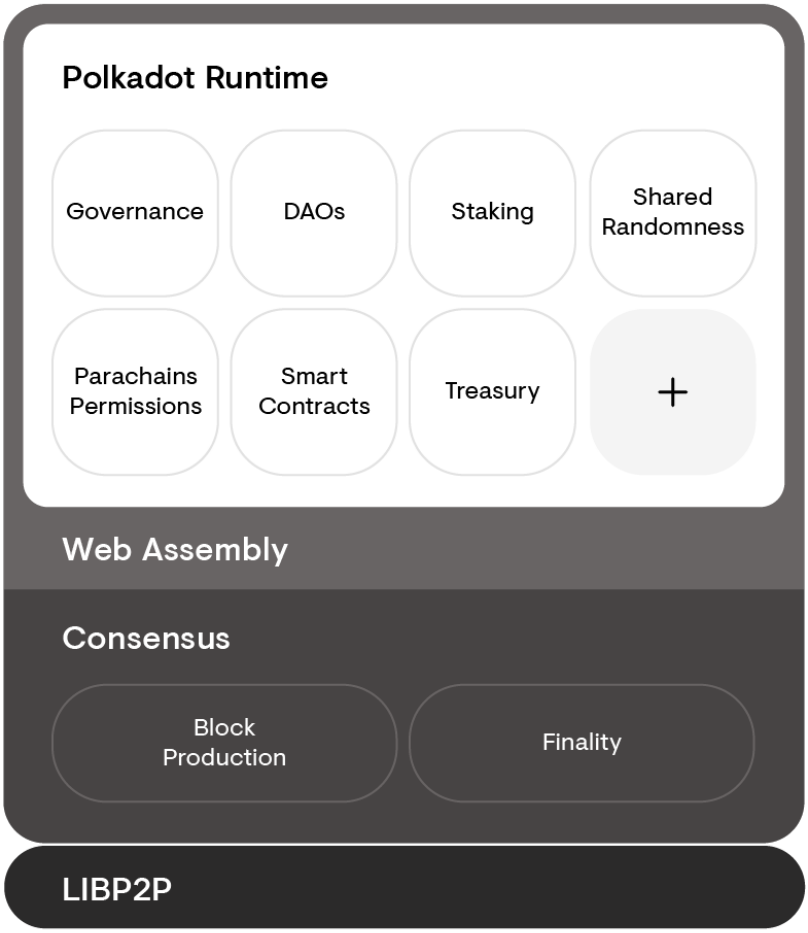
Components of the Polkadot host

- Networking components such as Libp2p that facilitates network interactions.
- State storage and the storage trie along with the database layer.
- Consensus engine for GRANDPA and BABE.
- Wasm interpreter and virtual machine.
- Low level primitives for a blockchain, such as cryptographic primitives like hash functions.

A compiled Polkadot runtime, a blob of Wasm code, can be uploaded into the Polkadot host and used as the logic for the execution of state transitions. Without a runtime, the Polkadot host is unable to make state transitions or produce any blocks.

Diagram

Below is a diagram that displays the Polkadot host surrounding the Polkadot runtime. Think of the runtime (in white) as a component that can be inserted, swapped out, or removed entirely. While the parts in grey are stable and can not change without an explicit hard fork.



Resources

- Polkadot Host Protocol Specification - Incubator for the Polkadot Host spec, including tests.
- ChainSafe's Go PH is a 25-person development team based in Toronto, Canada. ChainSafe is building an implementation of the beacon chain for Ethereum 2.0 client in TypeScript and this Go implementation of Polkadot.

Edit this page

*Last updated on **9/17/2021** by **Danny Salman***

FAQ
Contact
Build
Grants and Bounties
Careers

Tokens
Telemetry
Polkatools
Whitepaper
Lightpaper

Documentation
Brand Assets
Blog
Element Chat
Medium

about Polkadot updates and events.

Subscribe

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 24 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. This percentage is currently at 1% on Polkadot.

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 100.000 DOT (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

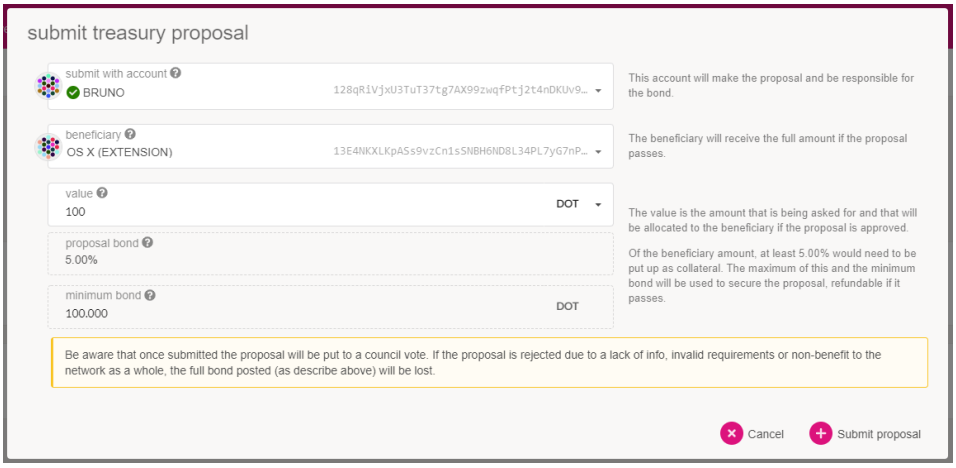
To minimize storage on chain, proposals don't contain contextual information. When a user submits a proposal, they will probably need to find an off-chain way to explain the proposal. Most discussion takes place on the following platforms:

- 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 the Polkadot Watercooler and Polkadot Direction room .

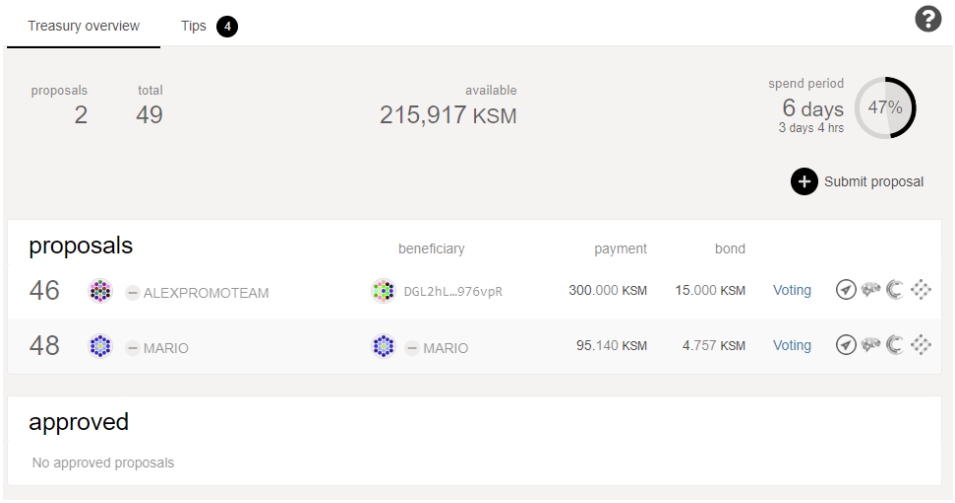
Creating the Proposal

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.

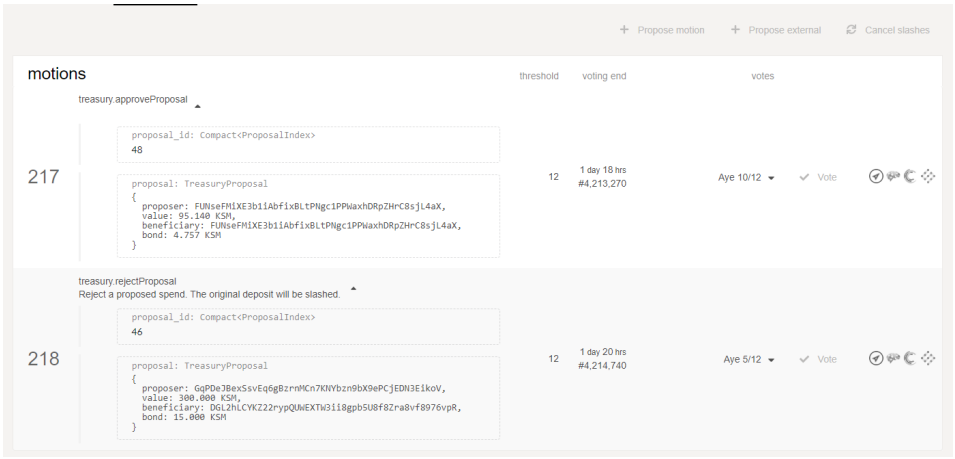
Once created, your proposal will become visible in the Treasury screen and the Council can start voting on it.



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 time frame, 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 1. Public tips carry a finder's fee of 20% 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. 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 Polkassembly 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.

For example:

Charlie tips 10 DOT. Dave tips 30 DOT. Eve tips 100 DOT.

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 30 DOT 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 Direction channel in Element or joining our Polkadot Discord server 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)

Edit this page

*Last updated on **11/13/2021** by **alex***

Using W3F Registrar

> **NOTE**: The beta version of the new registrar service is available at https://registrar.web3.foundation/. The previous service has been deprecated.

Edit this page

*Last updated on **9/17/2021** by **Danny Salman***

Runtime Upgrades

Runtime upgrades allow Polkadot to change the logic of the chain, without the need for a hard fork.

Forkless Upgrades

You may have come across the term "hard fork" before in the blockchain space. A **hard fork** occurs when a blockchain's logic changes such that nodes that do not include the new changes will not be able to remain in consensus with nodes that do. Such changes are backward incompatible. Hard forks can be political due to the nature of the upgrades, as well as logistically onerous due to the number (potentially thousands) of nodes in the network that need to upgrade their software.

Rather than encode the runtime (a chain's "business logic") in the nodes, Polkadot nodes contain a WebAssembly execution host. They maintain consensus on a very low level and well-established instruction set. The Polkadot runtime is stored on the Polkadot blockchain itself.

As such, Polkadot can upgrade its runtime by upgrading the logic stored on-chain, and removes the coordination challenge of requiring thousands of node operators to upgrade in advance of a given block number. Polkadot stakeholders propose and approve upgrades through the on-chain governance system, which also enacts them autonomously.

New Client Releases

The existing runtime logic is followed to update the Wasm runtime stored on the blockchain to a new version. The upgrade is then included in the blockchain itself, meaning that all the nodes on the network execute it. Generally, there is no need to upgrade your nodes manually before the runtime upgrade as they will automatically start to follow the new logic of the chain. Nodes only need to be updated when the runtime requires new host functions or there is a change in networking or consensus.

Transactions constructed for a given runtime version will not work on later versions. Therefore, a transaction constructed based on a runtime version will not be valid in later runtime versions. If you don't think you can submit a transaction before the upgrade, it is better to wait and construct it after the upgrade takes place.

Although upgrading your nodes is generally not necessary to follow an upgrade, we recommend following the Polkadot releases and upgrading promptly, especially for high priority or critical releases.

Runtime Upgrades for Various Users

For Infrastructure Providers

Infrastructure services include but are not limited to the following:

- Validators
- API services
- Node-as-a-Service (NaaS)

- General infrastructure management (e.g. block explorers, custodians)
- Wallets

For validators, keeping in sync with the network is key. At times, upgrades will require validators to upgrade their clients within a specific time frame, for example if a release includes breaking changes to networking. It is essential to check the release notes, starting with the upgrade priority and acting accordingly.

General infrastructure providers, aside from following the Polkadot releases and upgrading in a timely manner, should monitor changes to runtime events and auxiliary tooling, such as the Substrate API Sidecar.

Transactions constructed for runtime `n` will not work for runtimes `>n`. If a runtime upgrade occurs before broadcasting a previously constructed transaction, you will need to reconstruct it with the appropriate runtime version and corresponding metadata.

For Nominators

Runtime upgrades don't require any actions by a nominator, though it is always encouraged to keep up-to-date and participate with the latest runtime upgrade motions and releases, while keeping an eye on how the nodes on the network are reacting to a new upgrade.

Monitoring Changes

You can monitor the chain for upcoming upgrades. The client release notes include the hashes of any proposals related to any on-chain upgrades for easy matching. Monitor the chain for:

1. `democracy(Started)` events and log `index` and `blockNumber`. This event indicates that a referendum has started (although does not mean that it is a runtime upgrade). Get the referendum info*; it should have a status of `Ongoing`. Find the ending block number (`end`) and the enactment `delay` (delay). If the referendum passes, it will execute on block number `end + delay`.
2. `democracy(Passed)`, `democracy(NotPassed)`, or, `democracy(Cancelled)` events citing the index. If `Passed`, you need to look at the `scheduler(Scheduled)` event in the same block for the enactment block.
3. `democracy(PreimageNoted)` events with the same hash as the `ReferendumInfoOf(index)` item. This may be up to the last block before execution, but it will not work if this is missing.
4. `democracy(Executed)` events for actual execution. In the case of a runtime upgrade, there will also be a `system(CodeUpdated)` event.

You can also monitor Polkassembly for discussions on on-chain proposals and referenda.

* E.g. via `pallets/democracy/storage/ReferendumInfoOf?key1=index&at=blockNumber` on Sidecar.

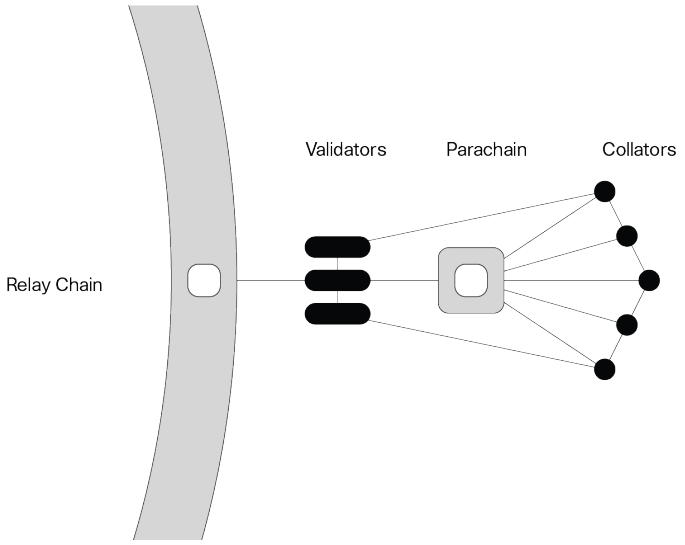
Edit this page

*Last updated on **10/23/2021** by **Danny Salman***

Parachains

> For information on how to participate in the crowdloan and parachain auction testing on Rococo, please see the Rococo Contenton the parachain development guide.

A parachain is an application-specific data structure that is globally coherent and validatable by the validators of the Relay Chain. They take their name from the concept of parallelized chains that run parallel to 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.



Due to their parallel nature, they are able to parallelize transaction processing and achieve scalability of the Polkadot system. They share in the security of the entire network and can communicate with other parachains through the XCM format.

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 the native token unless stipulated by the parachain implementation.

The Polkadot Host (PH) requires that the state transitions performed on parachains 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 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 to handle validation and finalization; parachains will not be required to do either of those things. However, since Polkadot 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 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.

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 two 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 many messages are in queue to be processed or if no nodes are running both of the parachain networks that can quickly gossip the message across the networks.

Due to the necessary latency involved in sending crosschain messages, some parachains plan 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 synergistically composed with others to create new applications. One example of this includes 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 composability property weakens 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 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 governance 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 and help remove transactions from the Relay Chain, allowing for more efficient parachain processing.

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

Parathreads 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 bids get reserved until the lease's end. 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 governance system. Generally, a common good parachain's lease would not expire; it would only be removed via governance.

See the Polkadot blog article and the common good parachains page for more information.

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. Polkadot guarantees valid state transitions. Executing a block finality outside the context of the relay chain is outside the scope of trust that Polkadot provides.

How about parachains that are not Substrate-based?

Substrate provides FRAME Pallets as part of its framework to seamlessly build a rustic-based blockchain. Part of FRAME are pallets that can be used for consensus. Polkadot being a Substrate-based chain relies on BABE as the block production scheme and GRANDPA as the finality gadget as part of its consensus mechanism. Collectively, this is a Hybrid Consensus Model, where block production and block finality are separate. Parachains only need to produce blocks as they can rely on the relay chain to validate the state transitions. Thus, parachains can have their own block production where the collators act as the block producers, even if the parachain is not Substrate-based.

How will parachain slots be distributed?

Parachain slots will be acquirable through auction, please see the parachain slots article. Additionally, some parachain slots will be set aside to run parathreads — 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 are too small to validate a block will skip those blocks. Their block production speed will slow down to an increment of six 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 recognize 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 information, see the PDK content.

Please see the Parachain Development page for more information.

Resources

- Polkadot: The Parachain - Blog post by Polkadot co-founder Rob Habermeier who 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.

Edit this page

*Last updated on **10/23/2021** by **Danny Salman***

Common Good Parachains

Overview

"Common Good" parachains are parachain slots reserved for functionality that benefits 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 governance system. Generally, a common good parachain's lease would not expire; it would only be removed via governance.

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. Moving system level logic to a parachain 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.

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.

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 triggering an upgrade.

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.

Common Good Chains in Development

Statemint

Statemint (and its cousin *Statemine* on Kusama) will likely be one of the first common good parachains.

> Statemine is the first common good parachain.

Statemint is a public utility chain in that it adds functionality not available in the Relay Chain, namely the creation and management of assets. Statemint will support both fungible and non-fungible assets. The chain offers an interface similar to ERC-20 for fungible assets and ERC-721 for non-fungible tokens. These interfaces are in the logic of the chain itself; by encoding this logic directly into the Statemint runtime, token storage, and actions do not need to be metered and can happen faster and cheaper.

Like most common good chains, Statemint will use the DOT token as its native token, i.e. represented in its instance of the Balances pallet. Statemint trusts messages about balances from the Relay Chain, and vice versa, so users can transfer DOT from the Relay Chain to their address on Statemint and back.

Because of the efficiency of executing logic in a parachain, the transaction fees and deposits (including the existential deposit) are about 1/10th of their value on the Relay Chain. These low fee levels mean that Statemint is well suited to handling DOT balances and transfers as well as managing on-chain assets.

As a common good parachain, Statemint must stay fully aligned with the Relay Chain. Upgrades to Statemint will require the Relay Chain's "root origin", i.e. a referendum. Some of the other logic (like privileged asset functionality) will defer to the Relay Chain's Council.

Bridges

See the Bridges page for information on the latest bridge projects.

Edit this page

*Last updated on **10/12/2021** by **Danny Salman***

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 (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 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.

Randomness in Action

The following example will showcase the randomness mechanics of the candle auction for the ninth auction on Kusama. Keep in mind that the candle phase has a uniform termination profile and has an equal probability of ending at any given block, and the termination block cannot be predicted before or during the auction.

- Auction 9 starts at `block 9362014`.
  > The auction has a full duration equal to `block 9362014` + `72000`. Here, `block 72000` is the "ending period", which is divided into **3600 samples of 20 blocks**. Figuratively, the candle is lit, and the candle phase lasts for 72,000 blocks.
- The winning sample during the ending period turned out to have the `index 1078`.
  > Sample 1078 refers to the winner as of `block 9362014 + 21560`, which equals `block 9383574`.
- The parent block was a new BABE session in the 'Logs', which updated the randomness that was used to select that sample index.
  > You'd be able to inspect the state at the end of `block 9434277` and see the sample indices with an archive node. The digest in the 'Logs' of `9434277` is decodable and contains the random value as well as the BABE authorities.
- As a result, the winner of this auction did not turn out to be the highest bid during the full duration.

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

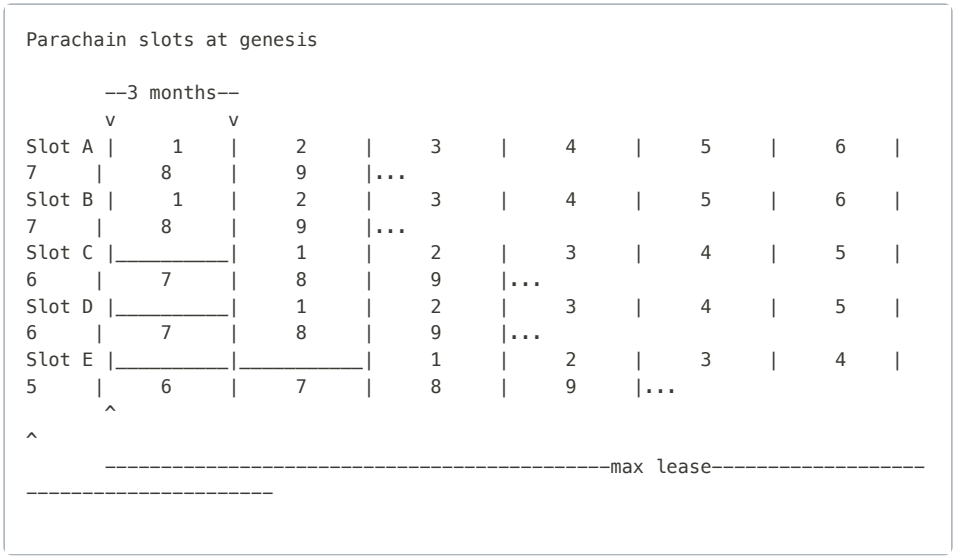
Polkadot will use a *random beacon* based on the VRF that's used also in other places of the protocol. The VRF will provide the base of the randomness, which will retroactively determine the end-time of the auction.

The slot durations are capped to 2 years and divided into 3-month periods ; 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.



Each period of the range 1 - 4 represents a 3-month duration for a total of 2 years

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 within 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 equals 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 three-month duration but the overall slot a 2-year duration, 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 just the period 9 (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 or common-good parachains may be granted slots by the governing bodies of the Relay Chain. System parachains can be recognized by a parachain ID lower than 1_000, and common-good parachains by a parachain ID between 1_000 and 1_999. Other parachains will have IDs 2_000 or higher. 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
- Research Update: The Case for Candle Auctions - W3F breakdown and research update about candle auctions

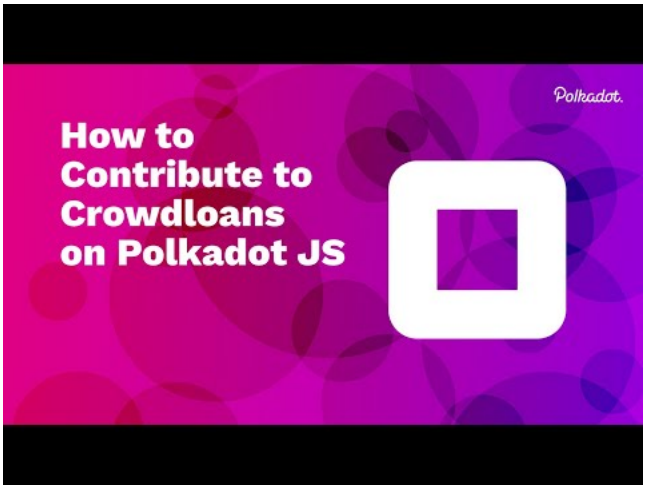
- Front-Running, Smart Contracts, and Candle Auctions W3F Research team discusses how to remedy current blockchain auction setbacks with candle auctions

Edit this page

*Last updated on **10/20/2021** by **Danny Salman***

Parachain Crowdloans

Polkadot allows parachains to source tokens for their parachain bids in a decentralised crowdloan. If you are here for guidance on how to contribute for a crowdloan, watch the video below or read this support article on crowdloans.



> For information on how to participate in the crowdloan and parachain auction testing on Rococo, please see the Rococo content.

Starting a Crowdloan Campaign

Anyone who has registered a parachain can create a new crowdloan campaign for a slot by depositing a specified number of tokens. A campaign is configured as a range of slots (i.e. the duration of the parachain 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.

When setting the parameters of a crowdloan campaign, consider the following:

> *Important*: a crowdloan campaign can start well before the auction slot is opened.
>
> - The campaign creation form requires setting a crowdloan cap — the maximum amount a campaign can collect. A team can still win an auction if the cap is not reached.
> - Set the desired end of the crowdloan in the "Ending block" field. This helps to ensure that the crowdloan is live during the entire duration of the auction. For example, if an auction starts in three days and will last for five days, you may want to set your crowdloan to end in 10 days, or a similar timescale.

- One way of calculating the ending block number is adding: `(10 * 60 * 24 * 7) * (x * 6) + y`
  - `x` is the number of auction periods you want the crowdloan to continue for
  - `y` is the current block number
  - `(Blocks/Min * Min/Hour * Hour/Day * Day/Week) * (x[Period] * Week/Period) + y[BlockNumber]`

- "First period" field refers to the first period you want to bid for. If the current auction encompasses periods `(3, 4, 5, 6)`, your first period can be at least `3`. The last slot must also be within that range.

- You can only cancel an ongoing crowdloan if no contributions have been made. Your deposit will be returned to you.

Prior to the start of the crowdloan campaign, the owner will upload the parachain data. Once the crowdloan is live, **the parachain configuration will be locked** and will be 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).

Supporting a Crowdloan Campaign

> **Important**: The minimum balance for contributions for a crowdloan campaign is currently set to 5 DOTs. This is in an attempt to make crowdloans as accessible as possible while maintaining a balance to justify the use of the network's resources..

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 an 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.

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.

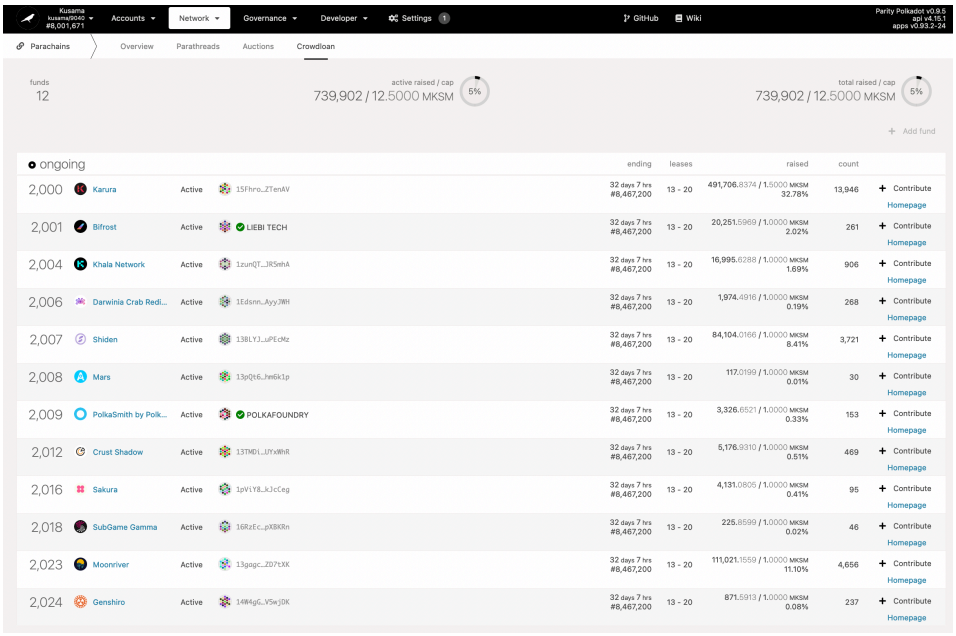
Participants will be able to reclaim their tokens in one of two ways:

- 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 unsuccessful, 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 entirely deleted from the system.

Many projects will have dashboards available that allow users to participate in their crowdloans. PolkadotJS apps also offers a breakdown of ongoing crowdloans on the Apps page.

Here is an example of the crowdloans in play during the very first Kusama auction.



Furthermore, check out this video on How to Participate in Crowdloans for steps on how to access available crowdloans on PolkadotJS apps.

Edit this page

*Last updated on **11/5/2021** by **Radha***

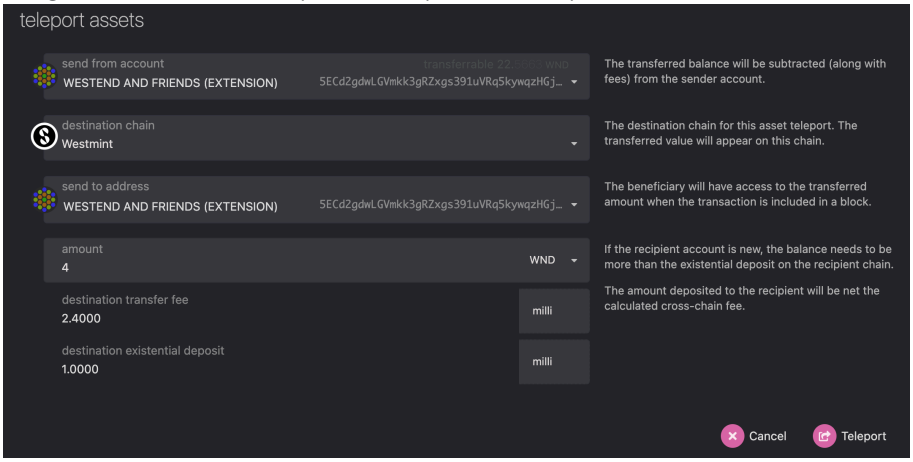
Teleporting Assets To Other Parachains

One of the main properties that Polkadot and Kusama bring to the ecosystems is decentralized blockchain interoperability. This interoperability allows for asset teleportation: the process of moving assets, such as coins, tokens, or NFTs, between chains (parachains) to use them as you would any other asset native to that chain. Interoperability is possible through XCM and SPREE modules, which together ensure that assets are not lost or duplicated across multiple chain.

How to Teleport

Teleportation can be done through the PolkadotJS Apps interface or through the `xcmPallet.teleportAssets()` extrinsic. In the following example, we will be using the PolkadotJS interface.

1. Navigate to PolkadotJS Apps and connect to the chain with the tokens you want to teleport.
2. Navigate to "Accounts > Teleport". This opens the 'teleport assets' interface:



3. Fill out the transaction:
   i. "send from account" - Select the account with the source tokens.
   ii. "destination chain" - Select the parachain you want to send the assets to.
   iii. "send to address" - Select the account you want to be in control of the coins on the destination chain.
   iv. "amount" - Insert the number of tokens you want to teleport
4. After reviewing the transaction information and fees, click the "Teleport" button.
5. Click "Sign and Submit".
6. Enter your password, and click "Sign the transaction".

The transaction will be signed and broadcasted, and the tokens will appear on the destination chain shortly.

Troubleshooting

If you do not see "Accounts > Teleport" in PolkadotJS Apps, the source chain that you have selected does not support teleportation yet. As of June 2021, unsupported chains include Polkadot mainnet, Rococo testnet, and their respective parachains.

Edit this page

*Last updated on **10/23/2021** by **Danny Salman***

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 several 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.

How do Parathreads Operate?

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.

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-objects 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. 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.

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 native currency 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

Edit this page

*Last updated on **10/20/2021** by **Danny Salman***

Bridges

A cornerstone technology of blockchain interoperability is the blockchain bridge. Blockchain bridges are ways for two economically sovereign and technologically diverse chains to communicate with each other. Bridge designs come in a variety of flavors ranging from centralised and trusted to more decentralised and trustless. Polkadot favors the latter bridge designs for its ecosystem. However, there is nothing that blocks a development team from building and deploying the former.

While bridge designs are now getting to a place where they are sufficiently planned out, there have not been too many used heavily in production. For this reason, you can consider this page a work in progress. It will be updated as more information is determined and available.

> Bridges are specifically for making the Polkadot ecosystem compatible with external blockchains such as Bitcoin, Ethereum, or Tezos (among others). For information on XCM, the native interoperability technology that allows parachains to communicate trustlessly, please see the dedicated cross consensus page on the Wiki.

Bridging Methods

Building a bridge that is as decentralised and trustless as possible can be done through any of the following methods (ordered by suggested methodology):

- *Bridge pallets* - For Substrate-native chains, use a bridge pallet (e.g. Kusama `\<\>` Polkadot bridge, since both networks' parachains use Substrate).
- *Smart contracts* - If the chain is not on Substrate, you should have smart contracts on the non-Substrate chain to bridge (e.g. Ethereum mainnet will have a bridge smart contract that initiates Eth transactions based on incoming XCMP messages).
- *Higher-order protocols* - If your chain does not support smart contracts (e.g. Bitcoin), you should use XClaim or similar protocols to bridge.

via Bridge Pallets

Receiving messages on Polkadot from an external, non-parachain blockchain can be possible through a Substrate pallet. The Substrate instance can then be deployed to Polkadot either as a system-level parachain (native extension to the core Polkadot software) or as a community-operated parachain.

An example of a bridge that would strictly use bridge pallets would be a Kusama `\<\>` Polkadot bridge, since both use parachains based on Substrate.

For the standalone chains that will not have a parachain bridging module on Polkadot (non-Substrate), it will be necessary to deploy bridge contracts (see below).

via Smart Contracts

Given the generality of blockchain platforms with Turing-complete smart contract languages, it is possible to bridge Polkadot and any other smart contract capable blockchain.

Those who are already familiar with Ethereum may know of the now archived Parity Bridge and the efforts being made to connect PoA sidechains to the Ethereum mainnet. The Parity bridge is a combination of two smart contracts, one deployed on each chain, that allow for cross-chain transfers of value. As an example of usage, the initial Parity Bridge proof of concept connects two Ethereum chains, `main` and `side`. Ether deposited into the contract on `main` generates a balance denominated in ERC-20 tokens on `side`. Conversely, ERC-20 tokens deposited back into the contract on `side` can free up Ether on `main`.

> To learn more on how Bitcoin and Ethereum can Cooperate and Collaborate Through Polkadot, check out this explainer video here

via Higher-Order Protocols

Higher-order protocols (like XCLAIM) can be used to bridge but should only be used when other options are not available. XCLAIM, in particular, requires any swappable asset to be backed by a collateral of higher value than the swappable assets, which adds additional overhead.

An example of a network that would be well-suited for higher-order protocols would be Bitcoin, since it does not support smart-contracts and it's not based on Substrate.

Examples

Ethereum Bridge (Smart Contracts \<> Polkadot)

As explained by Dr. Gavin Wood in a blog post from late 2019, there are three ways that the Polkadot and Substrate ecosystem can be bridged to the Ethereum ecosystem.

1. Polkadot <-> Ethereum Public Bridge.
2. Substrate <-> Parity Ethereum (Openethereum) Bridge.
3. The Substrate EVM module.

Please read the blog article for fuller descriptions of each one of these options.

Bitcoin Bridge (XCLAIM \<> Substrate \<> Polkadot)

The Interlay team has written a specification on a Bitcoin bridge that is based on the XCLAIM design paper. The protocol enables a two-way bridge between Polkadot and Bitcoin. It allows holders of BTC to "teleport" their assets to Polkadot as PolkaBTC, and holders of PolkaBTC to burn their assets for BTC on the Bitcoin chain.

The Bitcoin bridge, as documented in the specification, is composed of two logically different components:

- The XCLAIM component maintains all accounts that own PolkaBTC.
- The BTC-Relay is responsible for verifying the Bitcoin state when a new transaction is submitted.

For full details on how it works, please refer to the specification.

There is now a reference implementation and testnet available.

Additional Resources and Examples

For Bridge Builders

If your team is interested in building a bridge between an external chain and Polkadot, funding may be available from the W3F grants program. Please first check that the chain you intend to bridge between hasn't already been built or is in the process of being created by another team. More popular chains with clear use cases will be given priority, and novel bridge designs are welcome.

Resources and Examples

- Parity Bridges Common Resources
- Substrate/Ethereum Bridge - ChainSafe and Centrifuge were awarded a grant in W3F Grants Wave 5 to build a Substrate to Ethereum two-way bridge.
- PolkaBTC (Bitcoin \<> Polkadot Bridge)
- EOS Bridge - The Bifrost team was awarded a grant in W3F Grants Wave 5 to build a bridge to EOS.
- Tendermint Bridge - ChorusOne was awarded a grant in Wave 5 to build a GRANDPA light client in Tendermint.
- Interlay BTC Bridge - The Interlay team was awarded a grant in W3F grants Wave 5 to build a trust-minimized BTC bridge.
- ChainX BTC Bridge - ChainX have implemented a BTC to Substrate bridge for their parachain.
- POA Network
- Case study of POA Network's implementation of Parity's bridge chain solution.
- Edgeth Bridge - a bridge from Ethereum to Edgeware chain (a Substrate-based chain) - now defunct and not maintained, but a good example.
- XCLAIM - XCLAIM is a framework for achieving trustless and efficient cross-chain exchanges using cryptocurrency-backed assets.

Edit this page

*Last updated on **10/20/2021** by **Danny Salman***

Staking

Polkadot uses NPoS (Nominated Proof-of-Stake) as its consensus mechanism. The system encourages DOT holders to participate as nominators. Nominators may back up to 16 validators as trusted validator candidates. Both validators and nominators lock their tokens as collateral and receive staking rewards.

The staking system pays out rewards essentially equally to all validators regardless of stake. Having more stake on a validator does not influence the amount of block rewards it receives. However, there is a probabilistic component to reward calculation (discussed below), so rewards may not be exactly equal for all validators in a given era.

Distribution of the rewards are pro-rata to all stakers after the validator payment is deducted. In this way, the network creates incentives for the nomination of lower-staked validators to create an equally-staked validator set.

How does staking work in Polkadot?

1. Identifying which role you are

In staking, you can be either a nominator or a validator.

As a nominator, you can nominate validator candidates that you trust to help you earn rewards in the chain's native token. You can take a look at the nominator guide to understand your responsibilities as a nominator, and the validator docs to understand what you need to do as a validator.

If you are a beginner and would like to securely stake your tokens using Polkadot JS Apps, watch the video below

2. Nomination period

Any potential validators can indicate their intention to be a validator candidate. Their candidacies are made public to all nominators, and a nominator in turn submits a list of any number of candidates that it supports. In the next era, a certain number of validators having the most DOT backing get elected and become active.

There are no particular requirements to become a nominator, though we expect each nominator to carefully track the performance and reputation of the validators they back. Nominating is *not* a "set and forget" operation.

Once the nomination period ends, the NPoS election mechanism takes the nominators and their associated votes as input, and outputs a set of validators. This "election solution" has to meet certain requirements, such as maximizing the amount of stake to nominate validators and distributing the stake backing validators as evenly as possible. The objectives of this election mechanism are to maximize the security of the network, and achieve fair representation of the nominators. If you want to know more about how NPoS works (e.g. election, running time complexity, etc.), please read here.

3. Staking Rewards Distribution

To explain how rewards are paid to validators and nominators, we need to consider **validator pools**. A validator pool consists of the stake of an elected validator together with the nominators backing it.

If a nominator `n` with stake `s` backs several elected validators, say `k`, the NPoS election mechanism will split its stakes into pieces `s_1`, `s_2`, ..., `s_k`, so that it backs validator `i` with stake `s_i`. In that case, nominator `n` will be rewarded essentially the same as if there were `k` nominators in different pools, each backing a single validator `i` with stake `s_i`.

For each validator pool, we keep a list of nominators with the associated stakes.

The general rule for rewards across validator pools is that two validator pools get paid essentially the **same amount of tokens** for equal work, i.e. they are NOT paid proportional to the stakes in each pool. There is a probabilistic component to staking rewards in the form of era points and tips but these should average out over time.

Within a validator pool, a (configurable) percentage of the reward goes to pay the validator's commission fees and the remainder is paid **pro-rata** (i.e. proportional to stake) to the nominators and validator. Notice in particular that the validator is rewarded twice: once in commission fees for validating (if their commission rate is above 0%), and once for nominating itself with stake. If a validator's commission is set to 100%, no tokens will be paid out to any nominations in the validator pool.

To estimate the inflation rate and how many tokens you can get each month as a nominator or validator, you can use this tool as a reference and play around with it by changing some parameters (e.g. how many days you would like to stake with your DOT, provider fees, compound rewards, etc.) to have a better estimate. Even though it may not be entirely accurate since staking participation is changing dynamically, it works well as an indicator.

4. Rewards Mechanism

We highlight two features of this payment scheme. The first is that since validator pools are paid the same regardless of stake level, pools with less stake will generally pay more to nominators per-token than pools with more stake.

We thus give nominators an economic incentive to gradually shift their preferences to lower staked validators that gain a sufficient amount of reputation. The reason for this is that we want the stake across validator pools to be as evenly distributed as possible, to avoid a concentration of power among a few validators.

In the long term, we expect all validator pools to have similar levels of stake, with the stake being higher for higher reputation validators (meaning that a nominator that is willing to risk more by backing a validator with a low reputation will get paid more).

The following example should clarify the above. For simplicity, we have the following assumptions:

- These validators do not have a stake of their own.
- They each receive the same number of era points.
- There are no tips for any transactions processed.
- They do NOT charge any commission fees.
- Total reward amount is 100 DOT tokens.
- The current minimum amount of DOT to be a validator is 350 (note that this is *not* the actual value, which fluctuates, but merely an assumption for purposes of this example; to understand how the actual minimal stake is calculated, see here).

	A - Validator Pool		
Nominator (4)	Stake (600)	Fraction of the Total Stake	Rewards
Jin	100	0.167	16.7
Sam	50	0.083	8.3
Anson	250	0.417	41.7
Bobby	200	0.333	33.3

	B - Validator Pool		
Nominator (4)	Stake (400)	Fraction of the Total Stake	Rewards
Alice	100	0.25	25
Peter	100	0.25	25
John	150	0.375	37.5
Kitty	50	0.125	12.5

Both validator pools A & B have 4 nominators with the total stake 600 and 400 respectively.

Based on the above rewards distribution, nominators in validator pool B get more rewards per DOT than those in pool A because pool A has more overall stake. Sam has staked 50 DOT in pool A, but he only gets 8.3 in return, whereas Kitty gets 12.5 with the same amount of stake.

There is an additional factor to consider in terms of rewards. While there is no limit to the number of nominators a validator may have, a validator does have a limit to how many nominators to which it can pay rewards.

In Polkadot and Kusama, this limit is currently {{ polkadot_max_nominators }}, although this can be modified via runtime upgrade. A validator with more than {{ polkadot_max_nominators }} nominators is *oversubscribed*. When payouts occur, only the

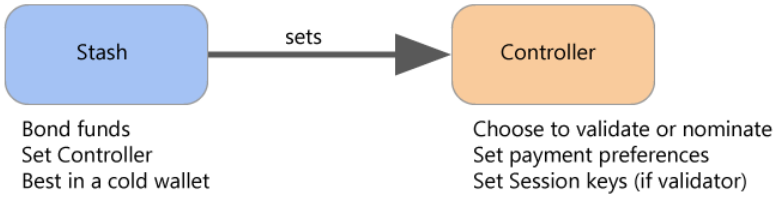
top {{ polkadot_max_nominators }} nominators as measured by amount of stake allocated to that validator will receive rewards. All other nominators are essentially "wasting" their stake - they used their nomination to elect that validator to the active stake, but receive no rewards in exchange for doing so.

We also remark that when the network slashes a validator slot for a misbehavior (e.g. validator offline, equivocation, etc.) the slashed amount is a fixed percentage (and NOT a fixed amount), which means that validator pools with more stake get slashed more DOT. Again, this is done to provide nominators with an economic incentive to shift their preferences and back less popular validators whom they consider to be trustworthy.

The second point to note is that each validator candidate is free to name their desired commission fee (as a percentage of rewards) to cover operational costs. Since validator pools are paid the same, pools with lower commission fees pay more to nominators than pools with higher fees. Thus, each validator can choose between increasing their fees to earn more, or decreasing their fees to attract more nominators and increase their chances of being elected. In the long term, we expect that all validators will need to be cost efficient to remain competitive, and that validators with higher reputation will be able to charge slightly higher commission fees (which is fair).

Accounts

There are two different accounts for managing your funds: `Stash` and `Controller`.



- **Stash:** This account holds funds bonded for staking, but delegates some functions to a Controller. As a result, you may actively participate with a Stash key kept in a cold wallet, meaning it stays offline all the time. You can also designate a Proxy account to vote in governance proposals.
- **Controller** This account acts on behalf of the Stash account, signalling decisions about nominating and validating. It sets preferences like payout account and commission. If you are a validator, it also sets your session keys. It only needs enough funds to pay transaction fees.

We designed this hierarchy of separate key types so that validator operators and nominators can protect themselves much better than in systems with only one key. As a rule, you lose security anytime you use one key for multiple roles, or even if you use keys related by derivation. You should never use any account key for a "hot" session key in particular.

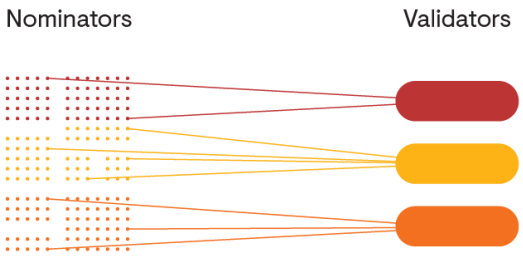
Controller and Stash account keys can be either sr25519 or ed25519. For more on how keys are used in Polkadot and the cryptography behind it see here.

Validators and nominators

Since validator slots are limited, most of those who wish to stake their DOT and contribute economic security to the network will be nominators.

Validators do most of the heavy lifting: they produce new block candidates in BABE, vote and come to consensus in GRANDPA, validate the state transition function of parachains, and possibly some other responsibilities regarding data availability and XCM.

Nominators, on the other hand, have far fewer responsibilities. Those include monitoring their validators' performance (uptime), keeping an eye on changing commission rates (a validator can change commission at any time), and general health monitoring of their and their validators' account. Thus, while not set-it-and-forget-it, a nominator's experience is relatively hands-off compared to a validators.



Want to stake DOT?

- Nominator Guide - Become a nominator on the Polkadot network.
- Validator Guide - Become a validator on the Polkadot network.

Slashing

Slashing will happen if a validator misbehaves (e.g. goes offline, attacks the network, or runs modified software) in the network. They and their nominators will get slashed by losing a percentage of their bonded/staked DOT.

Any slashed DOT will be added to the Treasury. The rationale for this (rather than burning or distributing them as rewards) is that slashes may then be reverted by the Council by simply paying out from the Treasury. This would be useful in situations such as a faulty runtime causing slashing or forcing validators offline through no fault of their own. In the case of legitimate slashing, it moves tokens away from malicious validators to those building the ecosystem through the normal Treasury process.

Validator pools with larger total stake backing them will get slashed more harshly than less popular ones, so we encourage nominators to shift their nominations to less popular validators to reduce their possible losses.

It is important to realize that slashing only occurs for active validations for a given nominator, and slashes are not mitigated by having other inactive or waiting nominations. They are also not mitigated by the validator operator running separate validators; each validator is considered its own entity for purposes of slashing, just as they are for staking rewards.

As an example, assume BIG_COMPANY has 50 validators that all go offline at the same time, thus causing a 1% unresponsiveness slash to their nominators. In this example, the nominator has nominated five validators, two of which are with BIG_COMPANY (BC_1 and BC_2) and three are with other validators that do not belong to BIG_COMPANY (OV_1, OV_2, and OV_3). In this era, BC_1 is the active validator for this nominator, BC_2 and OV_1 are inactive, and OV_2 and OV_3 are waiting. The nominator will be slashed 1% of bonded stake, since BC_1 is the active validator. The inactive and waiting validators (BC_2 and OV_1 through 3) don't have any effect on this, since they are not actively validating. Any nominator actively nominating BC_2 also receives a 1% slash, but any nominator actively nominating OV_1 is unaffected.

In rare instances, a nominator may be actively nominating several validators in a single era. In this case, the slash is proportionate to the amount staked to that specific validator. For instance, if another nominator had their stake split 50% to BC_1 and 50% to OV_1, they would receive a slash of 0.5% (50% of 1%). If a nominator were actively nominating BC_1 and BC_2, again with 50% of their stake allocated to each, they would still end up with a 1% slash, since a 1% slash is applied to both halves of their stake. Note that you cannot control the percentage of stake you have allocated to each validator or choose who your active validator will be (except in the trivial case of nominating a single validator). Staking allocations are controlled by the Phragmén algorithm.

Once a validator gets slashed, it goes into the state as an "unapplied slash". You can check this via Polkadot-JS Apps. The UI shows it per validator and then all the affected nominators along with the amounts. While unapplied, a governance proposal can be made to reverse it during this period (7 days on Kusama, 28 days on Polkadot). After the grace period, the slashes are applied.

The following levels of offence are defined. However, these particular levels are not implemented or referred to in the code or in the system; they are meant as guidelines for different levels of severity for offences. To understand how slash amounts are calculated, see the equations in the section below.

- Level 1: isolated unresponsiveness, i.e. being offline for an entire epoch. Generally no slashing, only *chilling*.
- Level 2: concurrent unresponsiveness or isolated equivocation. Slashes a very small amount of the stake and chills.
- Level 3: misconducts unlikely to be accidental, but which do not harm the network's security to any large extent. Examples include concurrent equivocation or isolated cases of unjustified voting in GRANDPA. Slashes a moderately small amount of the stake and chills.
- Level 4: misconduct that poses a serious security or monetary risk to the system, or mass collusion. Slashes all or most of the stake behind the validator and chills.

Let's look at these offences in a bit more detail.

Unresponsiveness

For every session, validators will send an "I'm Online" heartbeat to indicate they are online. If a validator produces no blocks during an epoch and fails to send the heartbeat, it will be reported as unresponsive. Depending on the repeated offences and how many other validators were unresponsive or offline during the epoch, slashing may occur.

Here is the formula for calculation:

```
Let x = offenders, n = total no. validators in the active set

min((3 * (x - (n / 10 + 1))) / n, 1) * 0.07
```

Let us run through a few examples to understand this equation. In all of the examples, assume that there are 100 validators in the active set.

Note that if less than 10% of all validators are offline, no penalty is enacted.

Validators should have a well-architected network infrastructure to ensure the node is running to reduce the risk of being slashed or chilled. A high availability setup is desirable, preferably with backup nodes that kick in **only once the original node is verifiably offline** (to avoid double-signing and being slashed for equivocation - see below). A comprehensive guide on validator setup is available here.

GRANDPA Equivocation

A validator signs two or more votes in the same round on different chains.

BABE Equivocation

A validator produces two or more blocks on the Relay Chain in the same time slot.

Both GRANDPA and BABE equivocation use the same formula for calculating the slashing penalty:

```
Let x = offenders, n = total no. validators in active set

Min( (3 * x / n )^2, 1)
```

As an example, assume that there are 100 validators in the active set, and one of them equivocates in a slot (for our purposes, it does not matter whether it was a BABE or GRANDPA equivocation). This is unlikely to be an attack on the network, but much more likely to be a misconfiguration of a validator. The penalty would be Min(3 * 1 / 100)^2, 1) = 0.0009, or a 0.09% slash for that validator pool (i.e., all stake held by the validator and its nominators).

Now assume that there is a group running several validators, and all of them have an issue in the same slot. The penalty would be Min((3 *5 / 100)^2, 1) = 0.0225, or a 2.25% slash. If 20 validators equivocate, this is a much more serious offence and possibly indicates a coordinated attack on the network, and so the slash will be much greater - Min((3* 20 / 100)^2, 1) = 0.36, or a 36% slash on all of these validators and their nominators. All slashed validators will also be chilled.

From the example above, the risk in nominating or running many validators in the active set are apparent. While rewards grow linearly (two validators will get you approximately twice as many staking rewards as one), slashing grows exponentially. A single validator equivocating causes a 0.09% slash, two validators equivocating does not cause a 0.09 * 2 = 0.18% slash, but rather a 0.36% slash - 4x as much as the single validator.

Validators may run their nodes on multiple machines to make sure they can still perform validation work in case one of their nodes goes down, but validator operators should be extremely careful in setting these up. If they do not have good coordination to manage signing machines, equivocation is possible, and equivocation offences are slashed at much higher rates than equivalent offline offences.

If a validator is reported for any one of the offences they will be removed from the validator set (chilled) and they will not be paid while they are out. They will be considered inactive immediately and will lose their nominators. They need to re-issue intent to validate and again gather support from nominators.

If you want to know more details about slashing, please look at our research page.

Chilling

Chilling is the act of stepping back from any nominating or validating. It can be done by a validator or nominator at any time themselves, taking effect in the next era. It can also specifically mean removing a validator from the active validator set by another validator, disqualifying them from the set of electable candidates in the next NPoS cycle.

Chilling may be voluntary and validator-initiated, e.g. if there is a planned outage in the validator's surroundings or hosting provider, and the validator wants to exit to protect themselves against slashing. When voluntary, chilling will keep the validator active in the current session, but will move them to the inactive set in the next. The validator will not lose their nominators.

When used as part of a punishment (initiated externally), being chilled carries an implied penalty of being un-nominated. It also disables the validator for the remainder of the current era and removes the offending validator from the next election.

Polkadot allows some validators to be disabled, but if the number of disabled validators gets too large, Polkadot will trigger a new validator election to get a full set. Disabled validators will need to resubmit their intention to validate and re-garner support from nominators.

For more on chilling, see the How to Chill page on this wiki.

Slashing Across Eras

There are 3 main difficulties to account for with slashing in NPoS:

- A nominator can nominate multiple validators and be slashed via any of them.
- Until slashed, stake is reused from era to era. Nominating with N coins for E eras in a row does not mean you have N*E coins to be slashed - you've only ever had N.
- Slashable offences can be found after the fact and out of order.

To balance this, we only slash for the maximum slash a participant can receive in some time period, rather than the sum. This ensures protection from overslashing. Likewise, the time span over which maximum slashes are computed are finite and the validator is chilled with nominations withdrawn after a slashing event, as stated in the previous section. This prevents rage-quit attacks in which, once caught misbehaving, a participant deliberately misbehaves more because their slashing amount is already maxed out.

Reward Distribution

Note that Kusama runs approximately 4x as fast as Polkadot, except for block production times. Polkadot will also produce blocks at approximately six second intervals.

Rewards are recorded per session (approximately one hour on Kusama and four hours on Polkadot) and calculated per era (approximately six hours on Kusama and twenty-four hours on Polkadot). Thus, rewards will be calculated four times per day on Kusama and once per day on Polkadot.

Rewards are calculated based on era points, which have a probabilistic component. In other words, there may be slight differences in your rewards from era to era, and even amongst validators in the active set at the same time. These variations should cancel out over a long enough timeline. See the page on Validator Payout Guide for more information on how these are calculated.

In order to be paid your staking rewards, someone must claim them for each validator that you nominate. Staking rewards are kept available for 84 eras, which is approximately 84 days on Polkadot and 21 days on Kusama. For more information on why this is so, see the page on simple payouts.

> WARNING: If nobody claims your staking rewards by this time, then you will not be able to claim them and some of your staking rewards will be lost. Additionally, if the validator unbonds all their own stake, any pending payouts will be lost. Since unbonding takes 28 days on Polkadot, nominators should check if they have pending payouts at least this often.

Claiming Rewards

If you go to the Staking payouts page on Polkadot-JS, you will see a list of all validators that you have nominated in the past 84 eras and for which you have not yet received a payout. Each one has the option to trigger the payout for all unclaimed eras. Note that this will pay everyone who was nominating that validator during those eras, and anyone can call it. Therefore, you may not see anything in this tab, yet still have received a payout if somebody (generally, but not necessarily, another nominator or the validator operator) has triggered the payout for that validator for that era.

If you wish to check if you received a payout, you will have to check via a block explorer. See the relevant Support page for details.

Reward Distribution Example

```
PER_ERA * BLOCK_TIME = **Reward Distribution Time**

3_600 * 6 seconds = 21_600 s = 6 hours
```

Validators can create a cut of the reward (a commission) that is not shared with the nominators. This cut is a percentage of the block reward, not an absolute value. After the commission gets deducted, the remaining portion is based on their staked value and split between the validator and all of the nominators who have voted for this validator.

For example, assume the block reward for a validator is 10 DOT. A validator may specify `validator_commission = 50%`, in which case the validator would receive 5 DOT. The remaining 5 DOT would then be split between the validator and their nominators based on the proportion of stake each nominator had. Note that validators can put up their own stake, and for this calculation, their stake acts just as if they were another nominator.

Rewards can be directed to the same account (controller), to the stash account (and either increasing the staked value or not increasing the staked value), or to a completely unrelated account. By using the Extrinsics tab (`Developer -> Extrinsics -> Staking -> Bond`) you can also send rewards to "None", effectively burning them. It is also possible to top-up / withdraw some bonded DOT without having to un-stake all staked DOT.

For specific details about validator payouts, please see this guide.

Inflation

DOT is inflationary; there is no maximum number of DOT. Inflation is designed to be approximately 10% annually, with validator rewards being a function of the amount staked and the remainder going to treasury. The current token supply of DOT is ~1,000,000,000, as a result of redenomination.

There is an *ideal staking rate* that the network tries to maintain. The goal is to have the *system staking rate* meet the *ideal staking rate*.

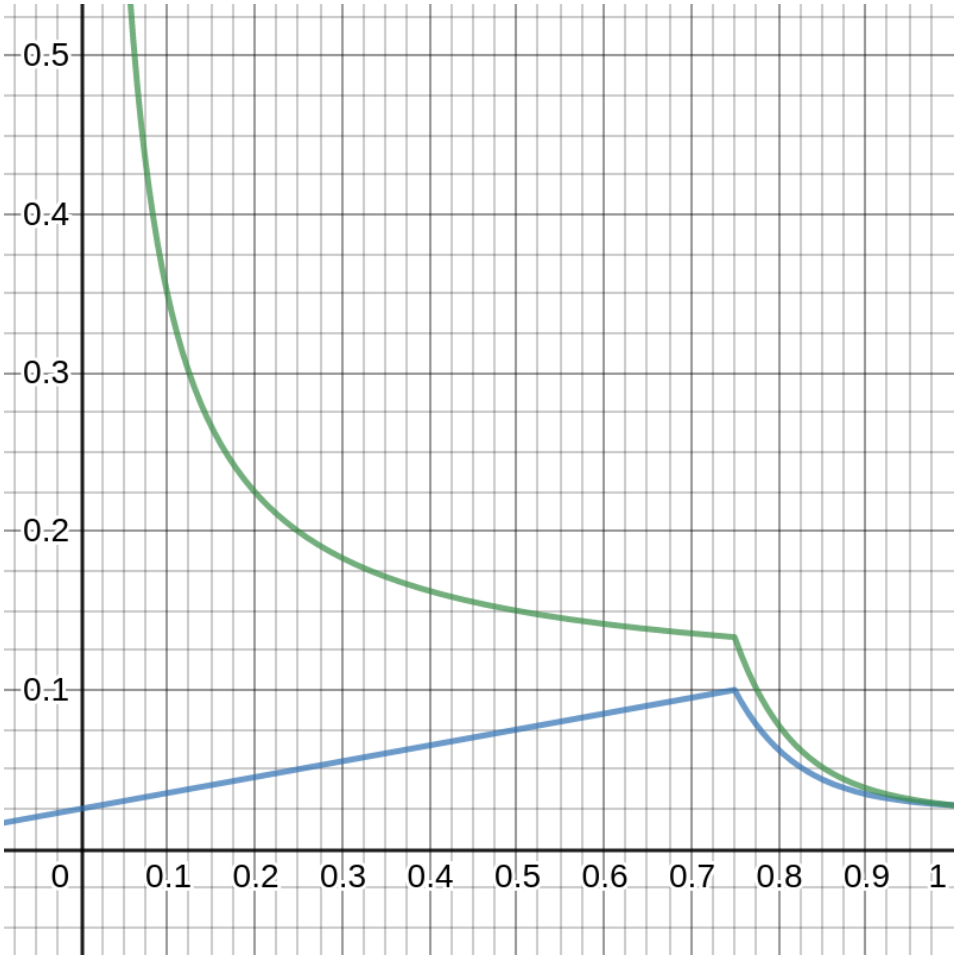
The system staking rate would be the total amount staked over the total token supply, where the total amount staked is the stake of all validators and nominators on the network. The ideal staking rate accounts for having sufficient backing of DOT to prevent the possible compromise of security while keeping the native token liquid. An **ideal staking rate of 50% stabilizes the network**. DOT is inflated according to the system staking rate of the entire network.

> According to the inflation model, this would suggest that if you do not use your DOT for staking, your tokens dilute over time.

The ideal staking rate on Polkadot also varies with the number of parachains (50% is the current estimation of all DOT that should be staked, per parachain slot).

> In the **absence of parachains, the suggested ideal staking rate is 75%**, as liquidity is not constrained by locked parachain bonds.

If the amount of tokens staked goes below the ideal rate, then staking rewards for nominators goes up. On the contrary, if it goes above, staking rewards drop. This is a result of the change in the percentage of staking rewards that go to the Treasury.



Source: Research - Web3 Foundation

- **x-axis**: Proportion of DOT staked
- **y-axis**: Inflation, annualized percentage
- **Blue line**: Inflation rewards to stakers
- **Green line**: Staker rate of return

You can determine the inflation rewards by checking the staking overview on Polkadot-JS Apps.

The above chart shows the inflation model of the network. Depending on the staking participation, the distribution of the inflation to validators/nominators versus the treasury will change dynamically to provide incentives to participate (or not participate) in staking.

For instance, assuming that the ideal staking rate is 50%, all of the inflation would go to the validators/nominators if 50% of all KSM / DOT are staked. Any deviation from the 50% - positive or negative - sends the proportional remainder to the treasury and effectively reduces staking rewards.

For those who are interested in knowing more about the design of inflation model for the network, please see here.

Why stake?

- 10% inflation/year when the network launches
- 50% targeted active staking

- ~20% annual nominal return

Up until now, the network has been following an inflation model that excludes the metric of active parachains. The ideal staking rate is not always 50%, as the number of active parachains influences the available liquidity that is available to secure the network.

Keep in mind that when the system's staking rate is lower than the ideal staking rate, the annual nominal return rate will be higher than 20%, encouraging more users to use their tokens for staking. On the contrary, when the system staking rate is higher than the ideal staking rate, the annual nominal return of will be less than 20%, encouraging some users to withdraw.

Why not stake?

- Tokens will be locked for about 28 days on Polkadot after unbonding, seven days on Kusama.
- Punishment in case of validator found to be misbehaving (see #slashing).
- You want to use the tokens for a parachain slot.

How many validators does Polkadot have?

Polkadot started with 20 open validator positions and has increased gradually to 297. The top bound on the number of validators has not been determined yet, but should only be limited by the bandwidth strain of the network due to peer-to-peer message passing. The estimate of the number of validators that Polkadot will have at maturity is around 1000. Kusama, Polkadot's canary network, currently has 900 validator slots in the active set.

Motion #108: New Minimum Nomination Bond

Motion #108 proposed new nomination limits to the Polkadot network, offering a temporary solution to increase the stability and security of the network. Note that this motion **does not** increase the maximum nominator count.

The goal of this motion is to increase the minimum nomination bond, allowing new nominators that meet this requirement to participate in the network's security. This motion will update the value of the minimum nominator bond from 80 DOTs to 120 DOTs. Prior to this, Motion #103 set a new parameter named `chill-threshold`. With `chill-threshold`, the permissionless `chill_other` may only be executed if, and only if, the current nominator count is greater than 90% of the maximum number of nominators. Any existing nominator can update their nomination preferences (amount of DOT bonded) to adjust to this change. A more permanent solution for lowering the minimum bond requirement for nominators is in progress.

Parameters changed: minimum nominator bond : `80` -> `120`

Parameters added: chill-threshold: `90%`

Resources

- How Nominated Proof of Stake will work in Polkadot - Blog post by Web3 Foundation researcher Alfonso Cevallos covering NPoS in Polkadot.
- Validator setup

Edit this page

*Last updated on **11/4/2021** by **Danny Salman***

Proxy Accounts

Polkadot provides a module that allows users to set proxy accounts to perform a limited number of actions on their behalf. Much like the Stash and Controller account relationship in staking, proxies allow users to keep one account in cold storage and actively participate in the network with the weight of the tokens in that account.

> Check out our Polkadot YouTube video that explains what are proxies.

Proxy Types

You can set a proxy account via the Proxy module. When you set a proxy, you must choose a type of proxy for the relationship. Polkadot offers:

- Any
- Non-transfer
- Governance
- Staking
- Identity Judgement
- Auction

When a proxy account makes a `proxy` transaction, Polkadot filters the desired transaction to ensure that the proxy account has the appropriate permission to make that transaction on behalf of the cold account.

Any Proxies

As implied by the name, a proxy type of "Any" allows the proxy account to make any transaction, including balance transfers. In most cases, this should be avoided as the proxy account is used more frequently than the cold account and is therefore less secure.

Non-transfer Proxies

Proxies that are of the type "Non-transfer" are accounts that allow any type of transaction except balance transfers (including vested transfers).

Governance Proxies

The "Governance" type will allow proxies to make transactions related to governance (i.e., from the Democracy, Council, Treasury, Technical Committee, and Elections pallets).

> See Governance for more information on governance proxies or watch our technical explainer video that explores this concept.

Staking Proxies

The "Staking" type allows staking-related transactions, but do not confuse a staking proxy with the Controller account. Within the Staking pallet, some transactions must come from the Stash, while others must come from the Controller. The Stash account is meant to stay in cold storage, while the Controller account makes day-to-day transactions like setting session keys or deciding which validators to nominate. The Stash account still

needs to make some transactions, though, like bonding extra funds or designating a new Controller. A proxy doesn't change the *roles* of Stash and Controller accounts, but does allow the Stash to be accessed even less frequently.

Identity Judgement Proxies

"Identity Judgement" proxies are in charge of allowing registrars to make judgement on an account's identity. If you are unfamiliar with judgements and identities on chain, please refer to this page.

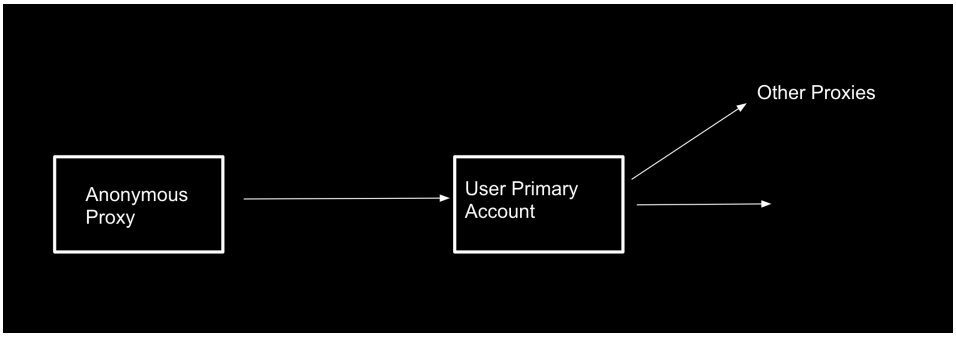
Auction Proxies

Proxies that are of the type "Auction" are accounts that allow transactions pertaining to parachain auctions and crowdloans. The Auction proxy account can sign those transactions on behalf of an account in cold storage. If you already setup a "Non-transfer" proxy account, it can do everything an "Auction" proxy can do. Before participating in a crowdloan using an Auction proxy, it is recommended that you check with the respective parachain team for any possible issues pertaining to the crowdloan rewards distribution.

Anonymous Proxies

Polkadot includes a function to create an anonymous proxy, an account that can only be accessed via proxy. That is, it generates an address but no corresponding private key. Normally, a primary account designates a proxy account, but anonymous proxies are the opposite. The account that creates the proxy relationship is the proxy account and the new account is the primary. Use extreme care with anonymous proxies; once you remove the proxy relationship, the account will be inaccessible.

> Learn more about anonymous proxies from our technical explainer video.



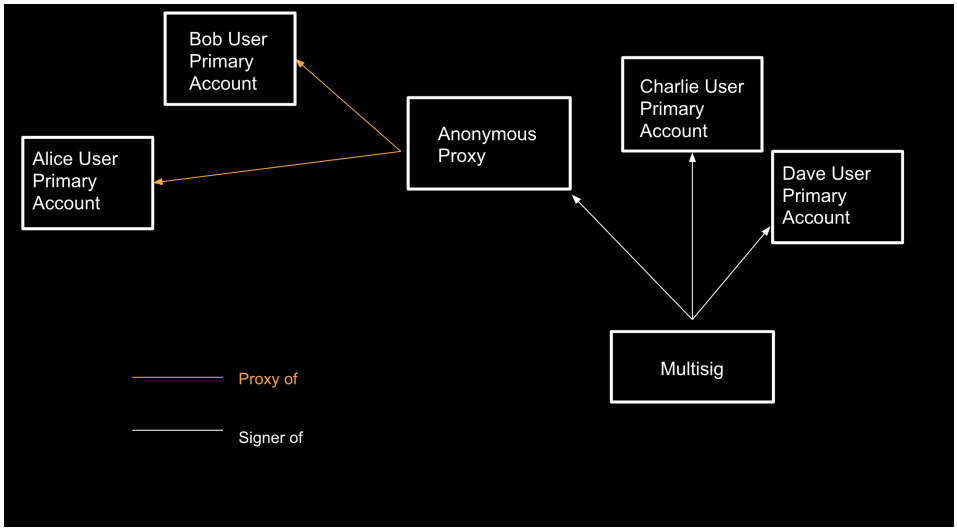
Time Delayed Proxies

We can add an additional layer of security to proxies by giving them a delay time. The delay will be quantified in number of blocks (blockNumber). Polkadot and Kusama both have 6 second blocks, hence a delay value of 10 will mean 10 blocks which will equal 1 minute of delay. The proxy will announce it's intended action and wait for the number of blocks defined in the delay time before executing it. The proxy will include the hash of the intended function call in the announcement. Within this time window, the intended action may be cancelled by accounts that control the proxy. Now we can use proxies knowing that any malicious actions can be noticed and reverted within a delay period.

Why use a Proxy?

Proxies are great to use for specific purposes because they add in a layer of security. Rather than using funds in one sole account, smaller accounts with unique roles complete tasks for the main stash account. This drives attention away from the main account and to proxies.

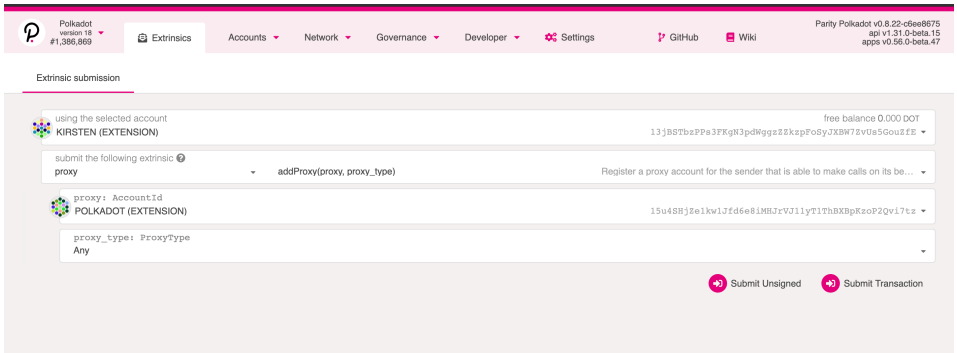
Anonymous proxies, in particular, can be used for permissionless management. In this example below, there is a multisig with four different accounts inside. Two of the accounts, Alice and Bob, have an anonymous proxy attached to them. In the case that the multisig account wanted to add or remove Alice or Bob or even add in a new account into the anonymous proxy, the anonymous proxy would take care of that change. If a multisig wanted to modify itself without an anonymous proxy, a whole new multisig would be created.



How to set up a Proxy

Using the Polkadot-JS UI

To set up a proxy, navigate to the Polkadot-JS UI and click on "Developer" > "Extrinsics". Here we will see a page that looks similar to this:



To add a proxy, click on the pallet selection dropdown menu. The dropdown is labeled "submit the following extrinsic". Select the `proxy` pallet, then the `addProxy` extrinsic (in the dropdown menu next to it). The `addProxy(proxy, proxy_type)` function will need to be selected in order to add in a proxy. The chosen proxy account that you set will be the account that has the proxy on it. The selected account at the top is the account that will be the primary account.

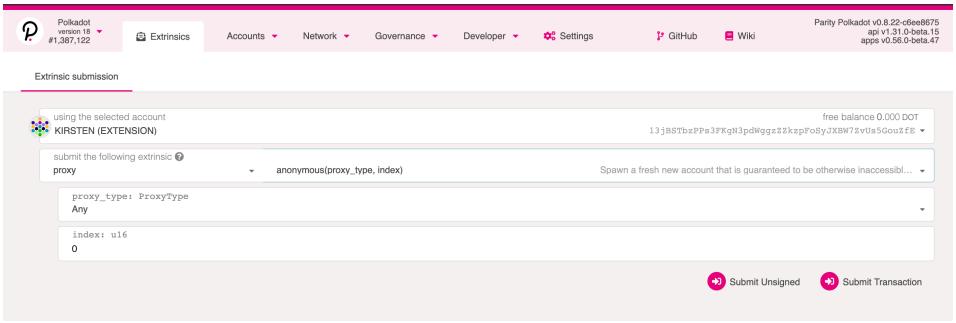
> NOTE: If you see an `unused` option when adding in a proxy, this is not a proxy type. This is an empty `enum`, and if you try to add this in as a proxy, nothing will happen. No new proxy will be created.

It is critical to setup Anonymous Proxies with appropriate permissions and be aware of potential dangers

Creating Anonymous Proxies on Polkadot-JS UI

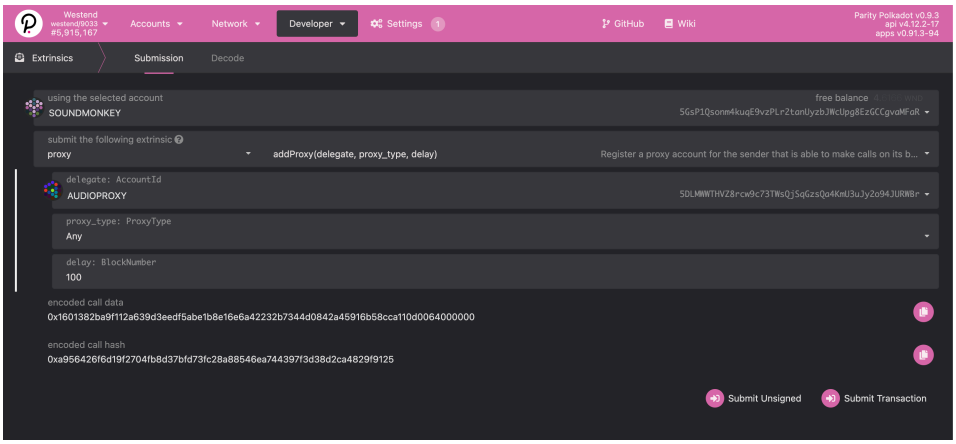
> NOTE: The first anonymous proxy you add should always be of *type* `Any`. Also, if there are multiple anonymous proxies for the proxied account, you should keep at least *one* `Any` type. Without having an `Any` type proxy, you won't be able to send funds, add new proxies, kill the anonymous proxy or take any action not specifically allowed by the types of the proxies the account has.

For anonymous proxies, a different function will need to be called, the `anonymous(proxy_type, index)`. This will let you select which kind of anonymous proxy you would like to set up if you choose, as well as the index.



Using Time Delayed Proxies

When creating a proxy through the PolkadotJS application, we are provided a delay field. In this example we are creating a proxy with a delay value of 100, which means 100 blocks. 100 * 6(minutes) = 600 minutes, or 10 hours.



Another way to create Proxies

There is another way you can set up a proxy on Polkadot-JS UI. Go to "Accounts" in the navigation and then click the "Accounts" button. For each of the accounts you have on this page, the three dot button will let you create a proxy by using "Add proxy". This will

open up a pop up onto your screen where you will be able to select the type of proxy for that specific account.

> NOTE: You cannot create an anonymous function from the Accounts page, you must be on the Extrinsics page.

Removing Proxies

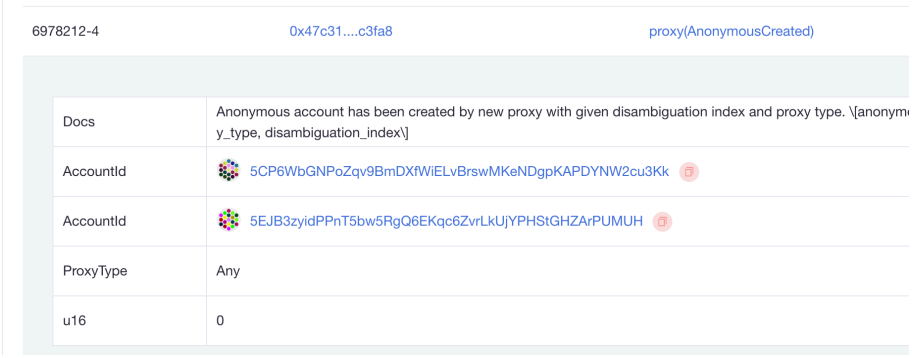
If you want to remove a proxy, there are a few functions on the extrinsic page that will help do this.

For non-anonymous proxies, you can use `removeProxy` or `removeProxies`, but must use the `killAnonymous` function for anonymous proxies. This must be called **from** the *anonymous* proxy. This means that the anonymous proxy must be added as an account to Polkadot-JS accounts.

The following steps can be used to remove your proxy:

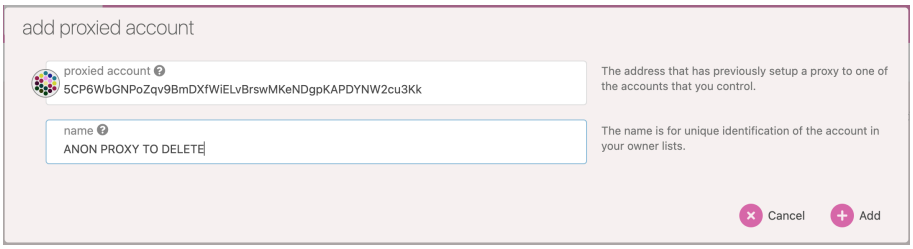
> WARNING: there is no way to get access to the proxy after deleting it.

- **Step 0**: You need to know the following information:
  - the **account** you created the anonymous proxy from
  - **type of proxy**, index (almost always 0)
  - **block height** it was created at
  - the **extrinsic index** in the block (on most block explorers, you will see the extrinsic ID listed as something along the lines of "9000-2" -> 9000 is the block height (block number) and 2 is the extrinsic index. You can find this information by looking up your account in a block explorer.



- **Step 1**: Go to https://polkadot.js.org/apps/#/accounts (make sure you are on correct network).

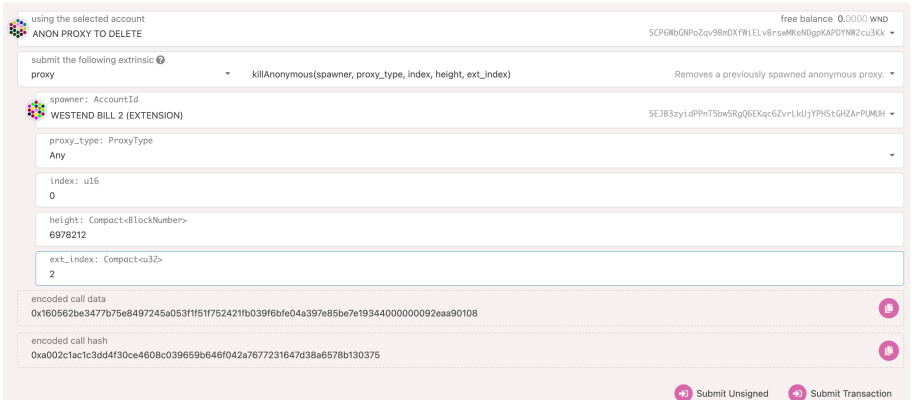
- **Step 2**: Click `Proxied` and add your address, name it `ANON PROXY`. You should now see this address in accounts. Now you need to call `killAnonymous` from the anonymous proxy. It is important to note that anonymous proxies *work backwards*; the original account acts as the proxy.



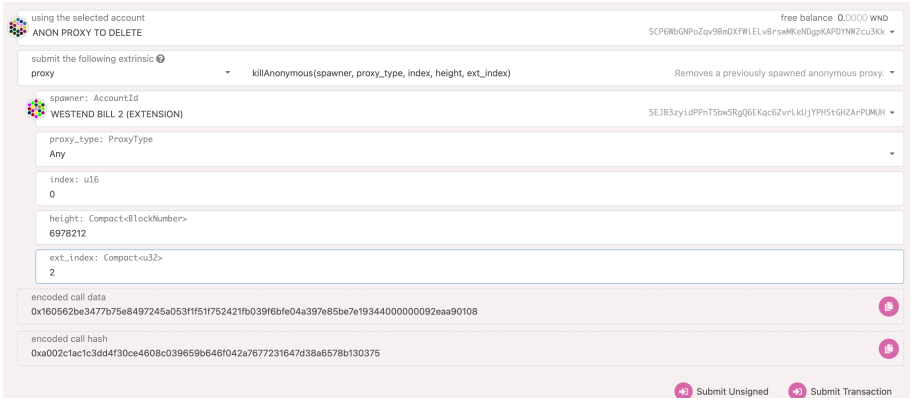
- **Step 3**: Go to https://polkadot.js.org/apps/#/extrinsics

- **Step 4**: Call extrinsic `proxy.killAnonymous` using the selected account ANON PROXY and the following parameters:

  - Spawner: (original account)
  - Proxy type (kind of proxy)
  - Index 0 (almost always, but can be seen in creating extrinsic)
  - Block number x
  - Extrinsic index y

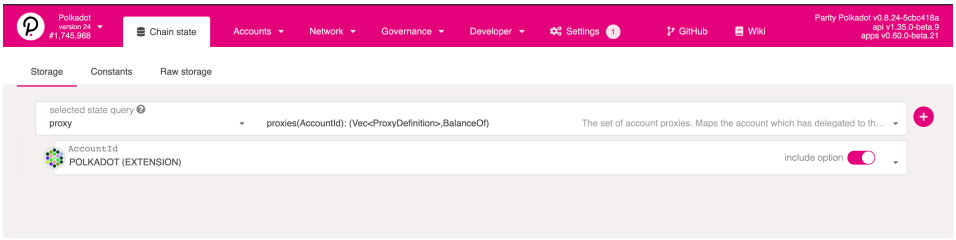


- **Step 5**: Submit and sign extrinsic



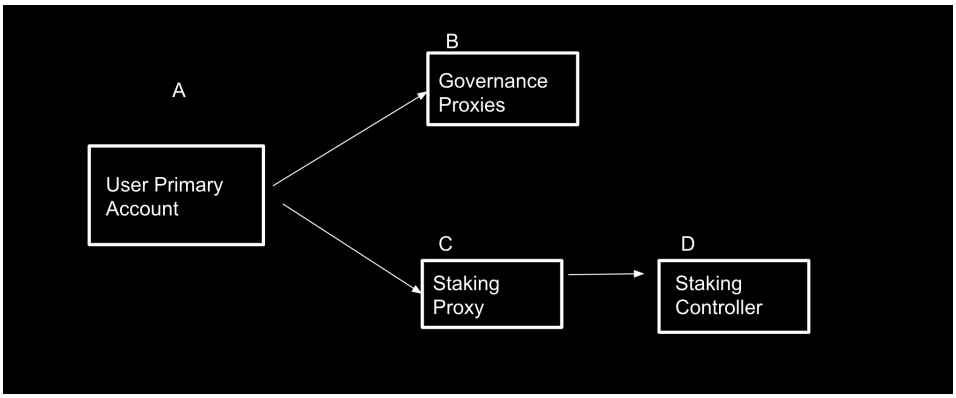
How to view your Proxies

To view your proxy, head over to the Chain State (underneath "Developer") page on Polkadot-JS Apps. If you've created your proxy on a Kusama account, it is required to change your network accordingly using the top left navigation button. On this page, the proxy pallet should be selected, returning the announcements and proxies functions. The proxies function will allow you to see your created proxies for either one account or for all accounts (using the toggle will enable this). Proxy announcements are what time lock proxies do to announce they are going to conduct an action.



Putting It All Together

If the idea of proxy types and their application seems abstract, it is. Here is an example of how you might use these accounts. Imagine you have one account as your primary token-holding account, and don't want to access it very often, but you do want to participate in governance and staking. You could set Governance and Staking proxies.



In this example, the primary account A would only make two transactions to set account B as its governance proxy and account C as its staking proxy. Now, account B could participate in governance activity on behalf of A.

Likewise, account C could perform actions typically associated with a stash account, like bonding funds and setting a Controller, in this case account D. Actions that normally require the Stash, like bonding extra tokens or setting a new Controller, can all be handled by its proxy account C. In the case that account C is compromised, it doesn't have access to transfer-related transactions, so the primary account could just set a new proxy to replace it.

By creating multiple accounts that act for a single account, it lets you come up with more granular security practices around how you protect private keys while still being able to actively participate in a network.

Proxy Deposits

Proxies require deposits in the native currency (i.e. DOT or KSM) in order to be created. The deposit is required because adding a proxy requires some storage space on-chain, which must be replicated across every peer in the network. Due to the costly nature of this, these functions could open up the network to a Denial-of-Service attack. In order to defend against this attack, proxies require a deposit to be reserved while the storage space is consumed over the life time of the proxy. When the proxy is removed, so is the storage space, and therefore the deposit is returned.

The deposits are calculated in the runtime, and the function can be found in the runtime code. For example, the deposits are calculated in Polkadot with the following functions:

```
// One storage item; key size 32, value size 8; .
pub const ProxyDepositBase: Balance = deposit(1, 8);
// Additional storage item size of 33 bytes.
pub const ProxyDepositFactor: Balance = deposit(0, 33);
```

The `ProxyDepositBase` is the required amount to be reserved for an account to have a proxy list (creates one new item in storage). For every proxy the account has, an additional amount defined by the `ProxyDepositFactor` is reserved as well (appends 33 bytes to storage location).

On Polkadot the `ProxyDepositBase` is 20.008 and the `ProxyDepositFactor` is 0.033.

So what this boils down to is that the required deposit amount for one proxy on Polkadot is equal to (in DOT):

```
20.008 + 0.033 * num_proxies
```

Edit this page

*Last updated on **10/26/2021** by **Radha***

Availability and Validity

The Availability and Validity (AnV) protocol of Polkadot is what allows for the network to be efficiently sharded among parachains while maintaining strong security guarantees.

Phases of the AnV protocol

There are five phases of the Availability and the Validity protocol.

1. Parachain phase.
2. Relay Chain submission phase.
3. Availability and unavailability subprotocols.
4. Secondary GRANDPA approval validity checks.
5. Invocation of a Byzantine fault tolerant *finality gadget* to cement the chain.

Parachain phase

The parachain phase of AnV is when the *collator* of a parachain proposes a *candidate block* to the validators that are currently assigned to the parachain.

> A **candidate block** is a new block from a parachain collator that may or may not be valid and must go through validity checks before being included into the Relay Chain.

Relay Chain submission phase

The validators then check the candidate block against the verification function exposed by that parachain's registered code. If the verification succeeds, then the validators will pass the candidate block to the other validators in the gossip network. However, if the verification fails, the validators immediately reject the candidate block as invalid.

When more than half of the parachain validators agree that a particular parachain block candidate is a valid state transition, they prepare a *candidate receipt*. The candidate receipt is what will eventually be included into the Relay Chain state. It includes:

- The parachain ID.
- The collator's ID and signature.
- A hash of the parent block's candidate receipt.
- A Merkle root of the block's erasure-coded pieces.
- A Merkle root of any outgoing messages.
- A hash of the block.
- The state root of the parachain before block execution.
- The state root of the parachain after block execution.

This information is **constant size** while the actual PoV block of the parachain can be variable length. It is enough information for anyone that obtains the full PoV block to verify the state transition contained inside of it.

Availability and unavailability subprotocols

During the availability and unavailability phases, the validators gossip the erasure coded pieces among the network. At least 1/3 + 1 validators must report that they possess their piece of the code word. Once this threshold of validators has been reached, the network can consider the PoV block of the parachain *available*.

Erasure Codes

Erasure coding transforms a message into a longer *code* that allows for the original message to be recovered from a subset of the code and in absence of some portion of the code. A code is the original message padded with some extra data that enables the reconstruction of the code in the case of erasures.

The type of erasure codes used by Polkadot's availability scheme are Reed-Solomon codes, which already enjoys a battle-tested application in technology outside the blockchain industry. One example is found in the compact disk industry. CDs use Reed-Solomon codes to correct any missing data due to inconsistencies on the disk face such as dust particles or scratches.

In Polkadot, the erasure codes are used to keep parachain state available to the system without requiring all validators to keep tabs on all the parachains. Instead, validators share smaller pieces of the data and can later reconstruct the entire data under the assumption that 1/3+1 of the validators can provide their pieces of the data.

Note: The 1/3+1 threshold of validators that must be responsive in order to construct the full parachain state data corresponds to Polkadot's security assumption in regard to Byzantine nodes.

Fishermen: Deprecated

The idea of Fishermen is that they are full nodes of parachains, like collators, but perform a different role in relation to the Polkadot network. Instead of packaging the state transitions and producing the next parachain blocks as collators do, fishermen will watch this process and ensure no invalid state transitions are included.

Fishermen are not available on Kusama or Polkadot and are not planned for formal implementation, despite previous proposals in the AnV protocol.

To address the motivation behind the Fishermen design consideration, the current secondary backing checkers perform a similar role in relation to the Polkadot network. From a security standpoint, security is based on having at least one honest validator either among parachain validators or secondary checker.

Further Resources

- Path of a Parachain Block - Article by Parity analyst Joe Petrowski expounding on the validity checks that a parachain block must pass in order to progress the parachain.
- Availability and Validity - Paper by the W3F Research Team that specifies the availability and validity protocol in detail.

Edit this page

*Last updated on **10/23/2021** by **Danny Salman***

Randomness

Randomness in Proof of Stake blockchains is important for a fair and unpredictable distribution of validator responsibilities. Computers are bad at random numbers because they are deterministic devices (the same input always produces the same output). What people usually call random numbers on a computer (such as in a gaming application), are *pseudo-random* - that is, they depend on a sufficiently random *seed* provided by the user or another type of *oracle*, like a weather station for atmospheric noise, your heart rate, or even lava lamps, from which it can generate a series of seemingly-random numbers. But given the same seed, the same sequence will always be generated.

Though, these inputs will vary based on time and space, and it would be impossible to get the same result into all the nodes of a particular blockchain around the world. If nodes get different inputs on which to build blocks, forks happen. Real-world entropy is not suitable for use as a seed for blockchain randomness.

There are two main approaches to blockchain randomness in production today: `RANDAO` and `VRF`.

Polkadot uses VRF.

VRF

A verifiable random function (VRF) is a mathematical operation that takes some input and produces a random number along with a proof of authenticity that this random number was generated by the submitter. The proof can be verified by any challenger to ensure the random number generation is valid.

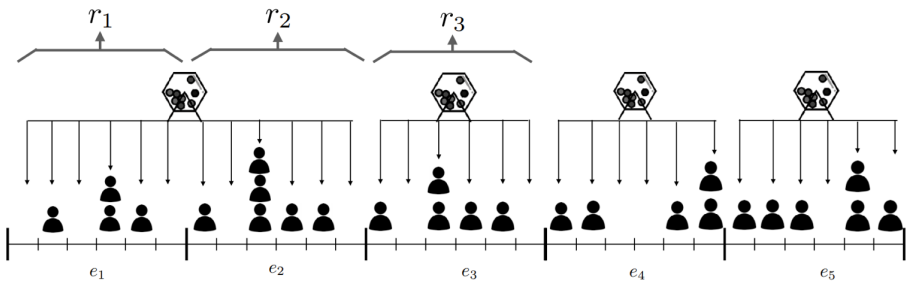
The VRF used in Polkadot is roughly the same as the one used in Ouroboros Praos. Ouroboros randomness is secure for block production and works well for BABE. Where they differ is that Polkadot's VRF does not depend on a central clock (the problem becomes - whose central clock?), rather, it depends on its own past results to determine present and future results, and it uses slot numbers as a clock emulator, estimating time.

Here's how it works in detail:

Slots are discrete units of time six seconds in length. Each slot can contain a block, but may not. Slots make up epochs - on Polkadot, 2400 slots make one epoch, which makes epochs four hours long.

In every slot, each validator "rolls a die". They execute a function (the VRF) that takes as input the following:

- **The "secret key",** a key specifically made for these die rolls.
- **An epoch randomness value,** which is the hash of VRF values from the blocks in the epoch before last (N-2), so past randomness affects the current pending randomness (N).
- **The slot number.**



The output is two values: a `RESULT` (the random value) and a `PROOF` (a proof that the random value was generated correctly).

The `RESULT` is then compared to a *threshold* defined in the implementation of the protocol (specifically, in the Polkadot Host). If the value is less than the threshold, then the validator who rolled this number is a viable block production candidate for that slot. The validator then attempts to create a block and submits this block into the network along with the previously obtained `PROOF` and `RESULT`. Under VRF, every validator rolls a number for themselves, checks it against a threshold, and produces a block if the random roll is under that threshold.

The astute reader will notice that due to the way this works, some slots may have no validators as block producer candidates because all validator candidates rolled too high and missed the threshold. We clarify how we resolve this issue and make sure that Polkadot block times remain near constant-time in the wiki page on consensus.

RANDAO

An alternative method for getting randomness on-chain is the RANDAO method from Ethereum. RANDAO requires each validator to prepare by performing many thousands of hashes on some seed. Validators then publish the final hash during a round and the random number is derived from every participant's entry into the game. As long as one honest validator participates, the randomness is considered secure (non-economically viable to attack).

RANDAO is optionally augmented with VDF.

VDFs

Verifiable Delay Functions are computations that take a prescribed duration of time to complete, even on parallel computers. They produce unique output that can be independently and efficiently verified in a public setting. By feeding the result of RANDAO into a VDF, a delay is introduced that renders any attacker's attempt at influencing the current randomness obsolete.

VDFs will likely be implemented through ASIC devices that need to be run separately from the other types of nodes. Although only one is enough to keep the system secure, and they will be open source and distributed at nearly no charge, running them is neither cheap nor incentivized, producing unnecessary friction for users of the blockchains opting for this method.

Resources

- Polkadot's research on blockchain randomness and sortition - contains reasoning for

choices made along with proofs

- Discussion on Randomness used in Polkadot - W3F researchers discuss the randomness in Polkadot and when it is usable and under which assumptions.

Edit this page

*Last updated on **9/17/2021** by **Danny Salman***

Cross-Consensus Message Format (XCM)

What started as an approach to *cross-chain communication*, has evolved into a format for **Cross-Consensus Communication** that is not only conducted between chains, but also smart contracts, pallets, bridges, and even sharded enclaves like SPREE.

Overview of XCM: A Format, Not a Protocol

XCM is related to cross-chain in the same way that REST is related RESTful. XCM cannot actually send messages between systems. It is a format for how message transfer should be performed, similar to how RESTful services use REST as an architectural style of deployment.

XCM aims to be a language communicating ideas between consensus systems, hence, "Cross-Consensus" with the following properties:

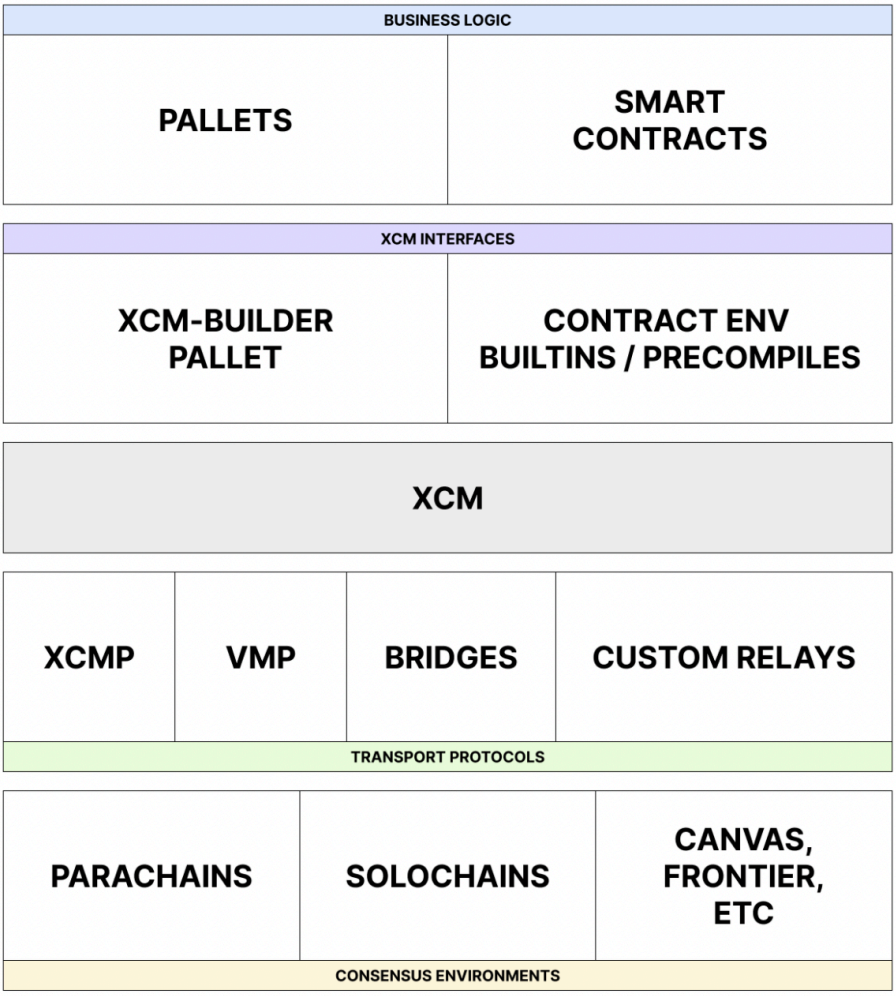
- Generic and extensible for use with fee-free and gas-metered smart contract platforms, community parachains, trusted interactions between system parachains and their relay chain, and more.
- Interacting with a system whose transaction format is unknown.
  - XCM is well-versioned, abstract and general and can be used as a means of providing a long-lasting transaction format for wallets to use to create many common transactions. It is *extensible*, and, in turn, *future-proof* and *forwards-compatible*.
- Highly efficient to operate in a tighly contrained and metered environment, as is the case with many chains.

> XCM is not designed in that every system supporting the format is expected to be able to interpret any possible XCM message. Practically speaking, one can imagine that some messages will not have reasonable interpretations under some systems or will be intentionally unsupported.

Example Use-Cases

- Request for specific operations to occur on the recipient system.
- Optionally include payment of fees on target network for requested operation.
- Provide methods for various token transfer models:
  - **Remote Transfers**: control an account on a remote chain, allowing the local chain to have an address on the remote chain for receiving funds and to eventually transfer those funds it controls into other accounts on that remote chain.
  - **Teleporting**: movement of an asset happens by destroying it on one side and creating a clone on the other side.
  - **Reverse-Based Transfer**: there may be two chains that want to nominate a third chain, where one includes a native asset that can be used as a reserve for that asset. Then, the derivative form of the asset on each of those chains would be fully backed, allowing the derivative asset to be exchanged for the underlying asset on the reserve chain backing it.

XCM Tech Stack



> XCM can be used to express the meaning of the messages over each of these three communication channels.

Cross-Consensus Protocols

With the XCM format established, common patters for protcols these messages are needed. Polkadot impliments two for acting on XCM messages between its constituent parachains.

VMP (Vertical Message Passing)

There are two kinds of vertical message-passing transport protocols:

- **UMP (Upward Message Passing)**: allows parachains to send messages to their relay chain.
- **DMP (Downward Message Passing)**: allows the relay chain to pass messages down to one of their parachains.

Messages that are passed via `DMP` may originate from a parachain. In which case, first `UMP` is used to communicate the message to the Relay Chain and `DMP` is used to move it down to another parachain.

XCMP (Cross-Chain Message Passing)

XCMP allows the parachains to exchange messages with other parachains on the same Relay Chain.

Cross-chain transactions are resolved using a simple queuing mechanism based around a Merkle tree to ensure fidelity. It is the task of the Relay Chain validators to move transactions on the output queue of one parachain into the input queue of the destination parachain. However, only the associated metadata is stored as a hash in the Relay Chain storage.

The input and output queue are sometimes referred to in the Polkadot codebase and associated documentation as `ingress` and `egress` messages respectively.

XCMP-Lite (HRMP)

While XCMP is still being implemented, a stop-gap protocol (see definition below) known as **Horizontal Relay-routed Message Passing (HRMP)** exists in its place. HRMP has the same interface and functionality as XCMP but is much more demanding on resources since it stores all messages in the Relay Chain storage. When XCMP has been implemented, HRMP is planned to be deprecated and phased out in favor of it.

> Note: A stop-gap protocol is a temporary substitute for the functionality that is not fully complete. While XCMP proper is still in development, HRMP is a working replacement.

XCMP Design

> XCMP is currently under development and the details are subject to change!

However, these overall architecture and design decisions are more stable:

- Cross-chain messages will *not* be delivered to the Relay Chain.
- Cross-chain messages will be constrained to a maximum size in bytes.
- Parachains are allowed to block messages from other parachains, in which case the dispatching parachain would be aware of this block.
- Collator nodes are responsible for routing messages between chains.
- Collators produce a list of `egress` messages and will receive the `ingress` messages from other parachains.
- On each block, parachains are expected to route messages from some subset of all other parachains.
- When a collator produces a new block to hand off to a validator, it will collect the latest ingress queue information and process it.
- Validators will check the proof that the new candidate for the next parachain block includes the processing of the expected ingress messages to that parachain.

XCMP queues must be initiated by first opening a channel between two parachains. The channel is identified by both the sender and recipient parachains, meaning that it's a one-way channel. A pair of parachains can have at most two channels between them, one for sending messages to the other chain and another for receiving messages. The channel will require a deposit in DOT to be opened, which will get returned when the channel is closed.

XCMP Message Format

For an updated and complete description of the XCMP message format please see the xcm-format repository on GitHub.

The Anatomy of an XCMP Inteaction

A smart contract that exists on parachain `A` will route a message to parachain `B` in which another smart contract is called that makes a transfer of some assets within that chain.

Charlie executes the smart contract on parachain `A`, which initiates a new cross-chain message for the destination of a smart contract on parachain `B`.

The collator node of parachain `A` will place this new cross-chain message into its outbound messages queue, along with a `destination` and a `timestamp`.

The collator node of parachain `B` routinely pings all other collator nodes asking for new messages (filtering by the `destination` field). When the collator of parachain `B` makes its next ping, it will see this new message on parachain `A` and add it into its own inbound queue for processing into the next block.

Validators for parachain `A` will also read the outbound queue and know the message. Validators for parachain `B` will do the same. This is so that they will be able to verify the message transmission happened.

When the collator of parachain `B` is building the next block in its chain, it will process the new message in its inbound queue as well as any other messages it may have found/received.

During processing, the message will execute the smart contract on parachain `B` and complete the asset transfer like intended.

The collator now hands this block to the validator, which itself will verify that this message was processed. If the message was processed and all other aspects of the block are valid, the validator will include this block for parachain `B` into the Relay Chain.

Check out our animated video below that explores how XCMP works.

XCVM (Cross-Consensus Virtual Machine)

An ultra-high level non-Turing-complete computer whose instructions are designed in a way to be roughly at the same level as transactions.

A *message* in XCM is simply just a programme that runs on the `XCVM`: in other words, one or more XCM instructions. To learn more about the XCVM and the XCM Format, see the latest blog post by Dr. Gavin Wood.

How To Make Cross-Chain Transfers

A tutorial on downward, upward, and lateral transfers can be found here.

Resources

- XCM: The Cross-Consensus Message Format - Detailed blog post by Dr. Gavin Wood about the XCM Format.
- XCM Format - Description of the high-level XCM format sent via XCMP.
- XCMP Scheme - Full technical description of cross-chain communication on the Web3 Foundation research wiki.
- Messaging Overview - An overview of the messaging schemes from the Parachain Implementor's guide.

Edit this page

*Last updated on **10/23/2021** by **Danny Salman***

SPREE

Shared Protected Runtime Execution Enclaves (SPREE) sometimes referred to as "trust wormholes," are fragments of logic comparable to runtime modules in Substrate, but live on the Polkadot Relay Chain and may be opted into by parachains.

SPREE in brief was described with the following properties and functions:

- Parachains can opt-in to special runtime logic fragments (like smart contracts).
- These fragments have their own storage and own XCM endpoint.
- All instances across parachains have identical logic.
- It executes alongside parachain logic.
- Protected: storage can not be altered by parachain logic; messages can not be faked from them by parachains.

Origin

On 28 March, 2019 u/Tawaren, a member of the Polkadot community, made a post on r/dot called "SmartProtocols Idea" and laid out a proposal for Smart Protocols. The core insight of the post was that XCMP had a complication in that it was difficult to verify and prove code was executed on a parachain without trust. A solution was to install the SmartProtocols in the Relay Chain that would be isolated blobs of code with their own storage per instance that could only be changed through an interface with each parachain. SmartProtocols are the precursor to SPREE.

What is a SPREE module?

SPREE modules are fragments of logic (in concrete terms they are blobs of WebAssembly code) that are uploaded onto Polkadot through a governance mechanism or by parachains. Once the blob is uploaded to Polkadot, all other parachains can decide to opt-in to the logic. The SPREE module would retain its own storage independent of the parachain, but would be callable through an interface with the parachain. Parachains will send messages to the SPREE module synchronously.

SPREE modules are important to the overall XCMP architecture because they give a guarantee to the code that will be executed on destination parachains. While XCMP guarantees the delivery of a message, it does not guarantee what code will be executed, i.e. how the receiving parachain will interpret the message. While XCMP accomplishes trustless message passing, SPREE is the trustless interpretation of the message and a key part of the usefulness of XCMP.

SPREE modules are like recipes in cookbooks. For example, if we give an order to a cook to make a soufflé, and we're decently confident in the ability of the cook, we have a vague idea of what will be made but no actual surety of how it will be made. However, let's say that a cook has the "Soufflé Maker's Manual" on their bookshelf and has committed themselves to only make souffles from this book. Now we can also consult the same book that the cook has, and we have a precise understanding of what will happen when we tell the cook to make a soufflé. In this example, "make a soufflé" was the message in XCMP and the cookbook was the SPREE module.

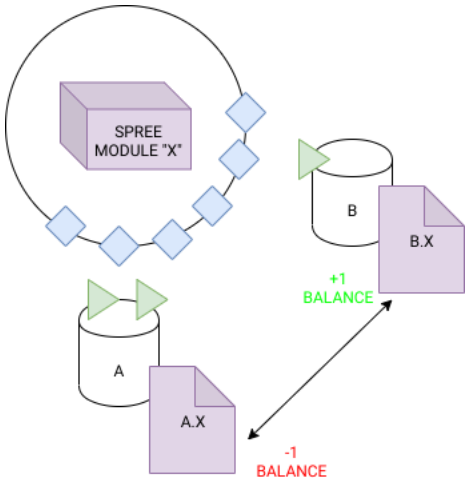
In concrete terms, SPREE modules could be useful for various functionality on Polkadot. One suggested use case of SPREE modules is for a trustless decentralized exchange that is offered as functionality to any parachain without any extra effort from parachain developers. One can imagine this working by having a SPREE module that exposes the interface for the incrementing and decrementing of balances of various assets based on a unique identifier.

Why?

Sending messages across parachains in XCMP only ensures that the message will be delivered but does not specify the code that will be executed, or how the message will be interpreted by the receiving parachain. There would be ways around this such as requesting a verifiable receipt of the execution from the receiving parachain, but in the naked case, the other parachain would have to be trusted. Having shared code that exists in appendices that the parachain can opt-in to resolves the need for trust and makes the execution of the appendices completely trustless.

SPREE would be helpful to ensure that the same logic is shared between parachains in the SPREE modules. An especially relevant use case would revolve around the use of token transfers across parachains in which it is important that the sending and receiving parachains agree about how to change the total supply of tokens and a basic interface.

Example



The diagram above is a simplification of the Polkadot system.

In this diagram, we see that the Wasm code for SPREE module "X" has been uploaded to the Polkadot Relay Chain. The two cylinders "A" and "B" represent two distinct parachains that have both opted-in to this SPREE module creating two distinct instances of it with their own XCMP endpoints "A.X" and "B.X".

In the example, we assume that this SPREE module "X" contains the functionality for incrementing of decrementing the balance of a particular asset that is unique to this module.

By initiating a transaction at A.X to decrease a particular balance by 1, a message over XCMP can be trustlessly sent to B.X to increase a balance by 1.

Collators, represented as the green triangle are responsible for relaying this message from parachain A to parachain B, as well as maintaining the storage for each particular instance of A.X and B.X for their respective parachains. They provide proofs of valid state transitions to the Relay Chain validators represented as blue diamonds.

Validators can validate the correct state transitions of SPREE modules A.X and B.X by being provided with the previous state root of the SPREE module instances, the data of the XCMP message between the instances, and the next state root of the instance. They do this validation by checking it against the `validate` function as provided by the SPREE module API. Collators are expected to be able to provide this information to progress their parachains.

Edit this page

*Last updated on **9/17/2021** by **Danny Salman***

WebAssembly (Wasm)

WebAssembly is used in Polkadot and Substrate as the compilation target for the runtime.

What is WebAssembly?

WebAssembly, shortened to simply *Wasm*, is a binary instruction format for a stack-based virtual machine. Wasm is designed as a portable target for compilation of high-level languages like C/C++/Rust, enabling deployment on the web for client and server applications.

Why WebAssembly?

WebAssembly is a platform agnostic binary format, meaning that it will run the same instructions across whatever machine it is operating on. Blockchains need determinacy in order to have reliable state transition updates across all nodes in the peer-to-peer network without forcing every peer to run the same exact hardware. Wasm is a nice fit for reliability among the possibly diverse set of machines. Wasm is both efficient and fast. The efficiency means that it can be uploaded onto the chain as a blob of code without causing too much state bloat while keeping its ability to execute at near-native speeds.

Forkless Upgrades

By using Wasm in Substrate, the framework powering Polkadot, Kusama, and many connecting chains, the chains are given the ability to upgrade their runtime logic without hard forking. Hard forking is a standard method of upgrading a blockchain that is slow, inefficient, and error prone due to the levels of offline coordination required, and thus, the propensity to bundle many upgrades into one large-scale event. By deploying Wasm on-chain and having nodes auto-enact the new logic at a certain block height, upgrades can be small, isolated, and very specific.

Resources

- WebAssembly.org - WebAssembly homepage that contains a link to the spec.
- Wasmi - WebAssembly interpreter written in Rust.
- Parity Wasm - WebAssembly serialization/deserialization in Rust.
- Wasm utils - Collection of Wasm utilities used in Parity and Wasm contract development.

Edit this page

*Last updated on **10/23/2021** by **Danny Salman***

Sequential Phragmén Method

What is the sequential Phragmén method?

The sequential Phragmén method is a multi-winner election method introduced by Edvard Phragmén in the 1890s. While sequential Phragmén is currently in use on Polkadot and Kusama, an improvement on the sequential Phragmén method named Phragmms will be used in the future.

The quote below taken from the reference Phragmén paper sums up the purpose of the sequential Phragmén method:

> The problem that Phragmén's methods try to solve is that of electing a set of a given numbers of persons from a larger set of candidates. Phragmén discussed this in the context of a parliamentary election in a multi-member constituency; the same problem can, of course, also occur in local elections, but also in many other situations such as electing a board or a committee in an organization.

Where is the Phragmén method used in Polkadot?

NPoS: Validator Elections

The sequential Phragmén method is used in the Nominated Proof-of-Stake scheme to elect validators based on their own self-stake and the stake that is voted to them from nominators. It also tries to equalize the weights between the validators after each election round. Since validators are paid equally in Polkadot, it is important that the stake behind each validator is spread out. Polkadot tries to optimize three metrics in its elections:

1. Maximize the total amount at stake.
2. Maximize the stake behind the minimally staked validator.
3. Minimize the variance of the stake in the set.

Off-Chain Phragmén

Given the large set of nominators and validators, Phragmén's method is a difficult optimization problem. Polkadot uses off-chain workers to compute the result off-chain and submit a transaction to propose the set of winners. The reason for performing this computation off-chain is to keep a constant block time of six seconds and prevent long block times at the end of each era, when the validator election takes place.

Because certain user actions, like changing nominations, can change the outcome of the Phragmén election, the system forbids calls to these functions for the last quarter of the session before an era change. These functions are not permitted:

- `bondExtra`
- `unbond`
- `withdrawUnbonded`
- `validate`
- `nominate`
- `chill`

- `payoutStakers`
- `rebond`

Council Elections

The Phragmén method is also used in the council election mechanism. When you vote for council members, you can select up to 16 different candidates, and then place a reserved bond as the weight of your vote. Phragmén will run once on every election to determine the top candidates to assume council positions and then again amongst the top candidates to equalize the weight of the votes behind them as much as possible.

What does it mean for node operators?

Phragmén is something that will run in the background and requires no extra effort from you. However, it is good to understand how it works since it means that not all the stake you've been nominated will end up on your validator after an election. Nominators are likely to nominate a few different validators that they trust to do a good job operating their nodes.

You can use the offline-phragmén script for predicting the outcome of a validator election ahead of a new election.

Understanding Phragmén

This section explains the sequential Phragmén method in-depth and walks through examples.

Basic Phragmén

Rationale

In order to understand the Weighted Phragmén method, we must first understand the basic Phragmén method. There must be some group of candidates, a group of seats they are vying for (which is less than the size of the group of candidates), and some group of voters. The voters can cast an approval vote - that is, they can signal approval for any subset of the candidates.

The subset should be a minimum size of one (i.e., one cannot vote for no candidates) and a maximum size of one less than the number of candidates (i.e., one cannot vote for all candidates). Users are allowed to vote for all or no candidates, but this will not have an effect on the final result, and so votes of this nature are meaningless.

Note that in this example, all voters are assumed to have equal say (that is, their vote does not count more or less than any other votes). The weighted case will be considered later. However, weighting can be "simulated" by having multiple voters vote for the same slate of candidates. For instance, five people voting for a particular candidate is mathematically the same as a single person with weight `5` voting for that candidate.

The particular algorithm we call here the "Basic Phragmén" was first described by Brill *et al.* in their paper "Phragmén's Voting Methods and Justified Representation".

Algorithm

The Phragmén method will iterate, selecting one seat at a time, according to the following rules:

1. Candidates submit their ballots, marking which candidates they approve. Ballots will not be modified after submission.
2. An initial load of 0 is set for each ballot.
3. The candidate who wins the next available seat is the one where the ballots of their supporters would have the *least average (mean) cost* if that candidate wins.
4. The *n* ballots that approved that winning candidate get *1/n* added to their load.
5. The load of all ballots that supported the winner of this round are averaged out so that they are equal.
6. If there are any more seats, go back to step 3. Otherwise, the selection ends.

Example

Let's walk through an example with four candidates vying for three seats, and five voters.

```
Open Seats: 3

Candidates:   A B C D  L0
-------------------------
Voter V1:       X      0
Voter V2:         X X  0
Voter V3:       X   X  0
Voter V4:     X X      0
Voter V5:       X X X  0
```

In this example, we can see that voter `V1` approves only of candidate `B`, voter `V2` approves of candidates `C` and `D`, etc. Voters can approve any number of candidates between 1 and `number_of_candidates - 1`. An initial "load" of `0` is set for each ballot (`L0` = load after round `0`, i.e., the "round" before the first round). We shall see shortly how this load is updated and used to select candidates.

We will now run through an iterative algorithm, with each iteration corresponding to one "seat". Since there are three seats, we will walk through three rounds.

For the first round, the winner is simply going to be the candidate with the most votes. Since all loads are equal, the lowest average load will be the candidate with the highest n, since `1/n` will get smaller as `n` increases. For this first example round, for instance, candidate `A` had only one ballot vote for them. Thus, the average load for candidate A is `1/1`, or 1. Candidate C has two ballots approving of them, so the average load is `1/2`. Candidate B has the lowest average load, at `1/4` and they get the first seat. Ballots loads are now averaged out, although for the first iteration, this will not have any effect.

```
Filled seats: 1 (B)
Open Seats: 2

Candidates:   A B C D  L0 L1
----------------------------
Voter V1:       X      0  1/4
Voter V2:         X X  0  0
Voter V3:       X   X  0  1/4
Voter V4:     X X      0  1/4
Voter V5:       X X X  0  1/4
```

We are now down to candidates `A`, `C`, and `D` for two open seats. There is only one voter (`V4`) for `A`, with load `1/4`. `C` has two voters, `V2` and `V5`, with loads of `0` and `1/4`. `D` has three voters approving of them, `V2`, `V3`, and `V5`, with loads of `0`, `1/4`, and `1/4`, respectively.

If Candidate `A` wins, the average load would be `(1/4 + 1/1) / 1`, or `5/4`. If candidate `C` wins, the average load would be `((0 + 1/2) + (1/4 + 1/2)) / 2`, or `5/8`. If candidate `D` wins, the average load would be `((0 + 1/3) + (1/4 + 1/3) + (1/4 + 1/3)) / 3`, or `1/2`. Since `1/2` is the lowest average load, candidate D wins the second round.

Now everybody who voted for Candidate `D` has their load set to the average, `1/2` of all the loads.

```
Filled seats: 2 (B, D)
Open Seats: 1

Candidates:   A B C D  L0 L1  L2
---------------------------------
Voter V1:       X      0  1/4 1/4
Voter V2:         X X  0  0   1/2
Voter V3:       X   X  0  1/4 1/2
Voter V4:     X X      0  1/4 1/4
Voter V5:       X X X  0  1/4 1/2
```

There is now one seat open and two candidates, `A` and `C`. Voter `V4` is the only one voting for `A`, so if `A` wins then the average load would be `(1/4 + 1/1) / 1`, or `5/4`. Voters `V2` and `V5` (both with load `1/2`) support `C`, so if `C` wins the average load would be `((1/2 + 1/2) + (1/2 + 1/2)) / 2`, or `1`. Since the average load would be lower with `C`, `C` wins the final seat.

```
Filled seats: 3 (B, D, C)
Open Seats: 0

Candidates:   A B C D  L0 L1  L2  L3
-------------------------------------
Voter V1:       X      0  1/4 1/4 1/4
Voter V2:         X X  0  0   1/2 1
Voter V3:       X   X  0  1/4 1/2 1/2
Voter V4:     X X      0  1/4 1/4 1/4
Voter V5:       X X X  0  1/4 1/2 1
```

An interesting characteristic of this calculation is that the total load of all voters will always equal the number of seats filled in that round. In the zeroth round, load starts at `0` and there are no seats filled. After the first round, the total of all loads is `1`, after the second round it is `2`, etc.

Weighted Phragmén

Rationale

While this method works well if all voters have equal weight, this is not the case in Polkadot. Elections for both validators and candidates for the Polkadot Council are weighted by the number of tokens held by the voters. This makes elections more similar to a corporate shareholder election than a traditional political election, where some members have more pull than others. Someone with a single token will have much less

voting power than someone with 100. Although this may seem anti-democratic, in a pseudonymous system, it is trivial for someone with 100 tokens to create 100 different accounts and spread their wealth to all of their pseudonyms.

Therefore, not only do we want to allow voters to have their preferences expressed in the result, but do so while keeping as equal a distribution of their stake as possible and express the wishes of minorities as much as is possible. The Weighted Phragmén method allows us to reach these goals.

Algorithm

Weighted Phragmén is similar to Basic Phragmén in that it selects candidates sequentially, one per round, until the maximum number of candidates are elected. However, it has additional features to also allocate weight (stake) behind the candidates.

NOTE: in terms of validator selection, for the following algorithm, you can think of "voters" as "nominators" and "candidates" as "validators".

1. Candidates are elected, one per round, and added to the set of successful candidates (they have won a "seat"). This aspect of the algorithm is very similar to the "basic Phragmén" algorithm described above.
2. However, as candidates are elected, a weighted mapping is built, defining the weights of each selection of a validator by each nominator.

In more depth, the algorithm operates like so:

1. Create a list of all voters, their total amount of stake, and which validators they support.
2. Generate an initial edge-weighted graph mapping from voters to candidates, where each edge weight is the total *potential* weight (stake) given by that voter. The sum of all potential weight for a given candidate is called their *approval stake*.
3. Now we start electing candidates. For the list of all candidates who have not been elected, get their score, which is equal to `1 / approval_stake`.
4. For each voter, update the score of each candidate they support by adding their total budget (stake) multiplied by the load of the voter and then dividing by that candidate's approval stake `(voter_budget * voter_load / candidate_approval_stake)`.
5. Determine the candidate with the lowest score and elect that candidate. Remove the elected candidate from the pool of potential candidates.
6. The load for each edge connecting to the winning candidate is updated, with the edge load set to the score of the candidate minus the voter's load, and the voter's load then set to the candidate's score.
7. If there are more candidates to elect, go to Step 3. Otherwise, continue to step 8.
8. Now the stake is distributed amongst each nominator who backed at least one elected candidate. The backing stake for each candidate is calculated by taking the budget of the voter and multiplying by the edge load then dividing by the candidate load (`voter_budget * edge_load / candidate_load`).

Example

Note: All numbers in this example are rounded off to three decimal places.

In the following example, there are five voters and five candidates vying for three potential seats. Each voter `V1 – V5` has an amount of stake equal to their number (e.g., `V1` has stake of 1, `V2` has stake of 2, etc.). Every voter is also going to have a *load,* which initially starts at `0`.

```
Filled seats: 0
Open Seats: 3

Candidates:    A B C D E  L0
-----------------------------
Voter V1 (1):  X X        0
Voter V2 (2):  X X        0
Voter V3 (3):  X          0
Voter V4 (4):    X X X    0
Voter V5 (5):  X     X    0
```

Let us now calculate the approval stake of each of the candidates. Recall that this is merely the amount of all support for that candidate by all voters.

```
Candidate A: 1 + 2 + 3 + 5 = 11
Candidate B: 1 + 2 + 4 = 7
Candidate C: 4 = 4
Candidate D: 4 + 5 = 9
Candidate E: 0
```

The first step is easy - candidate `E` has 0 approval stake and can be ignored from here on out. They will never be elected.

We can now calculate the initial scores of the candidates, which is `1 / approval_stake`:

```
Candidate A: 1 / 11 = 0.091
Candidate B: 1 / 7 = 0.143
Candidate C: 1 / 4 = 0.25
Candidate D: 1 / 9 = 0.111
Candidate E: N/A
```

For every edge, we are going to calculate the score, which is current score plus the total budget * the load of the voter divided by the approval stake of the candidate. However, since the load of every voter starts at 0, and anything multiplied by 0 is 0, any addition will be `0 / x`, or 0. This means that this step can be safely ignored for the initial round.

Thus, the best (lowest) score for Round 0 is Candidate A, with a score of `0.091`.

```
Candidates:    A B C D E  L0 L1
-----------------------------------
Voter V1 (1):  X X        0  0.091
Voter V2 (2):  X X        0  0.091
Voter V3 (3):  X          0  0.091
Voter V4 (4):    X X X    0  0
Voter V5 (5):  X     X    0  0.091
```

```
Filled seats: 1 (A)
Open Seats: 2

Candidates:    A B C D E  L0
-----------------------------
Voter V1 (1):  X X        0
Voter V2 (2):  X X        0
Voter V3 (3):  X          0
Voter V4 (4):    X X X    0
Voter V5 (5):  X     X    0
```

Candidate `A` is now safe; there is no way that they will lose their seat. Before moving on to the next round, we need to update the scores on the edges of our graph for any candidates who have not yet been elected.

We elided this detail in the previous round, since it made no difference to the final scores, but we should go into depth here to see how scores are updated. We first must calculate the new loads of the voters, and then calculate the new scores of the candidates.

Any voter who had one of their choices for candidate fill the seat in this round (i.e., voters `V1`, `V2`, `V3`, and `V5`, who all voted for `A`) will have their load increased. This load increase will blunt the impact of their vote in future rounds, and the edge (which will be used in determining stake allocation later) is set to the score of the elected candidate minus the *current* voter load.

```
edge_load = elected_candidate_score − voter_load
voter_load = elected_candidate_score
```

In this instance, the score of the elected candidate is `0.091` and the voter loads are all `0`. So for each voter who voted for `A`, we will calculate a new edge load `Voter` -> `A` of:

```
Edge load: 0.091 − 0 = 0.091
```

and a new voter load of:

```
Voter load: 0.091
```

As a reminder, here are the current scores. Loads of the voters are all `0`.

```
Candidate B : 0.143
Candidate C : 0.25
Candidate D : 0.111
```

Now, we go through the weighted graph and update the score of the candidate and the load of the edge, using the algorithm:

```
candidate_score = candidate_score + ((voter_budget ∗ voter_load) /
candidate_approval_stake)
```

Without walking through each step, this gives us the following modifications to the scores of the different candidates.

```
V1 updates B to 0.156
V2 updates B to 0.182
V4 updates B to 0.182
V4 updates C to 0.25
V4 updates D to 0.111
V5 updates D to 0.162
```

After scores are updated, the final scores for the candidates for this round are:

```
Candidate B: 0.182
Candidate C: 0.25
Candidate D: 0.162
```

`D`, with the lowest score, is elected. You will note that even though candidate `B` had more voters supporting them, candidate `D` won the election due to their lower score. This is directly due to the fact that they had the lowest score, of course, but the root reason behind them having a lower score was both the greater amount of stake behind them and that voters who did not get one of their choices in an earlier round (in this example, voter V4) correspond to a higher likelihood of a candidate being elected.

We then update the loads for the voters and edges as specified above for any voters who voted for candidate `D` (viz., `V4` and `V5`) using the same formula as above.

```
Filled seats: 2 (A, D)
Open Seats: 1

Candidates:    A B C D E  L0 L1    L2
-----------------------------------
Voter V1 (1):  X X        0  0.091 0.091
Voter V2 (2):  X X        0  0.091 0.091
Voter V3 (3):  X          0  0.091 0.091
Voter V4 (4):    X X X    0  0     0.162
Voter V5 (5):  X     X    0  0.091 0.162
```

Following a similar process for Round 2, we start with initial candidate scores of:

```
Candidate B : 0.143
Candidate C : 0.25
```

We can then update the scores of the remaining two candidates according to the algorithm described above.

```
V1 updates B to 0.156
V2 updates B to 0.182
V4 updates B to 0.274
V4 updates C to 0.412
```

With the lowest score of `0.274`, Candidate `B` claims the last open seat. Candidates `A`, `D`, and `B` have been elected, and candidates `C` and `E` are not.

Before moving on, we must perform a final load adjustment for the voters and the graph.

```
Filled seats: 3 (A, D, B)
Open Seats: 0

Candidates:    A B C D E  L0 L1    L2    L3
-----------------------------------------
Voter V1 (1):  X X        0  0.091 0.091 0.274
Voter V2 (2):  X X        0  0.091 0.091 0.274
Voter V3 (3):  X          0  0.091 0.091 0.091
Voter V4 (4):    X X X    0  0     0.162 0.274
Voter V5 (5):  X     X    0  0.091 0.162 0.162
```

Now we have to determine how much stake every voter should allocate to each candidate. This is done by taking the load of the each edge and dividing it by the voter load, then multiplying by the total budget of the voter.

In this example, the weighted graph ended up looking like this:

```
Nominator: V1
    Edge to A load= 0.091
    Edge to B load= 0.183
Nominator: V2
    Edge to A load= 0.091
    Edge to B load= 0.183
Nominator: V3
    Edge to A load= 0.091
Nominator: V4
    Edge to B load= 0.113
    Edge to D load= 0.162
Nominator: V5
    Edge to A load= 0.091
    Edge to D load= 0.071
```

For instance, the budget of `V1` is `1`, the edge load to `A` is `0.091`, and the voter load is `0.274`. Using our equation:

```
backing_stake (A) = voter_budget * edge_load / voter_load
```

We can fill these variables in with:

```
backing_stake (A) = 1 * 0.091 / 0.274 = 0.332
```

For `V1` backing stake of `B`, you can simply replace the edge load value and re-calculate.

```
backing_stake (B) = 1 * 0.183 / 0.274 = 0.668
```

Note that the total amount of all backing stake for a given voter will equal the total budget of the voter, unless that voter had no candidates elected, in which case it will be 0.

The final results are:

```
A is elected with stake 6.807.
D is elected with stake 4.545.
B is elected with stake 3.647.

V1 supports: A with stake: 0.332 and B with stake: 0.668.
V2 supports: A with stake: 0.663 and B with stake: 1.337.
V3 supports: A with stake: 3.0.
V4 supports: B with stake: 1.642 and D with stake: 2.358.
V5 supports: A with stake: 2.813 and D with stake: 2.187.
```

You will notice that the total amount of stake for candidates `A`, `D`, and `B` equals (aside from rounding errors) the total amount of stake of all the voters (`1 + 2 + 3 + 4 + 5 = 15`). This is because each voter had at least one of their candidates fill a seat. Any voter whose had none of their candidates selected will also not have any stake in any of the elected candidates.

Optimizations

The results for nominating validators are further optimized for several purposes:

1. To reduce the number of edges, i.e. to minimize the number of validators any nominator selects
2. To ensure, as much as possible, an even distribution of stake among the validators
3. Reduce the amount of block computation time

High-Level Description

After running the weighted Phragmén algorithm, a process is run that redistributes the vote amongst the elected set. This process will never add or remove an elected candidate from the set. Instead, it reduces the variance in the list of backing stake from the voters to the elected candidates. Perfect equalization is not always possible, but the algorithm attempts to equalize as much as possible. It then runs an edge-reducing algorithm to minimize the number of validators per nominator, ideally giving every nominator a single validator to nominate per era.

To minimize block computation time, the staking process is run as an off-chain worker. In order to give time for this off-chain worker to run, staking commands (bond, nominate, etc.) are not allowed in the last quarter of each era.

These optimizations will not be covered in-depth on this page. For more details, you can view the Rust implementation of elections in Substrate, the Rust implementation of staking in Substrate, or the `seqPhragménwithpostprocessing` method in the Python reference implementation. If you would like to dive even more deeply, you can review the W3F Research Page on Sequential Phragmén Method.

Rationale for Minimizing the Number of Validators Per Nominator

Paying out rewards for staking from every validator to all of their nominators can cost a non-trivial amount of chain resources (in terms of space on chain and resources to compute). Assume a system with 200 validators and 1000 nominators, where each of the nominators has nominated 10 different validators. Payout would thus require `1_000 * 10`, or 10_000 transactions. In an ideal scenario, if every nominator selects a single validator, only 1_000 transactions would need to take place - an order of magnitude fewer. Empirically, network slowdown at the beginning of an era has occurred due to the large number of individual payouts by validators to nominators. In extreme cases, this could be an attack vector on the system, where nominators nominate many different validators with small amounts of stake in order to slow the system at the next era change.

While this would reduce network and on-chain load, being able to select only a single validator incurs some diversification costs. If the single validator that a nominator has nominated goes offline or acts maliciously, then the nominator incurs a risk of a significant amount of slashing. Nominators are thus allowed to nominate up to 16 different validators. However, after the weighted edge-reducing algorithm is run, the number of validators per nominator is minimized. Nominators are likely to see themselves nominating a single active validator for an era.

At each era change, as the algorithm runs again, nominators are likely to have a different validator than they had before (assuming a significant number of selected validators). Therefore, nominators can diversify against incompetent or corrupt validators causing slashing on their accounts, even if they only nominate a single validator per era.

Rationale for Maintaining an Even Distribution of Stake

Another issue is that we want to ensure that as equal a distribution of votes as possible amongst the elected validators or council members. This helps us increase the security of the system by ensuring that the minimum amount of tokens in order to join the active validator set or council is as high as possible. For example, assume a result of five validators being elected, where validators have the following stake: `{1_000, 20, 10, 10, 10}`, for a total stake of 1_050. In this case, a potential attacker could join the active validator set with only 11 tokens, and could obtain a majority of validators with only 33 tokens (since the attacker only has to have enough stake to "kick out" the three lowest validators).

In contrast, imagine a different result with the same amount of total stake, but with that stake perfectly equally distributed: `{210, 210, 210, 210, 210}`. With the same amount of stake, an attacker would need to stake 633 tokens in order to get a majority of validators, a much more expensive proposition. Although obtaining an equal distribution is unlikely, the more equal the distribution, the higher the threshold - and thus the higher the expense - for attackers to gain entry to the set.

Rationale for Reducing Block Computing Time

Running the Phragmén algorithm is time-consuming, and often cannot be completed within the time limits of production of a single block. Waiting for calculation to complete would jeopardize the constant block production time of the network. Therefore, as much computation as possible is moved to an offchain worker, which validators can work on the problem without impacting block production time. By restricting the ability of users to make any modifications in the last 25% of an era, and running the selection of validators by nominators as an offchain process, validators have a significant amount of time to calculate the new active validator set and allocate the nominators in an optimal manner.

There are several further restrictions put in place to limit the complexity of the election and payout. As already mentioned, any given nominator can only select up to 16 validators to nominate. Conversely, a single validator can have only 256 nominators. A drawback to this is that it is possible, if the number of nominators is very high or the number of validators is very low, that all available validators may be "oversubscribed" and unable to accept more nominations. In this case, one may need a larger amount of stake to participate in staking, since nominations are priority-ranked in terms of amount of stake.

Phragmms (fka Balphragmms)

`Phragmms`, formerly known as `Balphragmms`, is a new election rule inspired by Phragmén and developed in-house for Polkadot. In general, election rules on blockchains is an active topic of research. This is due to the conflicting requirements for election rules and blockchains: elections are computationally expensive, but blockchains are computationally limited. Thus, this work constitutes state of the art in terms of optimization.

Proportional representation is a very important property for a decentralized network to have in order to maintain a sufficient level of decentralization. While this is already provided by the currently implemented `seqPhragmen`, this new election rule provides the advantage of the added security guarantee described below. As far as we can tell, at the time of writing, Polkadot and Kusama are the only blockchain networks that implement an election rule that guarantees proportional representation.

The security of a distributed and decentralized system such as Polkadot is directly related to the goal of avoiding *overrepresentation* of any minority. This is a stark contrast to traditional approaches to proportional representation axioms, which typically only seek to

avoid underrepresentation.

Maximin Support Objective and PJR

This new election rule aims to achieve a constant-factor approximation guarantee for the *maximin support objective* and the closely related *proportional justified representation* (PJR) property.

The maximin support objective is based on maximizing the support of the least-supported elected candidate, or in the case of Polkadot and Kusama, maximizing the least amount of stake backing amongst elected validators. This security-based objective translates to a security guarantee for NPoS and makes it difficult for an adversarial whale's validator nodes to be elected. The `Phragmms` rule, and the guarantees it provides in terms of security and proportionality, have been formalized in a peer-reviewed paper).

The PJR property considers the proportionality of the voter's decision power. The property states that a group of voters with cohesive candidate preferences and a large enough aggregate voting strength deserve to have a number of representatives proportional to the group's vote strength.

Comparing Sequential Phragmén, MMS, and Phragmms

Sequential Phragmén (`seqPhragmen`) and `MMS` are two efficient election rules that both achieve PJR.

Currently, Polkadot employs the `seqPhragmen` method for validator and council elections. Although `seqPhramen` has a very fast runtime, it does not provide constant-factor approximation for the maximin support problem. This is due to `seqPhramen` only performing an *approximate* rebalancing of the distribution of stake.

In contrast, `MMS` is another standard greedy algorithm that simultaneously achieves the PJR property and provides a constant factor approximation for maximin support, although with a considerably slower runtime. This is because for a given partial solution, `MMS` computes a balanced edge weight vector for each possible augmented committee when a new candidate is added, which is computationally expensive.

We introduce a new heuristic inspired by `seqPhragmen`, `PhragMMS`, which maintains a comparable runtime to `seqPhragmen`, offers a constant-factor approximation guarantee for the maximin support objective, and satisfies PJR. This is the fastest known algorithm to achieve a constant-factor guarantee for maximin support.

The New Election Rule: Phragmms

`Phragmms` is an iterative greedy algorithm that starts with an empty committee and alternates between the `Phragmms` heuristic for inserting a new candidate and *rebalancing* by replacing the weight vector with a balanced one. The main differentiator between `Phragmms` and `seqPhragmen` is that the latter only perform an approximate rebalancing. Details can be found in Balanced Stake Distribution.

The computation is executed by off-chain workers privately and separately from block production, and the validators only need to submit and verify the solutions on-chain. Relative to a committee A, the score of an unelected candidate c is an easy-to-compute rough estimate of what would be the size of the least stake backing if we added c to committee A. Observing on-chain, only one solution needs to be tracked at any given time, and a block producer can submit a new solution in the block only if the block passes the verification test, consisting of checking:

1. Feasibility,
2. Balancedness, and

3. Local Optimality - The least stake backing of *A* is higher than the highest score among unelected candidates

If the tentative solution passes the tests, then it replaces the current solution as the tentative winner. The official winning solution is declared at the end of the election window.

A powerful feature of this algorithm is the fact that both its approximation guarantee for maximin support and the above checks passing can be efficiently verified in linear time. This allows for a more scalable solution for secure and proportional committee elections. While `seqPhragmen` also has a notion of score for unelected candidates, `Phragmms` can be seen as a natural complication of the `seqPhragmen` algorithm, where `Phragmms` always grants higher score values to candidates and thus inserts them with higher support values.

To summarize, the main differences between the two rules are:

- In `seqPhragmen`, lower scores are better, whereas in `Phragmms`, higher scores are better.
- Inspired by `seqPhragmen`, the scoring system of `Phragmms` can be considered to be more intuitive and does a better job at estimating the value of adding a candidate to the current solution, and hence leads to a better candidate-selection heuristic.
- Unlike `seqPhragmen`, in `Phragmms`, the edge weight vector *w* is completely rebalanced after each iteration of the algorithm.

The `Phragmms` election rule is currently being implemented on Polkadot. Once completed, it will become one of the most sophisticated election rules implemented on a blockchain. For the first time, this election rule will provide both fair representation (PJR) and security (constant-factor approximation for the maximin support objection) to a blockchain network.

Algorithm

The `Phragmms` algorithm iterates through the available seats, starting with an empty committee of size *k*:

1. Initialize an empty committee *A* and zero edge weight vector $w = 0$.
2. Repeat *k* times:
    - Find the unelected candidate with highest score and add it to committee *A*.
    - Re-balance the weight vector *w* for the new committee *A*.
3. Return *A* and *w*.

External Resources

- Phragmms - W3F research paper that expands on the sequential Phragmén method.
- W3F Research Page on NPoS - An overview of Nominated Proof of Stake as its applied to Polkadot.
- Python Reference Implementations - Python implementations of Simple and Complicated Phragmén methods.
- Substrate Implementation - Rust implementation used in Substrate.
- Phragmén's and Thiele's Election Methods - 95-page paper explaining Phragmén's election methods in detail.
- Phragmén's Voting Methods and Justified Representation - This paper by Brill *et al.* is

the source for the simple Phragmén method, along with proofs about its properties.
- Offline Phragmén - Script to generate the Phragmén validator election outcome before the start of an era.

Edit this page

*Last updated on **10/18/2021** by **Dan Shields***

Simple Payouts

Polkadot and Kusama make stakers claim their rewards for past eras by submitting a transaction. This naturally leads to spreading out reward distribution, as people make transactions at disparate times, rather than updating the accounts of all stakers in a single block.

Even if everyone submitted a reward claim at the same time, the fact that they are individual transactions would allow the block construction algorithm to process only a limited number per block and ensure that the network maintains a constant block time. If all rewards were sent out in one block, this could cause serious issues with the stability of the network.

Simple payouts require one transaction per validator, per era, to claim rewards. The reason Polkadot requires this is to avoid an attack where someone has several thousand accounts nominating a single validator. The major cost in reward distribution is mutating the accounts in storage, and Polkadot cannot pay out several thousand accounts in a single transaction.

Claiming Rewards

Polkadot stores the last 84 eras of reward information (e.g. maps of era number to validator points, staking rewards, nomination exposure, etc.). Rewards will not be claimable more than 84 eras after they were earned. This means that all rewards must be claimed within a maximum of 84 eras, although under certain circumstances (described below) this may be as low as 28 eras.

If a validator kills their stash, any remaining rewards will no longer be claimable. Before doing this, however, they would need to first stop validating and then unbond the funds in their stash, which takes 28 eras. If a validator were to immediately chill and start unbonding after rewards are calculated, and nobody issued a payout for that era from that validator in the next 28 eras, the reward would no longer be claimable.

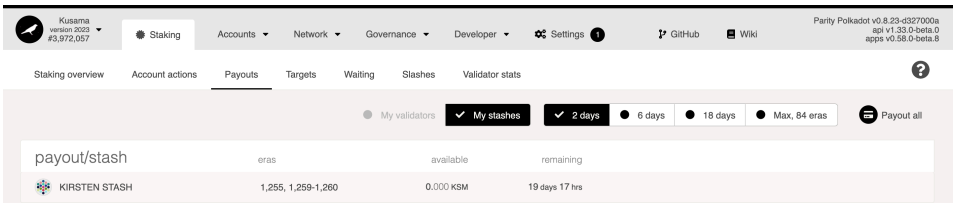
> In order to be absolutely sure that staking rewards can be claimed, users should trigger a payout before 28 eras have passed.

Anyone can trigger a payout for any validator, as long as they are willing to pay the transaction fee. Someone must submit a transaction with a validator ID and an era index. Polkadot will automatically calculate that validator's reward, find the top 256 nominators for that era, and distribute the rewards pro rata.

> NOTE: The Staking system only applies the highest 256 nominations to each validator to reduce the complexity of the staking set.

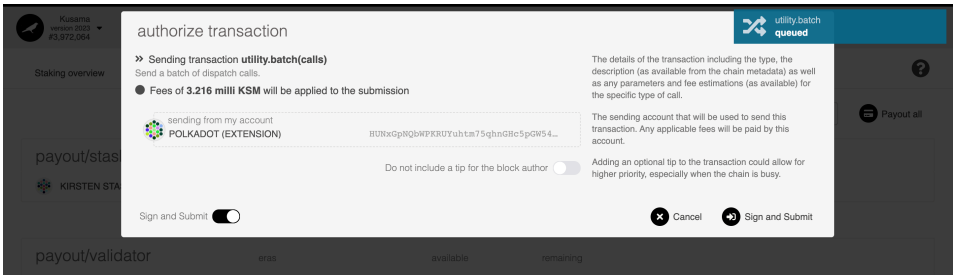
These details are handled for you automatically if you use the Polkadot-JS UI, which also allows you to submit batches of eras at once.

To claim rewards on Polkadot-JS UI, you will need to be in the "Payouts" tab underneath "Staking", which will list all the pending payouts for your stashes.



To then claim your reward, select the "Payout all" button. This will prompt you to select your stash accounts for payout.

Once you are done with payout, another screen will appear asking for you to sign and submit the transaction.



F.A.Q. and Cautionary Notes

1. Rewards expire after 84 eras. On Polkadot, that's about 84 days. On Kusama, it is approximately 21 days. Validators should claim all pending rewards before killing their stash in the event the validator decides to `chill` -> `unbonds all` -> `withdraws unbonded`. Nominators will not miss out on rewards if they claim the pending rewards for a validator within 28 days. Essentially, the deadline to ensure you get staking rewards is 28 eras. If the validator verifies its intent and does not unbond and withdraw, the 84 era timeline holds.
2. Claiming rewards (or neglecting to claim rewards) does not affect nominations in any way. Nominations will persist after claiming rewards or after the rewards expire.
3. Rewards are not minted until they are claimed. Therefore, if your reward destination is "stash, increasing amount at stake", then your staked amount does not reflect your rewards until you claim them. If you want to maximize compounding, then you will need to claim often or nominate validators which regularly claim for you.
4. Staking operations at the end of an era are closed to allow the off-chain validator election to take place. See Off-chain Phragmén for more information.

Edit this page

*Last updated on **11/16/2021** by **Alexander Gryaznov***

Cryptography Explainer

This is a high-level overview of the cryptography used in Polkadot. It assumes that you have some knowledge about cryptographic primitives that are generally used in blockchains such as hashes, elliptic curve cryptography (ECC), and public-private keypairs.

For detailed descriptions on the cryptography used in Polkadot please see the more advanced research wiki.

Hashing Algorithm

The hashing algorithm used in Polkadot is Blake2b. Blake2 is considered to be a very fast cryptographic hash function that is also used in the cryptocurrency Zcash.

Keypairs and Signing

Polkadot uses Schnorrkel/Ristretto x25519 ("sr25519") as its key derivation and signing algorithm.

Sr25519 is based on the same underlying Curve25519 as its EdDSA counterpart, Ed25519. However, it uses Schnorr signatures instead of the EdDSA scheme. Schnorr signatures bring some noticeable benefits over the ECDSA/EdDSA schemes. For one, it is more efficient and still retains the same feature set and security assumptions. Additionally, it allows for native multisignature through signature aggregation.

The names Schnorrkel and Ristretto come from the two Rust libraries that implement this scheme, the Schnorrkel library for Schnorr signatures and the Ristretto library that makes it possible to use cofactor-8 curves like Curve25519.

Edit this page

*Last updated on **9/17/2021** by **Danny Salman***

Polkadot Keys

Public and private keys are an important aspect of most crypto-systems and an essential component that enables blockchains like Polkadot to exist.

Account Keys

Account keys are keys that are meant to control funds. They can be either:

- The vanilla `ed25519` implementation using Schnorr signatures.
- The Schnorrkel/Ristretto `sr25519` variant using Schnorr signatures.
- ECDSA signatures on secp256k1

There are no differences in security between `ed25519` and `sr25519` for simple signatures.

We expect `ed25519` to be much better supported by commercial HSMs for the foreseeable future.

At the same time, `sr25519` makes implementing more complex protocols safer. In particular, `sr25519` comes with safer version of many protocols like HDKD common in the Bitcoin and Ethereum ecosystem.

"Controller" and "Stash" Keys

When we talk about "controller" and "stash" keys, we usually talk about them in the context of running a validator or nominating DOT, but they are useful concepts for all users to know. Both keys are types of account keys. They are distinguished by their intended use, not by an underlying cryptographic difference. All the info mentioned in the parent section applies to these keys. When creating new controller or stash keys, all cryptography supported by account keys are an available option.

The controller key is a semi-online key that will be in the direct control of a user, and used to submit manual extrinsics. For validators or nominators, this means that the controller key will be used to start or stop validating or nominating. Controller keys should hold some DOT to pay for fees, but they should not be used to hold huge amounts or life savings. Since they will be exposed to the internet with relative frequency, they should be treated carefully and occasionally replaced with new ones.

The stash key is a key that will, in most cases, be a cold wallet, existing on a piece of paper in a safe or protected by layers of hardware security. It should rarely, if ever, be exposed to the internet or used to submit extrinsics. The stash key is intended to hold a large amount of funds. It should be thought of as a saving's account at a bank, which ideally is only ever touched in urgent conditions. Or, perhaps a more apt metaphor is to think of it as buried treasure, hidden on some random island and only known by the pirate who originally hid it.

Since the stash key is kept offline, it must be set to have its funds bonded to a particular controller. For non-spending actions, the controller has the funds of the stash behind it. For example, in nominating, staking, or voting, the controller can indicate its preference with the weight of the stash. It will never be able to actually move or claim the funds in the stash key. However, if someone does obtain your controller key, they could use it for slashable behavior, so you should still protect it and change it regularly.

Session Keys

Session keys are hot keys that must be kept online by a validator to perform network operations. Session keys are typically generated in the client, although they don't have to be. They are *not* meant to control funds and should only be used for their intended purpose. They can be changed regularly; your controller only need create a certificate by signing a session public key and broadcast this certificate via an extrinsic.

Polkadot uses four session keys:

- GRANDPA: ed25519
- BABE: sr25519
- I'm Online: sr25519
- Parachain: sr25519

BABE requires keys suitable for use in a Verifiable Random Function as well as for digital signatures. Sr25519 keys have both capabilities and so are used for BABE.

In the future, we plan to use a BLS key for GRANDPA because it allows for more efficient signature aggregation.

FAQ

Why was `ed25519` selected over `secp256k1`?

The original key derivation cryptography that was implemented for Polkadot and Substrate chains was `ed25519`, which is a Schnorr signature algorithm implemented over the Edward's Curve 25519 (so named due to the parameters of the curve equation).

Most cryptocurrencies, including Bitcoin and Ethereum, currently use ECDSA signatures on the secp256k1 curve. This curve is considered much more secure than NIST curves, which have possible backdoors from the NSA. The Curve25519 is considered possibly *even more* secure than this one and allows for easier implementation of Schnorr signatures. A recent patent expiration on it has made it the preferred choice for use in Polkadot.

The choice of using Schnorr signatures over using ECDSA is not so cut and dry. As stated in Jeff Burdges' (a Web3 researcher)original forum post on the topic:

> There is one sacrifice we make by choosing Schnorr signatures over ECDSA signatures for account keys: Both require 64 bytes, but only ECDSA signatures communicate their public key. There are obsolete Schnorr variants that support recovering the public key from a signature, but they break important functionality like hierarchical deterministic key derivation. In consequence, Schnorr signatures often take an extra 32 bytes for the public key.

But ultimately the benefits of using Schnorr signatures outweigh the tradeoffs, and future optimizations may resolve the inefficiencies pointed out in the quote above.

What is `sr25519` and where did it come from?

Some context: The Schnorr signatures over the Twisted Edward's Curve25519 are considered secure, however Ed25519 has not been completely devoid of its bugs. Most notably, Monero and all other CryptoNote currencies were vulnerable to a double spend exploit that could have potentially led to undetected, infinite inflation.

These exploits were due to one peculiarity in Ed25519, which is known as its cofactor of 8. The cofactor of a curve is an esoteric detail that could have dire consequences for the security of implementation of more complex protocols.

Conveniently, Mike Hamburg's Decaf paper provides a possible path forward to solving this potential bug. Decaf is basically a way to take Twisted Edward's Curves cofactor and mathematically change it with little cost to performance and gains to security.

The Decaf paper approach by the Ristretto Group was extended and implemented in Rust to include cofactor-8 curves like the Curve25519 and makes Schnorr signatures over the Edward's curve more secure.

Web3 Foundation has implemented a Schnorr signature library using the more secure Ristretto compression over the Curve25519 in the Schnorrkel repository. Schnorrkel implements related protocols on top of this curve compression such as HDKD, MuSig, and a verifiable random function (VRF). It also includes various minor improvements such as the hashing scheme STROBE that can theoretically process huge amounts of data with only one call across the Wasm boundary.

The implementation of Schnorr signatures that is used in Polkadot and implements the Schnorrkel protocols over the Ristretto compression of the Curve25519 is known as **sr25519**.

Are BLS signatures used in Polkadot?

Not yet, but they will be. BLS signatures allow more efficient signature aggregation. Because GRANDPA validators are usually signing the same thing (e.g. a block), it makes sense to aggregate them, which can allow for other protocol optimizations.

As stated in the BLS library's README,

> Boneh-Lynn-Shacham (BLS) signatures have slow signing, very slow verification, require slow and much less secure pairing friendly curves, and tend towards dangerous malleability. Yet, BLS permits a diverse array of signature aggregation options far beyond any other known signature scheme, which makes BLS a preferred scheme for voting in consensus algorithms and for threshold signatures.

Even though Schnorr signatures allow for signature aggregation, BLS signatures are much more efficient in some fashions. For this reason it will be one of the session keys that will be used by validators on the Polkadot network and critical to the GRANDPA finality gadget.

Resources

- Key discovery attack on BIP32-Ed25519 - Forum post detailing a potential attack on BIP32-Ed25519. A motivation for transition to the sr25519 variant.
- Account signatures and keys in Polkadot - Original forum post by Web3 researcher Jeff Burdges.
- Are Schnorr signatures quantum computer resistant?

Appendix A: On the security of curves

From the introduction of Curve25519 into `libssl`:

```
The reason is the following: During summer of 2013, revelations from ex-
consultant at [the] NSA Edward Snowden gave proof that [the] NSA willingly
inserts backdoors
into software, hardware components and published standards. While it is still
believed that the mathematics behind ECC (Elliptic-curve cryptography) are still
sound and solid,
some people (including Bruce Schneier [SCHNEIER]), showed their lack of
confidence
in NIST-published curves such as nistp256, nistp384, nistp521, for which
constant
parameters (including the generator point) are defined without explanation. It
is also believed that [the] NSA had a word to say in their definition. These
curves
are not the most secure or fastest possible for their key sizes [DJB], and
researchers think it is possible that NSA have ways of cracking NIST curves.
It is also interesting to note that SSH belongs to the list of protocols the NSA
claims to be able to eavesdrop. Having a secure replacement would make passive
attacks much harder if such a backdoor exists.

However an alternative exists in the form of Curve25519. This algorithm has been
proposed in 2006 by DJB [Curve25519]. Its main strengths are its speed, its
constant-time run time (and resistance against side-channel attacks), and its
lack of nebulous hard-coded constants.
```

Edit this page

*Last updated on **10/23/2021** by **Danny Salman***

What is the difference between Polkadot and Kusama?

Although they share many parts of their code, Polkadot and Kusama are independent, standalone networks with different priorities.

Kusama is wild and fast; great for bold experimentation and early-stage deployment. Polkadot is more conservative, prioritizing stability and dependability.

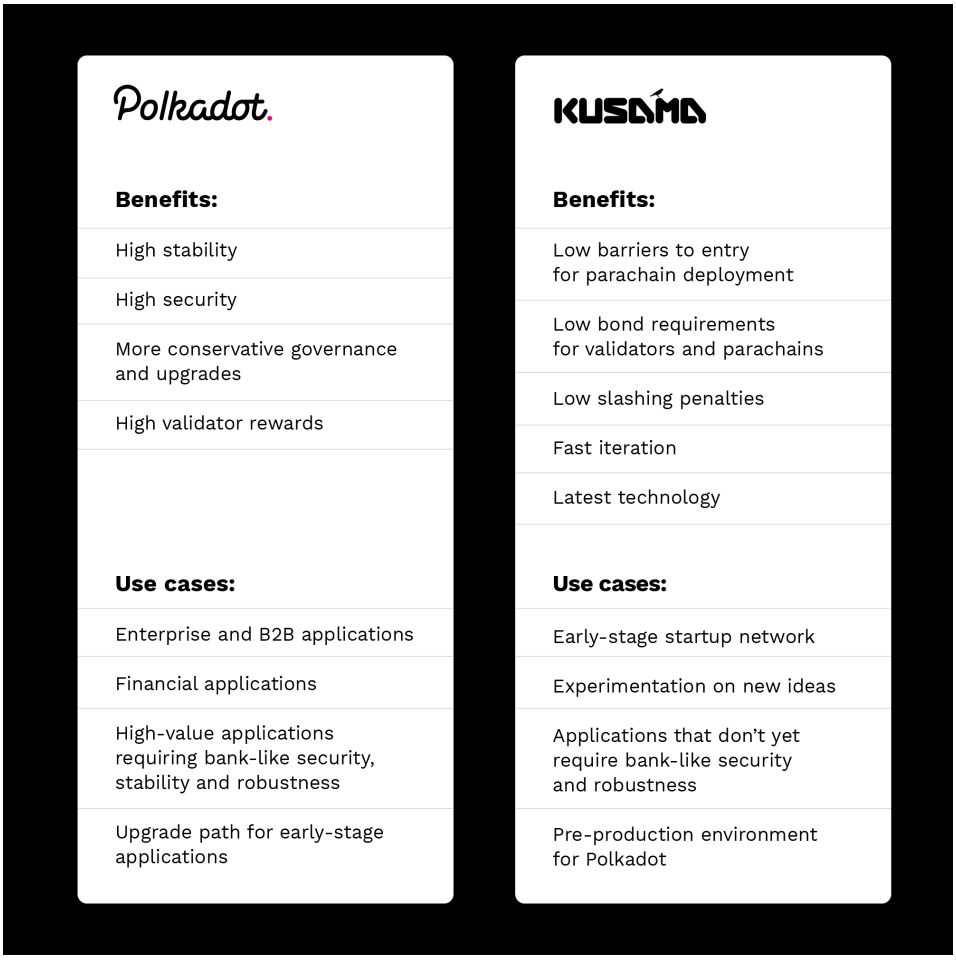
Cousins have their differences after all.

What the two networks have in common

Kusama was released as an early version of the same code to be used in Polkadot, which means they share the same underlying architecture: a multichain, heterogeneously-sharded design based on Nominated Proof of Stake (NPoS). Both networks also share key innovations like on-chain governance, hot-swappable runtimes for forkless, on-chain upgrades, and Cross-Consensus Message Passing (XCM) for interoperability. Governance on both Polkadot and Kusama is designed to be decentralized and permissionless, giving a say in how the network is run to everyone who owns the native token (DOT for Polkadot, KSM for Kusama). Therefore, **over time the networks will evolve independently, converging or diverging according to the decisions of their respective communities.**

Key differences

There are a few important distinctions to be made.



Both networks also have different circulating supplies.

Speed

The first key technical difference between Polkadot and Kusama is that Kusama has modified governance parameters that allow for faster upgrades. Kusama is up to four times faster than Polkadot, with seven days for token holders to vote on referendums followed by an enactment period of eight days, after which the referendum will be enacted on the chain. This means stakeholders need to stay active and vigilant if they want to keep up with all the proposals, referenda, and upgrades, and validators on Kusama often need to update on short notice.

On Polkadot, votes last 28 days followed by an enactment period of 28 days. This does not mean that the Kusama blockchain itself is faster, in the sense of faster block times or transaction throughput (these are the same on both networks), but that there's a shorter amount of time between governance events such as proposing new referenda, voting, and enacting approved upgrades. This allows Kusama to adapt and evolve faster than Polkadot.

Lean setups

Teams wishing to run a parachain need to bond tokens as security. The bonding requirement on Kusama is likely to be lower than on Polkadot.

Use cases

Polkadot is and always will be the primary network for the deployment of enterprise-level applications and those that entail high-value transactions requiring bank-level security and stability.

The initial use case for Kusama is as a pre-production environment, a “canary network”. For the average developer, this seems like it could be a testnet, what is the difference? What does *canary* even mean?

Canary is a type of bird: back in the day, coal miners would put canaries into coal mines as a way to measure the amount of toxic gases that were present. Similarly, canary testing is a way to validate software by releasing software to a limited number of users, or perhaps, an isolated environment - without hurting any birds.

Releases made onto Kusama can be thought of as Canary Releases. These releases are usually staged. In Kusama's early days, the network won't just be used for parachain candidates to innovate and test changes, but a proof of concept for Polkadot's sharded model.

In a typical blockchain development pipeline, Kusama would sit in between a "testnet" and a "mainnet"

```
Testnet –> Kusama –> Polkadot
```

As you can imagine, building on Kusama first allows teams to test things out in a live, fully decentralized, and community-controlled network with real-world conditions and lower stakes in the event of problems or bugs than on Polkadot.

Many projects will maintain parachains on both networks, experimenting and testing new technologies and features on Kusama before deploying them to Polkadot. Some teams will decide just to stay on Kusama, which is likely to be a place where we see some exciting experimentation with new technologies going forward. Projects that require high-throughput but don’t necessarily require bank-like security, such as some gaming, social networking, and content distribution applications, are particularly good candidates for this use case.

Kusama may also prove to be the perfect environment for ambitious experiments with new ideas and innovations in areas like governance, incentives, monetary policy, and DAOs (decentralized autonomous organizations). Future upgrades to the Polkadot runtime will also likely be deployed to Kusama before Polkadot mainnet. This way, not only will we be able to see how these new technologies and features will perform under real-world conditions before bringing them to Polkadot, but teams who have deployed to both networks will also get an advanced look at how their own technology will perform under those upgrades.

Going forward

Ultimately, Kusama and Polkadot will live on as independent, standalone networks with their own communities, their own governance, and their own complementary use cases, though they will continue to maintain a close relationship, with many teams likely deploying applications to both networks. In the future, we’re also likely to see Kusama bridged to Polkadot for cross-network interoperability. Web3 Foundation remains committed to both networks going forward, providing crucial support and guidance to teams building for the ecosystem.

Explore more

- About Kusama
- The Kusama Wiki
- Kusama on Polkadot-JS Apps

Edit this page

*Last updated on **10/23/2021** by **Danny Salman***

Polkadot and Ethereum 2.0

Polkadot and Ethereum 2.0 are both sharded blockchain protocols. As such, they provide scalability by executing transactions in separate shards and provide a protocol to send messages between shards.

Model

The shards in Ethereum 2.0 all have the same state transition function (STF), as in the rules governing how the blockchain can change state with each block. This STF provides an interface for smart contract execution. Contracts exist on a single shard and can send asynchronous messages between shards.

Likewise, in Polkadot, each shard hosts core logic, the shards are executed in parallel, and Polkadot can send cross-shard asynchronous messages. However, each Polkadot shard (in Polkadot terminology, "parachain") has a unique STF. Applications can exist either within a single shard or across shards by composing logic. Polkadot uses WebAssembly (Wasm) as a "meta-protocol". A shard's STF can be abstract as long as the validators on Polkadot can execute it within a Wasm environment. Polkadot will support smart contracts through parachains. To offer some perspective, on Ethereum, smart contracts can call each other synchronously in the same shard and asynchronously between shards. On Polkadot, smart contracts will be able to call each other synchronously in the same parachain and asynchronously across parachains.

Architecture

Ethereum 2.0

Ethereum 2.0's main chain is called the Beacon Chain. The primary load on the Beacon Chain is attestations, which are votes on the availability of shard data and Beacon Chain validity. Each shard in Ethereum 2 is simply a blockchain with the Ethereum Wasm (eWasm) interface.

Ethereum 2.0 launched phase 0 of a multi-phase rollout in December 2020, operating in parallel to the legacy Ethereum 1.0 chain:

- **Phase 0** provisioned the Beacon Chain, accepting deposits from validators and implementing proof-of-stake consensus, eventually among many shards.
- **Phase 1** launches 64 shards as simple chains, to test the Beacon Chain's finality. Each shard submits "crosslinks" to the Beacon Chain, which contains the information to finalize shard data.
- **Phase 1.5** integrates Eth 1 as a shard to finalize the proof-of-work chain's blocks.
- **Phase 2** implements the eWasm interface, phasing out proof-of-work, finally making the system usable to end-users. [1]

After the launch of the Beacon Chain in phase 0, the roadmap was altered to prioritize the transition of the legacy Ethereum 1.0 chain from Proof-of-Work to Ethereum 2.0's Proof-of-Stake consensus, preceding the rollout of shards on the network. [2]

The network will also have "side chains" to interact with chains that are not under the finality protocol of Ethereum 2.0.

Polkadot

Like Ethereum 2.0, Polkadot also has a main chain, called the Relay Chain, with several shards, called parachains. Parachains are not restricted to a single interface like eWasm. Instead, they can define their own logic and interface, as long as they provide their STF to the Relay Chain validators so that they can execute it.

Polkadot, now live as a Relay Chain, only plans to launch the ability to validate up to 20 shards per block, gradually scaling up to 100 shards per block. Besides parachains, which are scheduled for execution every block, Polkadot also has parathreads, which are scheduled on a dynamic basis. This allows chains to share the sharded slots, much like multiple small airlines might share a gate at an airport.

In order to interact with chains that want to use their own finalization process (e.g. Bitcoin), Polkadot has bridge parachains that offer two-way compatibility.

Consensus

Both Ethereum 2.0 and Polkadot use hybrid consensus models where block production and finality each have their own protocol. The finality protocols - Casper FFG for Ethereum 2.0 and GRANDPA for Polkadot - are both GHOST-based and can both finalize batches of blocks in one round. For block production, both protocols use slot-based protocols that randomly assign validators to a slot and provide a fork choice rule for unfinalized blocks - RandDAO/LMD for Ethereum 2.0 and BABE for Polkadot.

There are two main differences between Ethereum 2.0 and Polkadot consensus:

1. Ethereum 2.0 finalizes batches of blocks according to periods of time called "epochs". The current plan is to have 32 blocks per epoch, and finalize them all in one round. With a predicted block time of 12 seconds, this means the expected time to finality is 6 minutes (12 minutes maximum). [3] Polkadot's finality protocol, GRANDPA, finalizes batches of blocks based on availability and validity checks that happen as the proposed chain grows. The time to finality varies with the number of checks that need to be performed (and invalidity reports cause the protocol to require extra checks). The expected time to finality is 12-60 seconds.
2. Ethereum 2.0 requires a large number of validators per shard to provide strong validity guarantees. Polkadot can provide stronger guarantees with fewer validators per shard. Polkadot achieves this by making validators distribute an erasure coding to all validators in the system, such that anyone - not only the shard's validators - can reconstruct a parachain's block and test its validity. The random parachain-validator assignments and secondary checks performed by randomly selected validators make it impossible for the small set of validators on each parachain to collude.

Staking Mechanics

Ethereum 2.0 is a proof-of-stake network that requires 32 ETH to stake for each validator instance. Validators run a primary Beacon Chain node and multiple validator clients - one for each 32 ETH. These validators get assigned to "committees", which are randomly selected groups to validate shards in the network. Ethereum 2.0 relies on having a large validator set to provide availability and validity guarantees: They need at least 111 validators per shard to run the network and 256 validators per shard to finalize all shards within one epoch. With 64 shards, that's 16_384 validators (given 256 validators per shard). [4][5]

Polkadot can provide strong finality and availability guarantees with much fewer validators. Polkadot uses Nominated Proof of Stake (NPoS) to select validators from a smaller set, letting smaller holders nominate validators to run infrastructure while still claiming the rewards of the system, without running a node of their own. Polkadot plans to have 1_000 validators by the end of its first year of operation, and needs about ten validators for each parachain in the network.

Shards

Every shard in Ethereum 2.0 has the same STF. Each shards will submit "crosslinks" to the beacon chain and implement an eWasm execution environment. EWasm is a restricted subset of Wasm for contracts in Ethereum. The eWasm interface provides a set of methods available to contracts. There should be a similar set of development tools like Truffle and Ganache to develop for eWasm. [7]

Every shard in Polkadot has an abstract STF based on Wasm. Each shard can expose a custom interface, as long as the logic compiles to Wasm and the shard provides an "execute block" function to Polkadot validators. Polkadot has the Substrate development framework that allows full spectrum composability with a suite of modules that can be configured, composed, and extended to develop a chain's STF.

Message Passing

Shards in Ethereum 2.0 will have access to each other's state via their crosslinks and state proofs. In the model of Ethereum 2.0 with 64 shards, each one posts a crosslink in the Beacon Chain for every block, [4] meaning that shards could contain logic that executes based on some light client proof of a transaction on another shard. [8] Ethereum 2.0 has not released a specification for which nodes pass messages between shards.

Polkadot uses Cross-Consensus Message Passing Format (XCM) for parachains to send arbitrary messages to each other. Parachains open connections with each other and can send messages via their established channels. Given that collators will need to be full nodes of the Relay Chain as well, they will be connected and will be able to relay messages from parachain A to parachain B.. Messages do not pass through the Relay Chain, only proofs of post and channel operations (open, close, etc.) go into the Relay Chain. This enhances scalability by keeping data on the edges of the system.

Polkadot will add a protocol called SPREE that provides shared logic for cross-chain messages. Messages sent with SPREE carry additional guarantees about provenance and interpretation by the receiving chain.

Governance

Ethereum 2.0 governance is still unresolved. Ethereum currently uses off-chain governance procedures like GitHub discussions, All Core Devs calls, and Ethereum Magicians to make decisions about the protocol. [9]

Polkadot uses on-chain governance with a multicameral system. There are several avenues to issue proposals, e.g. from the on-chain Council, the Technical Committee, or from the public. All proposals ultimately pass through a public referendum, where the majority of tokens can always control the outcome. For low-turnout referenda, Polkadot uses adaptive quorum biasing to set the passing threshold. Referenda can cover a variety

of topics, including fund allocation from an on-chain Treasury or modifying the underlying runtime code of the chain. Decisions get enacted on-chain and are binding and autonomous.

Upgrades

Upgrades on Ethereum 2.0 will follow the normal hard-fork procedure, requiring validators to upgrade their nodes to implement protocol changes.

Using the Wasm meta-protocol, Polkadot can enact chain upgrades and successful proposals without a hard fork. Anything that is within the STF, the transaction queue, or off-chain workers can be upgraded without forking the chain.

Conclusion

Ethereum 2.0 and Polkadot both use a sharded model where shard chains ("shards" in Ethereum 2.0 and "parachains/parathreads" in Polkadot) are secured by a main chain by linking shard state in the blocks of the main chains. The two protocols differ in a few main areas. First, all shards in Ethereum 2.0 has the same STF, while Polkadot lets shards have an abstract STF. Second, governance processes in Ethereum 2.0 are planned to be off-chain and thus require coordination for a hard fork to enact governance decisions, while in Polkadot the decisions are on-chain and enacted autonomously. Third, the validator selection mechanisms are different because Polkadot can provide strong availability and validity guarantees with a smaller number of validators per shard.

References

1. Ethereum 2.0 Phases
2. Ethereum 2.0 Merge
3. Ethereum 2 Block Time
4. Ethereum 2.0 Economics
5. Buterin, Eth2 shard chain simplification proposal
6. Messari Crypto Theses for 2020
7. eWasm Design
8. Sharding FAQ
9. Ethereum Governance Compendium

Edit this page

*Last updated on **10/23/2021** by **Danny Salman***

Polkadot and Cosmos

Polkadot and Cosmos are both protocols that provide an interface for different state machines to communicate with each other. Both protocols are predicated on the thesis that the future will have multiple blockchains that need to interoperate with each other rather than individual blockchains existing in isolation.

Model

Polkadot uses a sharded model where each shard in the protocol has an abstract state transition function (STF). Polkadot uses WebAssembly (Wasm) as a "meta-protocol". A shard's STF can be abstract as long as the validators on Polkadot can execute it within a Wasm environment.

The shards of Polkadot are called "parachains". Every time a parachain wants to make a state transition, it submits a block (batch of state transitions) along with a state proof that Polkadot validators can independently verify. These blocks are finalized for the parachains when they are finalized by Polkadot's Relay Chain, the main chain of the system. As such, all parachains share state with the entire system, meaning that a chain re-organization of a single parachain would require a re-organization of all parachains and the Relay Chain.

Cosmos uses a bridge-hub model that connects Tendermint chains. The system can have multiple hubs (the primary being the "Cosmos Hub"), but each hub connects a group of exterior chains, called "zones". Each zone is responsible for securing the chain with a sufficiently staked and decentralized validator set. Zones send messages and tokens to each other via the hub using a protocol called Inter-Blockchain Communication (IBC). As zones do not share state, a re-organization of one zone would not re-organize other zones, meaning each message is trust-bound by the recipient's trust in the security of the sender.

Architecture

Polkadot

Polkadot has a Relay Chain acting as the main chain of the system. All validators in Polkadot are on the Relay Chain. Parachains have collators, who construct and propose parachain blocks to validators. Collators don't have any security responsibilities, and thus do not require a robust incentive system. Collators can submit a single parachain block for every Relay Chain block every 6 seconds. Once a parachain submits a block, validators perform a series of availability and validity checks before committing it to the final chain.

Parachain slots are limited, and thus parachain candidates participate in an auction to reserve a slot for up to two years. For chains that do not have the funding for a parachain slot or the necessity to execute with a six-second block time, Polkadot also has parathreads. Parathreads execute on a pay-as-you-go basis, only paying to execute a block when they need to.

In order to interact with chains that want to use their own finalization process (e.g. Bitcoin), Polkadot has bridge parachains that offer two-way compatibility.

Cosmos

Cosmos has a main chain called a "Hub" that connects other blockchains called "zones". Cosmos can have multiple hubs, but this overview will consider a single hub. Each zone must maintain its own state and therefore have its own validator community. When a zone wants to communicate with another zone, it sends packets over IBC. The Hub maintains a multi-token ledger of token balances (non-transfer messages are relayed but their state not stored in the Hub).

Zones monitor the state of the Hub with a light client, but the Hub does not track zone states. Zones must use a deterministic finality algorithm (currently, all use Tendermint) and implement the IBC interface to be able to send messages to other chains through the Hub.

Cosmos can also interact with external chains by using "peg zones", which are similar to bridged parachains.

Consensus

Polkadot uses a hybrid consensus protocol with two sub-protocols: BABE and GRANDPA, together called "Fast Forward". BABE (Blind Assignment for Blockchain Extension) uses a verifiable random function (VRF) to assign slots to validators and a fallback round-robin pattern to guarantee that each slot has an author. GRANDPA (GHOST-based Recursive Ancestor Deriving Prefix Agreement) votes on chains, rather than individual blocks. Together, BABE can author candidate blocks to extend the finalized chain and GRANDPA can finalize them in batches (up to millions of blocks at a time).

This isolation of tasks provides several benefits. First, it represents a reduction in transport complexity for both block production and finalization. BABE has linear complexity, making it easy to scale to thousands of block producers with low networking overhead. GRANDPA has quadratic complexity, but is reduced by a factor of the latency, or how many blocks it finalizes in one batch.

Second, having the capacity to extend the chain with unfinalized blocks allows other validators to perform extensive availability and validity checks to ensure that no invalid state transitions make their way into the final chain.

Cosmos (both the Hub and the zones) uses Tendermint consensus, a round-robin protocol that provides instant finality. Block production and finalization are on the same path of the algorithm, meaning it produces and finalizes one block at a time. Because it is a PBFT-based algorithm (like GRANDPA), it has quadratic transport complexity, but can only finalize one block at a time.

Staking Mechanics

Polkadot uses Nominated Proof of Stake (NPoS) to select validators using the sequential Phragmén algorithm. The validator set size is set by governance (1_000 validators planned) and stakers who do not want to run validator infrastructure can nominate up to 16 validators. Phragmén's algorithm selects the optimal allocation of stake, where optimal is based on having the most evenly staked set.

All validators in Polkadot have the same weight in the consensus protocols. That is, to reach greater than 2/3 of support for a chain, more than 2/3 of the *validators* must commit to it, rather than 2/3 of the *stake.* Likewise, validator rewards are tied to their activity, primarily block production and finality justifications, not their amount of stake. This creates an incentive to nominate validators with lower stakes, as they will earn higher returns on their staked tokens.

The Cosmos Hub uses Bonded Proof of Stake (a variant of Delegated PoS) to elect validators. Stakers must bond funds and submit a delegate transaction for each validator they would like to delegate to with the number of tokens to delegate. The Cosmos Hub plans to support up to 300 validators.

Consensus voting and rewards are both stake-based in Cosmos. In the case of consensus voting, more than 2/3 of the *stake* must commit, rather than 2/3 of the *validators.* Likewise, a validator with 10% of the total stake will earn 10% of the rewards.

Finally, in Cosmos, if a staker does not vote in a governance referendum, the validators assume their voting power. Because of this, many validators in Cosmos have zero commission in order to acquire more control over the protocol. In Polkadot, governance and staking are completely disjoint; nominating a validator does not assign any governance voting rights to the validator.

Message Passing

Polkadot uses Cross-Consensus Message Passing Format (XCM) for parachains to send arbitrary messages to each other. Parachains open connections with each other and can send messages via their established channels. Collators are full nodes of parachains and full nodes of the Relay Chain, so collator nodes are a key component of message passing. Messages do not pass through the Relay Chain, only proofs of post and channel operations (open, close, etc.) go into the Relay Chain. This enhances scalability by keeping data on the edges of the system.

In the case of a chain re-organization, messages can be rolled back to the point of the re-organization based on the proofs of post in the Relay Chain. The shared state amongst parachains means that messages are free from trust bounds; they all operate in the same context.

Polkadot has an additional protocol called SPREE that provides shared logic for cross-chain messages. Messages sent with SPREE carry additional guarantees about provenance and interpretation by the receiving chain.

Cosmos uses a cross-chain protocol called Inter-Blockchain Communication (IBC). The current implementation of Cosmos uses the Hub to pass tokens between zones. However, Cosmos does have a new specification for passing arbitrary data. Nonetheless, as chains do not share state, receiving chains must trust the security of a message's origin.

Governance

Polkadot has a multicameral governance system with several avenues to pass proposals. All proposals ultimately pass through a public referendum, where the majority of tokens can always control the outcome. For low-turnout referenda, Polkadot uses adaptive quorum biasing to set the passing threshold. Referenda can contain a variety of proposals, including fund allocation from an on-chain Treasury. Decisions get enacted on-chain and are binding and autonomous.

Polkadot has several on-chain, permissionless bodies. The primary one is the Council, which comprises a set of accounts that are elected in Phragmén fashion. The Council represents minority interests and as such, proposals that are unanimously approved of by the Council have a lower passing threshold in the public referendum. There is also a Technical Committee for making technical recommendations (e.g. emergency runtime upgrade to fix a bug).

Cosmos uses coin-vote signaling to pass referenda. The actual enactment of governance

decisions is carried out via a protocol fork, much like other blockchains. All token holders can vote, however, if a delegator abstains from a vote then the validator they delegate to assume their voting power. Validators in Polkadot do not receive any voting power based on their nominators.

Upgrades

Using the Wasm meta-protocol, Polkadot can enact chain upgrades and successful proposals without a hard fork. Anything that is within the STF, the transaction queue, or off-chain workers can be upgraded without forking the chain.

As Cosmos is not based on a meta-protocol, it must enact upgrades and proposals via a normal forking mechanism.

Development Framework

Both Cosmos and Polkadot are designed such that each chain has its STF and both provide support for smart contracts in both Wasm and the Ethereum Virtual Machine (EVM). Polkadot provides an ahead-of-time Wasm compiler as well as an interpreter (Wasmi) for execution, while Cosmos only executes smart contracts in an interpreter.

Cosmos chains can be developed using the Cosmos SDK, written in Go. The Cosmos SDK contains about 10 modules (e.g. staking, governance, etc.) that can be included in a chain's STF. The SDK builds on top of Tendermint.

The primary development framework for parachains is Substrate, written in Rust. Substrate comes with FRAME, a set of about 40 modules (called "pallets") to use in a chain's STF. Beyond simply using the pallets, Substrate adds a further layer of abstraction that allows developers to compose FRAME's pallets by adding custom modules and configuring the parameters and initial storage values for the chain.

> Note: Polkadot can support an STF written in any language, so long as it compiles to its meta-protocol Wasm. Likewise, it could still use the Substrate client (database, RPC, networking, etc.); it only needs to implement the primitives at the interface.

Conclusion

Polkadot was designed on the principle that scalability and interoperability require shared validation logic to create a trust-free environment. As more blockchains are developed, their security must be cooperative, not competitive. Therefore, Polkadot provides the shared validation logic and security processes across chains so that they can interact knowing that their interlocutors execute within the same security context.

The Cosmos network uses a bridge-hub model to connect chains with independent security guarantees, meaning that inter-chain communication is still bounded by the trust that the receiving chain has in the sending chain.

Edit this page

*Last updated on **10/23/2021** by **Danny Salman***

Polkadot Comparisons

Polkadot is a blockchain technology but makes some innovations that sets it apart from other popular chains.

In-Depth Comparisons

- vs Ethereum 2.0
- vs Cosmos

Other Comparisons

Ethereum 1.x

Ethereum is a smart contract blockchain that allows for general computation to be deployed on-chain and operated across the p2p network. Ethereum 1.x refers to the current Ethereum release and the immediately planned future upgrades.

The difference between Ethereum 1.x and Polkadot is quite large. Ethereum is a single chain that allows developers to extend its functionality through the deployment of blobs of code onto the chain (called smart contracts). Polkadot, as described in the whitepaper, is a fully extensible and scalable blockchain network that provides security and interoperability through shared state.

In practical terms, this means that the layer of abstraction between these two projects is remarkably different for developers. In Ethereum, developers write smart contracts that all execute on a single virtual machine, called the Ethereum Virtual Machine (EVM). In Polkadot, however, developers write their logic into individual blockchains, where the interface is part of the state transition function of the blockchain itself. Polkadot will also support smart contract blockchains for Wasm and EVM to provide compatibility with existing contracts, but will not have smart contract functionality on its core chain, the Relay Chain.

As such, Polkadot is a possible augmentation and scaling method for Ethereum 1.x, rather than competition.

Binance Smart Chain

Binance Chain is a Proof of Stake Authority (PoSA) blockchain used to exchange digital assets on Binance DEX. Binance Smart Chain is an EVM-compatible smart contract chain bridged to Binance Chain. Together, they form the Binance Dual Chain System. Binance Smart Chain is also a Proof of Stake Authority chain and allows users to create smart contracts and dapps.

Both chains are built with Cosmos SDK and therefore are a part of the Cosmos ecosystem. Due to specifics of the Cosmos architecture, interoperability of Binance Smart Chain is based on bridges. This means all validators of both chains are also bridge operators, therefore the security of the system relies on trusting validators. At the moment, there are 21 Binance Smart Chain validator nodes.

Polkadot has an entirely different purpose, as it was built to connect and secure unique blockchains. It is a protocol on which single blockchains (such as Binance Smart Chain) could be built and benefit from shared security, interoperability and scalability. Interoperability within Polkadot is based on pooled security on Polkadot, and the security of the entire Polkadot network, and has much stronger economic security.

Scalability based on bridges relies on each bridged chain finding its own set of validators, therefore duplicate resources are required. Scalability on Polkadot is based on the security of the Relay Chain, and as the number of validators in the active set on Polkadot are increased, more parachains can be supported.

Edit this page

*Last updated on **9/17/2021** by **Danny Salman***

Video Tutorials

Getting Started

- Polkadot for Beginners
- A Walkthrough of Polkadot's UI
- Polkadot Webinars
- How to protect yourself from scams

Tutorials

Accounts, Transfers & Staking

- How to create Polkadot account
- Understanding Accounts and Keys
- How to Stake DOT/KSM on Polkadot JS
- Using Polkadot with Ledger live
- How to Stake DOT/KSM with Ledger
- Why do transfers fail?
- Existential Deposit and Keep-Alive Checks
- Why should you Nominate on Polkadot and Kusama?
- Picking validators to Nominate (Stake)

Parachain Auctions and Crowdloans

- Contribute to Crowdloans using Polkadot JS
- Introduction to Parachain Slot auctions

Governance

- Voting on Referenda on Polkadot and Kusama
- Voting for Polkadot and Kusama council

For validators

- Validator Resources
- Why should you become a Validator?
- How to upgrade your node
- Roles and Responsibilities of a Validator

Technical Content

- Rust, Substrate and Polkadot
- Introduction to Substrate

Past and Ongoing Events

- Encode Polkadot Club 2021
- Polkadot Decoded 2021
- Polkadot Buildathon India 2021
- Polkadot Decoded 2020
- Hackusama Webinar Series 2020
- Web3 Builders Series 2020

Edit this page

*Last updated on **11/18/2021** by **Radha***