

# About Litentry

This page discusses the general functionality of the Litentry protocol.

Litentry is a decentralized, cross-chain identity aggregator that enables users to link their identities in a privacy-preserving context. We aim to give users full control over their personal data and enable them to gain social and economic value from it. Our protocol can be adopted in on-chain reputation, governance, DeFi, and customized data services.

## Our Mission statement


Litentry is unlocking verifiable personal data in a private and secure way, to pave the way for identity-based social and economic innovations.

## Guides: Jump right in

Follow our handy guides to get started on the basics as quickly as possible.

The parachain is our blockchain on the Kusama & Polkadot ecosystem.

The Identity Hub is the interface to our protocol and it allows users to interact with the parachain.

 [Parachain](#)

 [IdentityHub](#)

Next  
[Problem & Solution](#)



# Identity is fragmented.

## Problem & Solution

### The Problems:

- **Data Fragmentation.** Decentralized identity differs from traditional identity in that it is generated and managed without centralized control or oversight, using blockchains as a permissionless, distributed, and transparent database. As web3 continues to grow, more decentralized identity data will be generated and stored on the blockchain. However, this data is fragmented and lives on different networks and blockchains, making it difficult to manage and integrate.
- **Data Shortage for Web3 Services.** People are currently unwilling to be personally identified on the blockchain, resulting in a dearth of identity data being available for the creation of web3 products and services

### The Solution: Litentry's Value Proposition

Litentry allows users to move from fragmented, individual identities to aggregated identities. An aggregated identity creates a more comprehensive identity profile by creating an identity graph of the account relationships of the identity owner. This aggregated identity follows W3C DID standards and can potentially solve the problem of isolated ID registry systems that many tech companies face.

1. Litentry's decentralized identity pallet provides a secure and private way to link different Web3 wallets and Web2 accounts. This allows users to enhance their Web3 profiles, creating a more tangible and trustworthy digital identity.
2. Litentry's Trusted Execution Environment (TEE) infrastructure adds a privacy-preserving layer between raw blockchain data and data shared with a dApp. It also provides secure storage for the sensitive identity graph.
3. Identity Hub by Litentry enables identity management, personal data analysis, and grants granular access to control which data is shared with third parties.



Previous  
About Litentry

Next

High Level Functionality



# Make your identity tangible without being seen

## High Level Functionality

This page discusses a high level overview of the infrastructure and functionality of Litentry.

### The 3 layers required for decentralized identity data computation

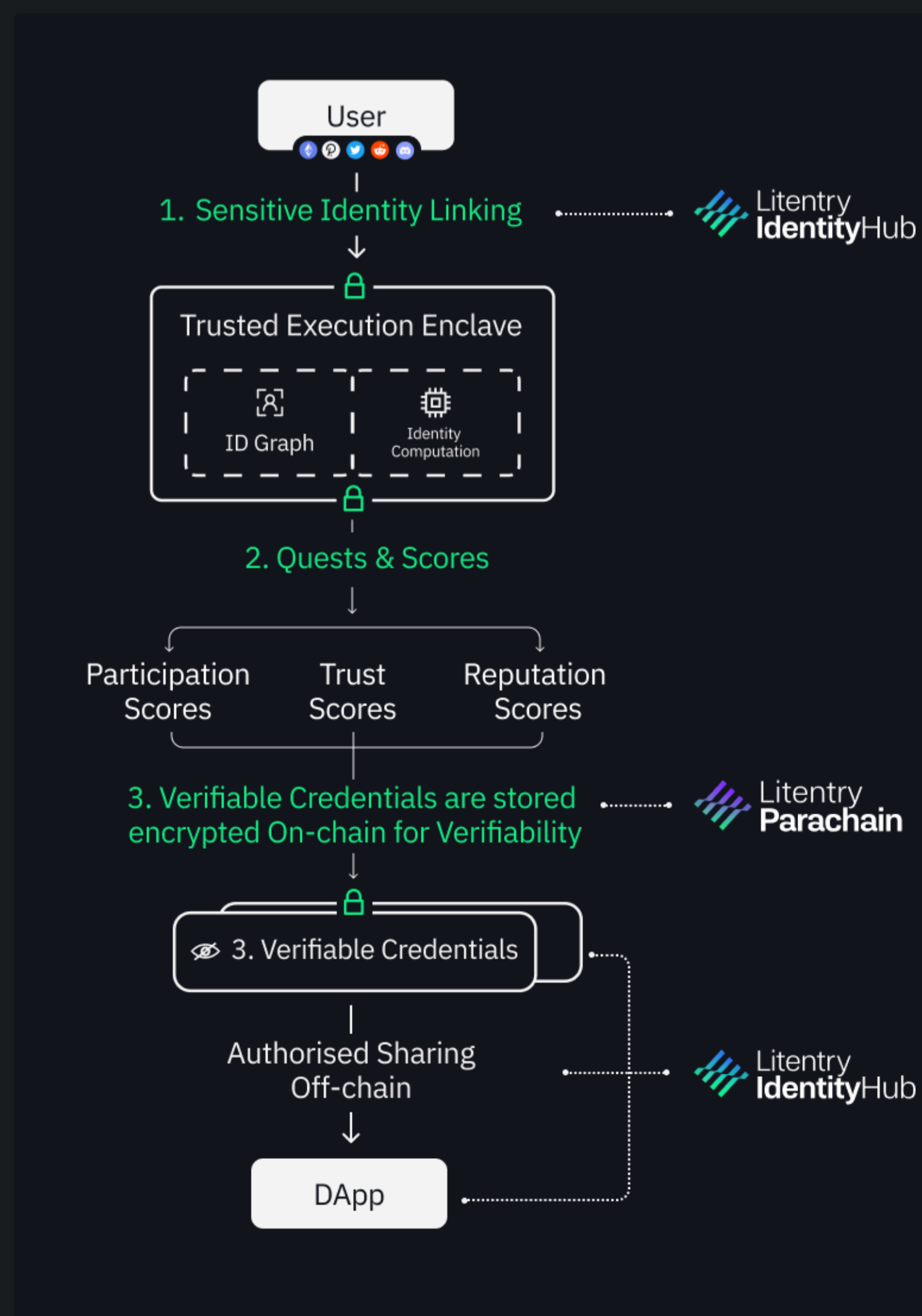
Litentry utilizes an infrastructure where identity data progresses from a disordered and scattered state to a structured state. The Litentry identity computation network consists of three core layers that create a verifiable, privacy-enhancing identity computation process. They are:

- Source data layer.** This layer obtains and indexes raw data from the blockchain and other networks, such as Etherscan, The Graph, Subquery, Onfinality, and other data providers.
- Address analysis layer.** This layer mainly serves to provide detailed data analysis or scores & labels, such as Chainalysis and Achainable.
- Identity aggregation layer.** Litentry enables users to store address relations associated with a single subject as an Identity Graph in a secure manner. This is achieved by encrypting the data and using Trusted Execution Environment (TEE) for computation. Persuaded by the Identity Graph, we obtain the corresponding address analysis results from the address analysis layer, and generate verifiable credentials or perform a weighted score calculation.

### The 3 stages of the Litentry Protocol functionality

The functionality of the Litentry protocol is divided into three main stages. These stages are interrelated and they interact together to ensure the privacy of user data.

- Linking sensitive identities:** The Litentry Protocol starts with the creation of an aggregated identity. In the Identity Hub, the user can prove ownership of their various accounts. The relationships between these accounts are stored in the form of an identity graph inside a Trusted Execution Environment (TEE). This TEE is a hardware black box where the sensitive account relationships are stored, managed, and calculated. It cannot be tampered with and is only visible to the root user.
- Generating scores and credentials:** When an identity score is requested from a specific user's identity graph, the necessary web2 and web3 data is fetched in real-time. The score or credential is calculated inside the TEE and issued as a verifiable credential without exposing any root accounts or metadata. The verifiable credential simply states the truth. It is stored encrypted on Litentry's parachains for verification purposes and sent to the user's local storage.
- Issuance of verifiable credentials:** Litentry uses W3C Verifiable Credential standards as the format for sharing identity scores or labels outside the Identity Hub. This allows for privacy-preserving, selective disclosure of identity data according to a self-sovereign identity framework.

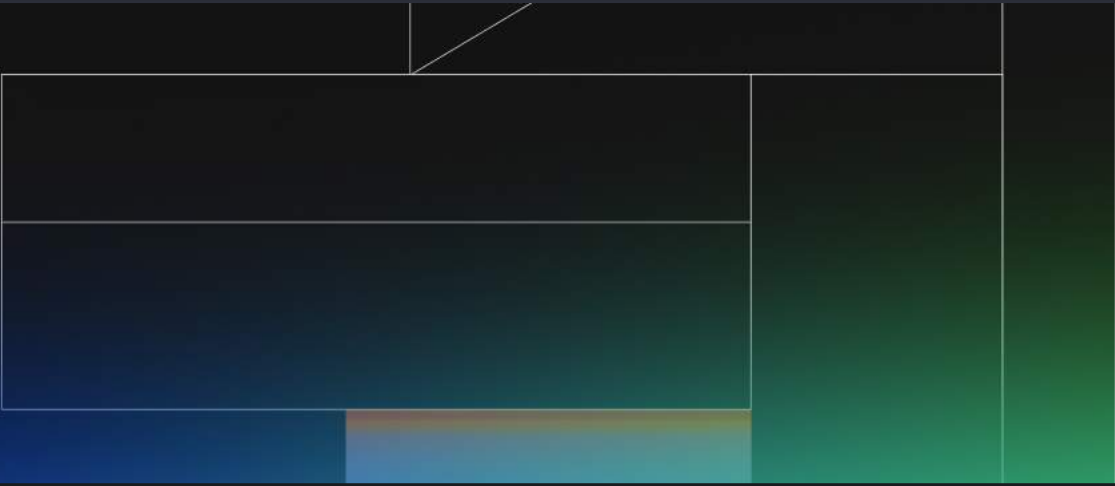


Previous  
Problem & Solution

Next

Identity Use Cases





# Identity Use Cases ⋮

This page discusses some of the use cases that Litentry unlocks

Litentry's identity aggregation and privatized verifiable credentials can be useful in several situations.

## Airdrop Whitelisting

Currently projects distribute airdrops to their users based on unique addresses, but not unique identities. The validation and contribution process is centralized and raises questions about transparency, token claim fees, and one-time distribution. The Litentry identity verification system allows crypto projects to identify and filter out low-quality engagement in an airdrop.

## Audience Selection & Community Insights

Decentralized identities allow communities and projects to gain a better understanding of their audience and reward them for sharing their pseudonymous identity data. By requesting specific, narrowed down details about their users, projects can respect the privacy of their ideal users.

## Soulbound Tokens or NFTs

The Litentry protocol allows projects and users to inject pseudonymous identity data into NFTs, dApps, and other apps in a private and secure way. An NFT can start to contain a personal "community participation score" based on various metrics of community engagement.

## Web3 Native Job Markets

Anonymous or pseudonymous verifiable credentials unlock a market for talent scouting, headhunting, and user research with highly experienced community members. Due to the increased level of privacy, participants can be judged on their relevant skills instead of being discriminated based on personal characteristics.

## Credit Scores

By providing a complete and thorough picture of a user's trading history, asset values, and borrowing behavior, we can provide credit scores. This increases the eligibility for under-collateralized lending and other reputation-based benefits or provides risk insights for partners.

## Cross-Platform Reputation

Litentry's identity aggregation makes it possible to transport reputation and status metrics across platforms from Web2 to Web3 and back.

← Previous  
**High Level Functionality**

Next →  
**FAQ**

# FAQ



Please feel free to ask any questions on our Telegram or Discord channels to help us refine this section with frequently asked question.

> **What's the history of Litentry?**

> **Why did Litentry start with Polkadot?**

> **What are the use cases for Litentry's protocol & front-end products?**

> **What products of litentry are currently working?**

> **What are the next chains that you would like to be more present on?**

> **What's the difference between Galxe and Litentry?**

> **Does Litentry make use of Zero Knowledge Proofs?**

> **Can Litentry see the sensitive relationships between the accounts in my profile?**

> **Is it possible to use the protocol without using the Identity Hub?**

> **How are you managing identity data and 3rd partners?**



Previous  
**Identity Use Cases**

Next - Parachain  
**Introduction**





# Litentry

## Introduction

A basic concept introduction to parachain

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.

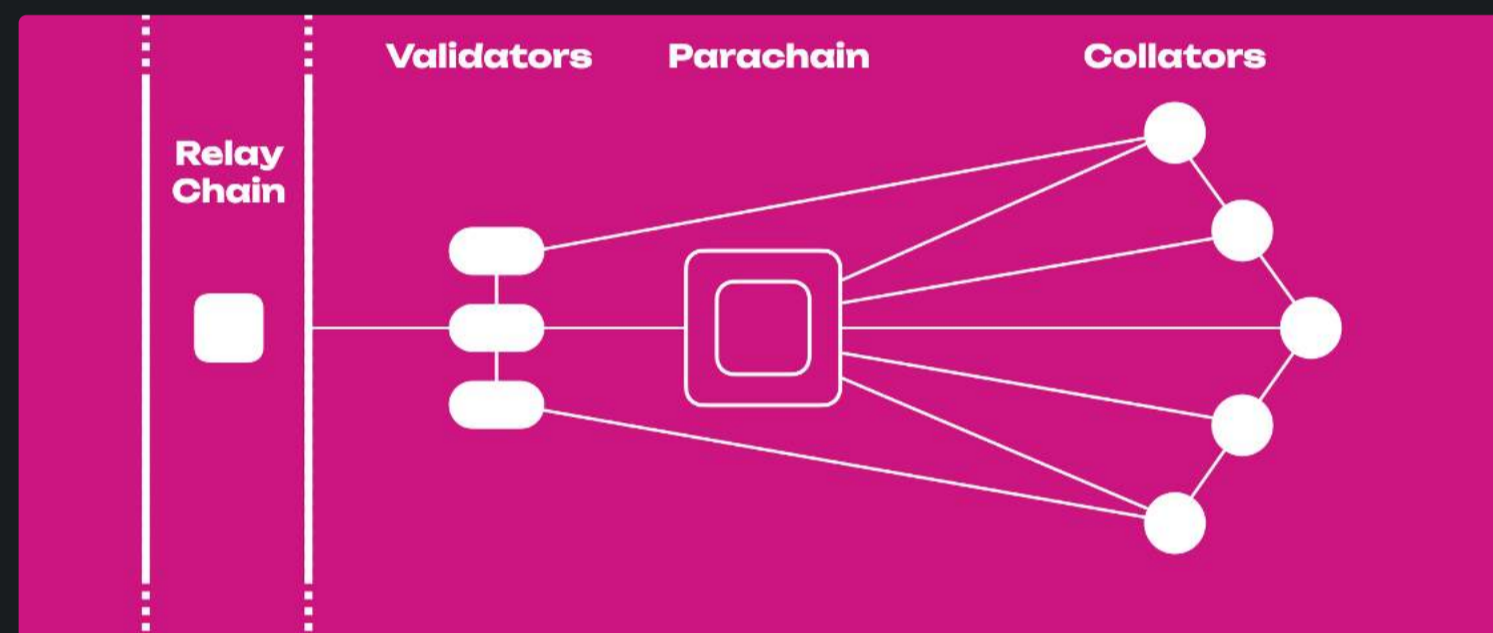


Image from <https://wiki.polkadot.network/docs/learn-parachains>

Basically, **parachains** are layer-1 blockchains that connect to the **relay chains** (Polkadot or Kusama for example), which validates the state transition of connected parachains, providing a shared state across the entire ecosystem. Since the validator set on the relay chains is expected to be secure with a large amount of stake put up to back it, it is desirable for parachains to benefit from this shared security.

Moreover, by using heterogeneous sharding, each parachain could be easily tailored through the [substrate](#) framework, optimizing them for a specific use case and running in parallel rather than the same across all shards.

To serve as the backbone platform for various Litentry products and achieve a transparent and decentralized user experience, we launched two parachains with customized features:

- [Litmus](#) on Kusama (onboarded on Feb 20th, 2022)
- [Litentry](#) on Polkadot (onboarded on July 4th, 2022)

[i](#) The source code of our parachain can be found here: <https://github.com/litentry/litentry-parachain>



Previous  
FAQ

Next - Parachain  
Get Started



# Get Started

Details of the Litmus/Litentry Parachains

## Litmus Network (Kusama)


Chain ID: **2106**

Onboarded: **Feb 20th, 2022**

Leasing period: **48** weeks

Polkadot-js endpoint: <https://polkadot.js.org/apps/?rpc=wss%3A%2F%2Frpc.litmus-parachain.litentry.io#/explorer>

Blockchain explorer: [Statescan](#)

 [Litmus Network](#)

## Litentry Network (Polkadot)

Chain ID: **2013**

Onboarded: **July 4th, 2022**

Leasing period: **96** weeks

Polkadot-js endpoint: <https://polkadot.js.org/apps/?rpc=wss%3A%2F%2Frpc.litentry-parachain.litentry.io#/explorer>

Blockchain explorer: [Statescan](#)

 [Litentry Network](#)



Parachain - Previous  
**Introduction**

Next  
**Litmus Network**





# Litmus

## Litmus Network

Entry page for Litmus network

Litmus is a companion canary network to Litentry and connects to the Kusama ecosystem as parachain. Litmus is expected to be fast-paced and integrated with cutting-edge technology, molding itself into a perfect field to experiment with new ideas and products before they finally land on Litentry parachain.

No new tokens will be issued on Litmus. Instead, we will provide a [token migration](#) mechanism to allow users to transfer their LIT tokens between Ethereum(ERC20) and Litmus.

### Outlines:

 [Rollout Plan](#)

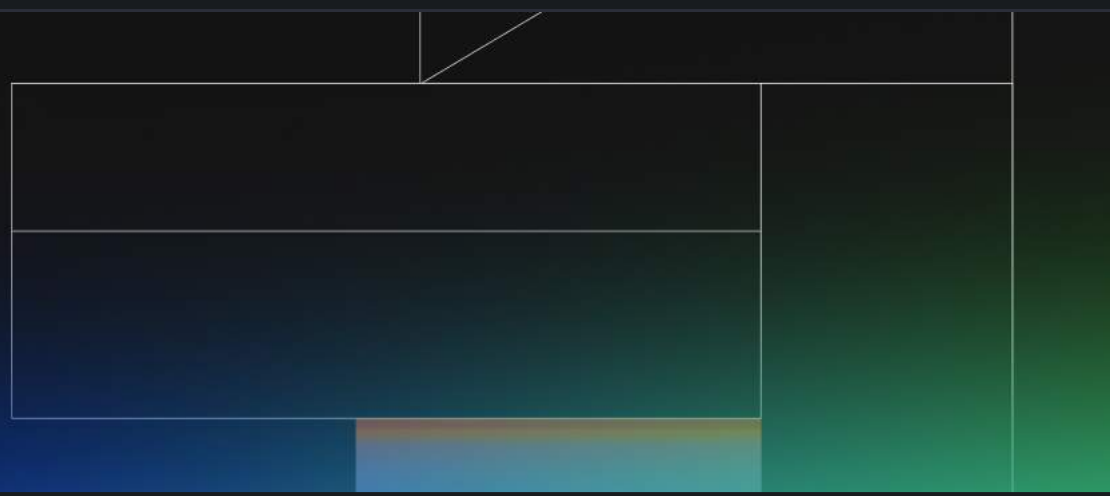
 [Collator](#)

 [Full Node](#)

[←](#) [Parachain - Previous](#)  
**Get Started**

[Next](#) [→](#)  
**Rollout Plan**





# Rollout Plan

This page describes the launch plan of Litmus parachain

## ✔ Phase 0 - Launch with Litmus genesis

Following [winning a Kusama parachain slot](#), the Litmus parachain was launched on **Feb 20th** with the genesis wasm. The initial launch will only include the minimum viable functionality to keep the chain running. It means:

- The parachain is running under the protection of a super-user ( `sudo pallet`).
- Most of the other pallets are deactivated, including governance, balance transfer and all other feature modules.
- The users' activities are very limited in this phase.

✔ What users can do

Users will be able to observe block production and finalization via explorer (e.g. [polkadot-js endpoint](#)), and perform the read-only RPC queries to the chain state and storage.

## ✔ Phase 1 - Technical verification & Crowdloan distribution

Once Litmus goes live, we will start to keep track of the chain state and verify that everything goes fine, including block authorization and finalization, collator status, extrinsic testing and runtime upgradability etc.

After we confirm that the parachain sits in a stable state and is running as expected, we will start to distribute the crowdloan rewards in LIT, with a linearly unlocked, per-block base.

✔ What users can do

Users can start to claim their crowdloan rewards. We will have [a tutorial page](#) later to explain how to claim it. Please note that the balance transfer remains **deactivated**, which means transferring the crowdloan reward to another account is still disabled.

## ✔ Phase 2 - Token migration & XCM

A token migration mechanism with corresponding pallets will be activated, which allows the user to transfer ERC20 LIT tokens to Litmus parachain.

✔ What users can do

Users can **optionally** migrate their ERC20 LIT tokens to Litmus parachain. [A guidance page](#) will be added later to demonstrate how to make such a transfer.

⚠ Initially, the token migration in this phase is uni-directional: ERC20 -> Litmus. However, we have now expanded the functionality and migration can also be done the other way round i.e Litmus -> ERC20.

A cross-chain mechanism based on XCM with other parachains will be active. These parachains processing DEX features will make secondary market trading of LIT available.

## ✔ Phase 3 - Sudo removal

After we confirm that the previous phases have been successfully carried out, we will start to activate the governance-related pallets in this phase with a subsequent `sudo` removal. From this point on, Litmus parachain will be fully governed by democracy, all on-chain state transitions that require `sudo` before will now have to go through a democratic process.

✔ What users can do

When the `sudo` key has been completely removed from the Litmus parachain, users will be able to observe the activation of governance features, including council information, technical committee, and referendum. LIT token holders can then take part in on-chain governance activities.

## ✔ Phase 4 - Balance transfer activation

Balance transfer has been enabled in this phase through a democratic runtime upgrade.

✔ What users can do

Users will be able to freely transfer their funds between accounts.

Note that Balance transfer and XCM are two independent mechanisms. So without balance transfer enabled, our users can still use the XCM function freely.

## ✔ Phase 5 - Feature pallets onboard

More feature pallets will be gradually onboarded in this phase. They are bound up with the Litentry products which widen the LIT token usage scenarios and bring the integrative service experience to the users.

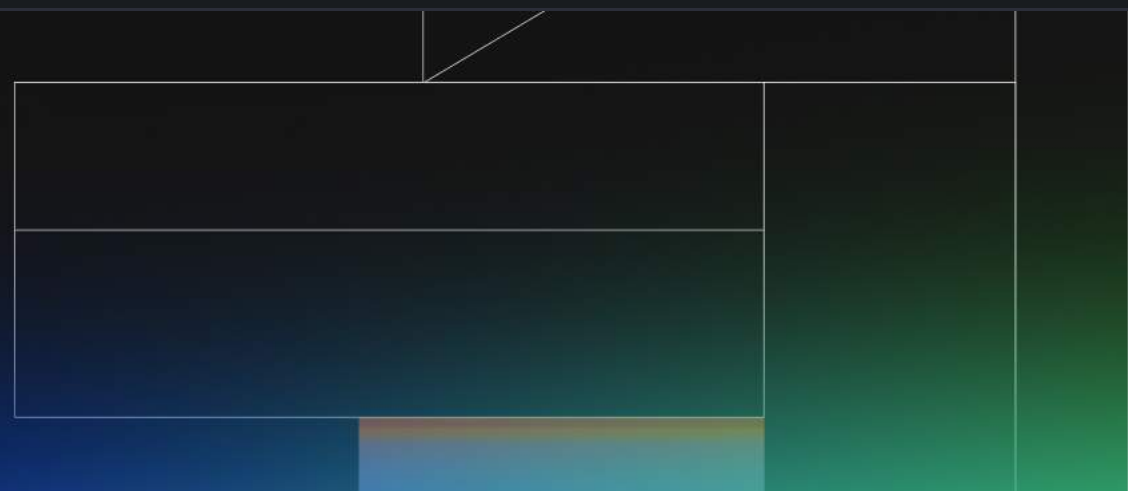
The feature pallets are expected to contain but are not limited to:

- Credit computation service with [TEE](#)
- [Identity Management Pallet](#)
- [VC Management Pallet](#)
- [Parachain Staking Pallet](#)
- NFT-based use cases

✔ What users can do

Users will be able to take part in the various products and services firsthand and discover the diverse LIT token utilities.

⚠ Depending on the development progress, some pallets could be experienced earlier than planned, in the form of internal/public testing.



# Collator

## Functionalities of collators

The Litmus [collators](#) retain all necessary information to collate and execute transactions to create an unsealed block and provide it, together with a proof of state transition, to one or more relaychain validators responsible for proposing a parachain block.

Collators will also watch the progress of block-producing and consensus protocols in BABE and build on what they think is the latest relay chain block that will be finalised. Collators do not directly participate in the consensus for the relay chain and therefore never stake DOT.

To become a collator and start to author blocks, one has to possess a valid session key, which is either set in the initial genesis config as **invulnerables** or by selection via the cumulus [collator-selection pallet](#).

## Litmus collator model

Currently, to keep it simple and focus on product development, we are running collators with our own nodes for Litmus. However, we are not ruling out the possibility to invite third-party or the community to join our collator set when we think the time is right.

If you want to run a full node, please refer to the [Full Node Guide](#).

← Previous **Rollout Plan** Next **Full Node** →



# Full Node



✔ Please refer to the [Full Node](#) chapter in Litmus.

Launching a Litentry full node is very similar to Litmus, you just need to replace the parachain type with `litentry` and relaychain type with `polkadot`. The commands are listed here again for reference:

## using docker

```
docker run -d --network=host -v /var/lib/litentry:/data \  
-u $(id -u):$(id -g) \  
litentry/litentry-parachain:v0.9.11 \  
--base-path=/data \  
--name="litentry-node" \  
--chain=litentry \  
--state-pruning=archive \  
--state-cache-size 0 \  
--ws-external \  
--rpc-external \  
--rpc-cors=all \  
--execution=wasm \  
-- \  
--execution=wasm \  
--chain polkadot
```

## using binary

```
./target/release/litentry-collator \  
--name="litentry-node" \  
--chain=litentry \  
--state-pruning=archive \  
--state-cache-size 0 \  
--ws-external \  
--rpc-external \  
--rpc-cors=all \  
--execution=wasm \  
-- \  
--execution=wasm \  
--chain polkadot
```



Previous  
Collator

Next - Parachain

Pallets and Modules





# Litentry

## Litentry Network

Litentry is a parachain that connects to the Polkadot ecosystem. As compared to Litmus, Litentry is expected to be more stable and have longer iteration cycles.

No new tokens will be issued on Litentry. Instead, we will provide a [token migration](#) mechanism to allow users to transfer their LIT tokens between Ethereum(ERC20) and Litentry.

### Outlines:

 [Collator](#)

 [Rollout Plan](#)

 [Full Node](#)



Previous  
[Full Node](#)

Next  
[Rollout Plan](#)



# Rollout Plan

This page describes the launch plan of Litentry parachain

## Phase 0 - Launch with Litentry genesis

Following winning a Polkadot parachain slot on Apr 25th, the Litentry parachain was launched on Jun 4th with the genesis wasm. The initial launch will only include the minimum viable functionality to keep the chain running. It means:

- The parachain is running under the protection of a super-user ( sudo pallet).
- Most of the other pallets are deactivated, including governance, balance transfer and all other feature modules.
- The users' activities are very limited in this phase.

### What users can do

Users will be able to observe block production and finalization via explorer (e.g. polkadot-js endpoint), and perform the read-only RPC queries to the chain state and storage.

## Phase 1 - Technical verification & Crowdloan distribution

Since Litentry went live, we keep track of the chain state and verify that everything goes fine, including block authorization and finalization, collator status, extrinsic testing, and runtime upgradability, etc. Most of Litmus features are available on Litentry, and we will make runtime upgrade for new features constantly in the future after Litmus confirm them as fully tested and robust.

After we confirm that the parachain is stable and running as expected, we will start distributing the crowdloan rewards in LIT, with a linearly unlocked, per-block base.

### What users can do

Users can start to claim their crowdloan rewards. We will have a tutorial page later to explain how to claim it. Please note that the balance transfer has been activated, which means the crowdloan reward can now be transferred to another account.

## Phase 2 - Token migration & XCM

A token migration mechanism with corresponding pallets has been activated, which allows the user to transfer ERC20 LIT tokens to Litentry parachain.

### What users can do

Users can optionally migrate their ERC20 LIT tokens to Litentry parachain. A guidance page will be added later to demonstrate how to make such a transfer.

Please note the token migration in this phase is uni-directional: ERC20 -> Litentry. We will expand it to allow the other direction in the near future.

A cross-chain mechanism based on XCM with other parachains will be active. These parachains processing DEX features will make secondary market trading of LIT available.

## Phase 3 - Collator Staking

Unlike Litmus, Litentry will have an extended staking mechanism based on Polkadot's Proof-of-Stake model as well as a delegation feature, making it a Delegated Proof of Stake(DPoS) setup.

Collators are initially maintained by Litentry team and users make a delegation request through staking, which allows users to participate in the block reward process.

Please refer to # Litentry collator model for more information

## Phase 4 - Sudo removal

After we confirm that the previous phases have been successfully carried out, we will start to activate the governance-related pallets in this phase with a subsequent sudo removal. From this point on, Litentry parachain will be fully governed by democracy, all on-chain state transitions that require sudo before will now have to go through a democratic way.

### What users can do

When the sudo key has been completely removed from the Litmus parachain, users will be able to observe the activation of governance features, including council information, technical committee, and referendum. LIT token holders can then take part in on-chain governance activities.

## Phase 5 - Balance transfer activation

Balance transfer has been enabled in this phase through a democratic runtime upgrade.

### What users can do

Users will be able to freely transfer their funds between accounts.

Notice that Balance transfer and XCM are independent two mechanisms. So without balance transfer enabled, users can still use the XCM function freely.

## Phase 6 - Feature pallets onboard

More feature pallets will be gradually onboarded in this phase. They are bound up with the Litentry products which widen the LIT token usage scenarios and bring the integrative service experience to the users.

The feature pallets are expected to contain but are not limited to:

- Crowdloan Distribution (Linear Vesting Unlock)
- A bidirectional token bridge between ERC20 LIT tokens and Litentry parachain (first unidirectional, then extend to both directions)
- XCM cooperation with available Defi-hub (e.g. Moonbeam/Acala) to liquidate and invest LIT in their financial products
- XCM support between Litmus and Litentry
- Activation of PoS collation and collator-related staking (Please note this is for the Litentry parachain only, not on Litmus)
- Cooperation with PNS (Polkadot Name Systems)
- LIT utility: On-chain governance and tip/bounty program
- LIT utility: Privacy-preserving identity aggregation service
- LIT utility: Privacy-preserving airdrop distribution and claiming service
- Privacy-preserving anti-fraud scoring service
- Credit computation service with TEE
- Parachain Staking Pallet
- VC Management Pallet
- Identity Management Pallet
- NFT-based use cases

### What users can do

Users will be able to take part in the various products and services firsthand and discover the diverse LIT token utilities.

Depending on the development progress, some pallets could be experienced earlier than planned, in the form of internal/public testing.



# Full Node



## Litmus RPC nodes

By default, Litmus has a few self-hosted, load-balanced RPC nodes which provide public service. Contrary to [collators](#), the RPC nodes don't produce blocks but only sync the chain states and provide RPC/Websocket services to the users.

You can find the Litmus RPC endpoint [here](#) on polkadot-js.

With the service, the end-users can query the chain state, inspect the constant and storage, and execute extrinsics.

## Run your own full nodes

### using docker (preferred)

1. create a local directory to store the chain database:

```
mkdir /var/lib/litentry
# or use sudo if you don't have permission
sudo mkdir /var/lib/litentry
```

2. make sure the permission and ownership of the local directory are correctly set:

```
sudo chown -R $(id -u):$(id -g) /var/lib/litentry
```

3. run the following docker command, you can replace the `--name="litmus-node"` with your own node name:

```
docker run -d --network=host -v /var/lib/litentry:/data \
-u $(id -u):$(id -g) \
litentry/litentry-parachain:v0.9.11 \
--base-path=/data \
--name="litmus-node" \
--chain=litmus \
--state-pruning=archive \
--state-cache-size 0 \
--ws-external \
--rpc-external \
--rpc-cors=all \
--execution=wasm \
-- \
--execution=wasm \
--chain kusama
```

ⓘ `litentry/litentry-parachain:v0.9.11` is used as an example, please check [github release page](#) for the up-to-date releases

The command will run the docker container in the background and the container ID will be printed in the console. With `docker logs -f <container-id>` you should be able to see the node starts to sync.

Wait until syncing is done, depending on the hardware and network status it could take several days to fully sync the parachain and relaychain database.

After it's fully synced, you should be able to access the chain via local ws endpoint in polkadot-js: <https://polkadot.js.org/apps/?rpc=ws%3A%2F%2F127.0.0.1%3A9944#/explorer>

### using binary

Running a full node with the raw binary is very similar to the docker setup above, it only differs a bit in the command line arguments. So instead of step 1-3 above, run:

```
./target/release/litentry-collator \
--name="litmus-node" \
--chain=litmus \
--state-pruning=archive \
--state-cache-size 0 \
--ws-external \
--rpc-external \
--rpc-cors=all \
--execution=wasm \
-- \
--execution=wasm \
--chain kusama
```

By default the database is stored at `~/local/share/`, you can override it by using `--base-path=<your-path>`.

ⓘ To get the binary, you could either download it directly from Litentry's [Github release page](#) (**Linux x86-64** only), or [build](#) it from the source.



Previous  
Collator

Next  
Litentry Network



# Identity is fragmented.

## Pallets and Modules ⋮

A detailed description of feature modules in parachain

This chapter portrays the feature pallets and modules that are used in parachain in more detail.

At the moment it includes:


 [Verifiable Credential Management Pallet \(VCMP\)](#)


 [Identity Management Pallet \(IMP\)](#)

 [Teerex Pallet](#)

 [Token Bridge](#)

 [TEE](#)

 [Litentry Identity Registrar](#)

 More subchapters will be added as we onboard more feature pallets.

[←](#) Previous  
**Full Node**

Next [→](#)  
**Verifiable Credential Management P...**



# Verifiable Credential Management Pallet (VCMP)

The Verifiable Credential Management Pallet (VCMP) enables the creation, management, and verification of verifiable credentials.

## Introduction

The VCMP provides a way for users to manage the lifecycle of their verifiable credentials, including creating, issuing, revoking, and updating them. It also enables individuals to store and manage their own verifiable credentials, granting them more control over their personal data and the ability to selectively disclose it to others.

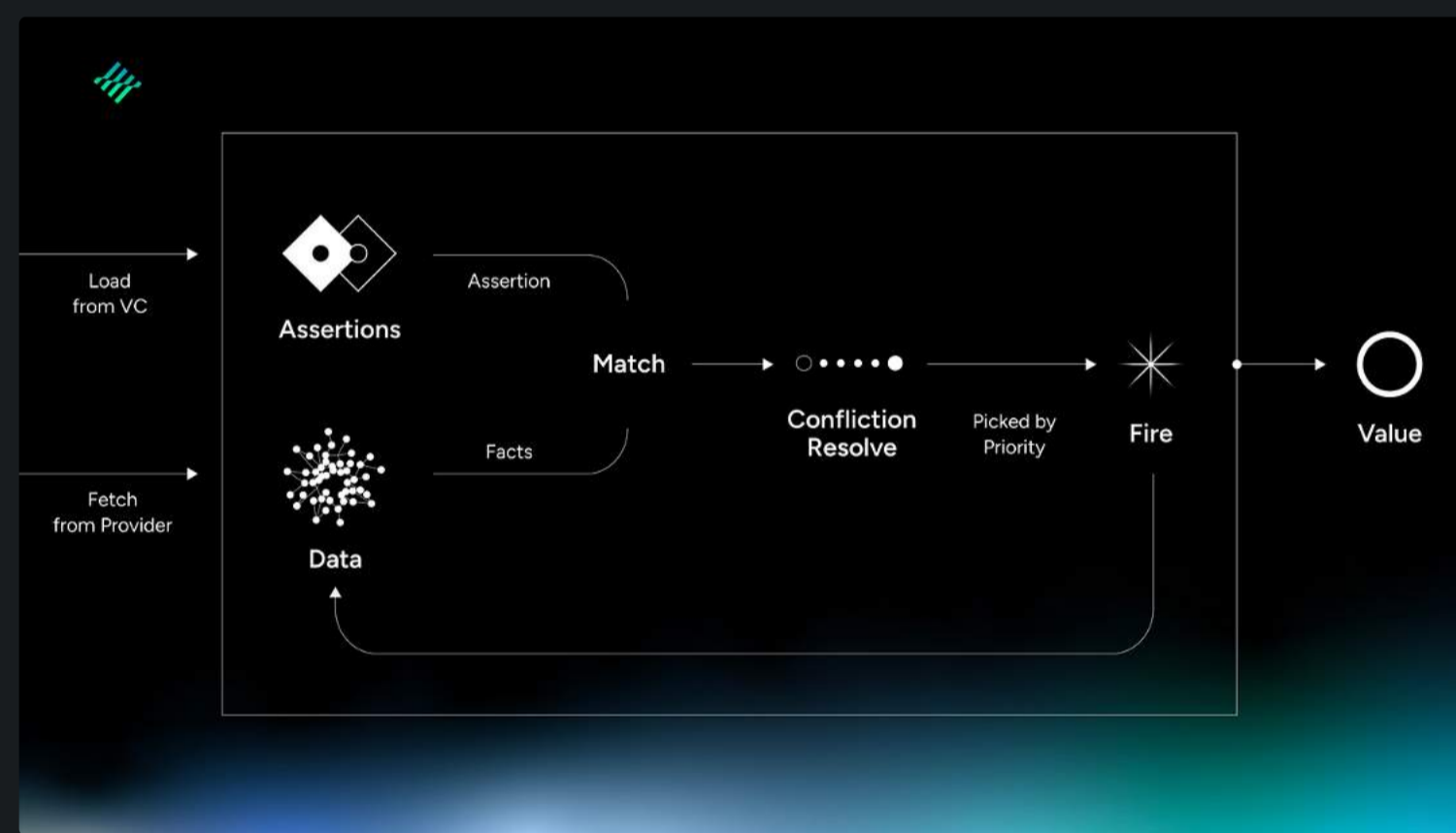
The VCMP interoperates with the Litentry IdentityHub (IDHub) which incorporates features such as secure storage, user authentication, and audit logging to enhance the security and trustworthiness of the credentialing process.

Overall, it is an important component of the identity ecosystem, providing a standardized way for users to create and manage verifiable credentials that can be trusted and verified by others.

Since the VCMP manages VCs, it is important to discuss what VCs are - Verifiable credentials (VCs) are stored as clear text JSON files, which use the JavaScript Object Notation (JSON) format for storing and exchanging data. JSON is a text-based format that uses human-readable text to represent data objects consisting of attribute-value pairs and array data types. It is often used for transmitting data over networks or storing data in NoSQL databases because it is lightweight and easy to read and write.

In a JSON file, data is organized into key-value pairs, with keys represented by strings enclosed in quotation marks and values represented by strings, numbers, booleans, arrays, or other data types. In the case of Litentry VCs, the JSON files are composed of three main parts:

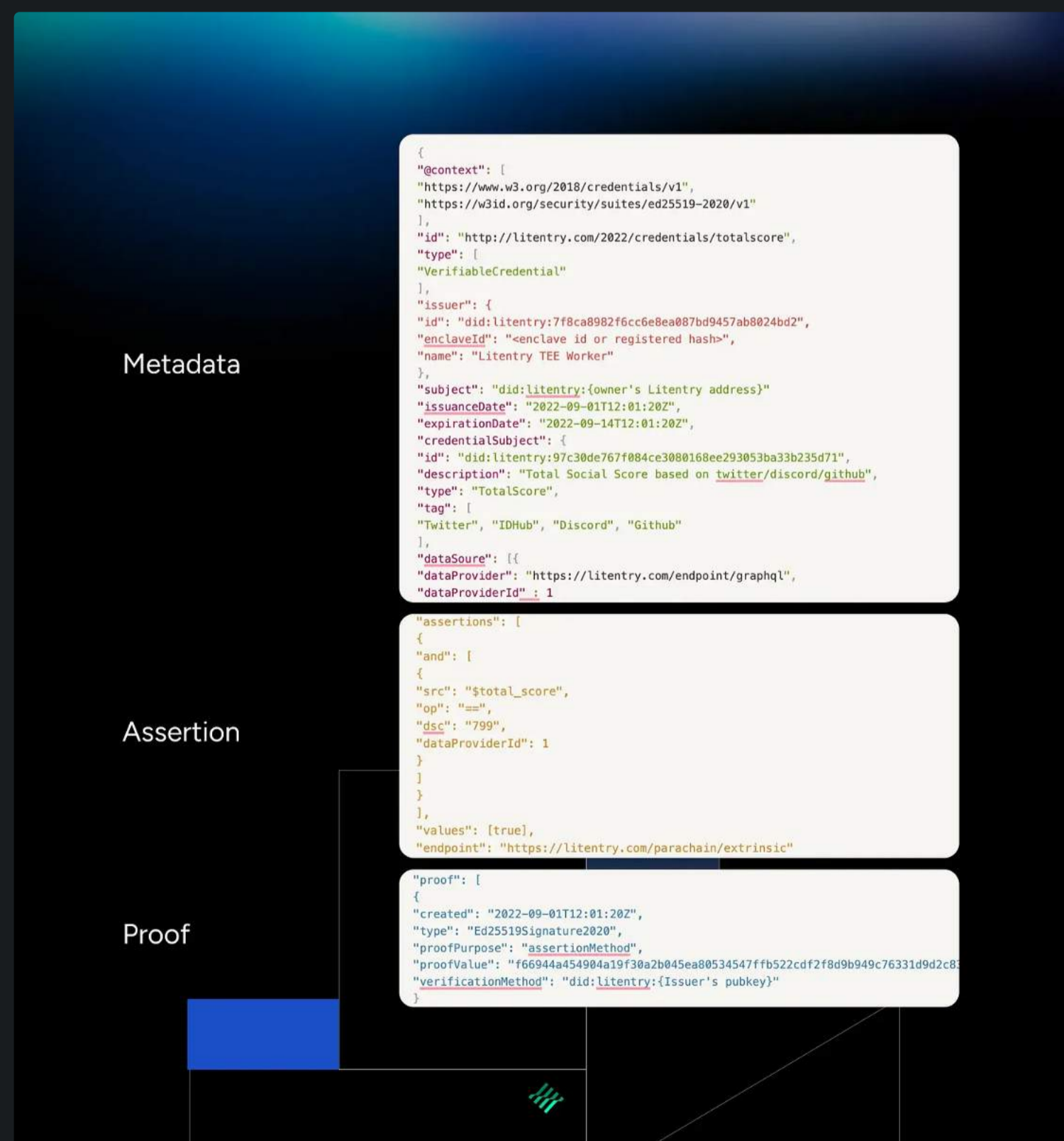
1. **The issuer's enclave attestation metadata** — Verifiable Credentials often include metadata about the issuer, which is the entity that issues the VCs. This metadata is called the issuer's enclave attestation metadata and it provides information about the security of the issuer's enclave. An enclave is a secure, isolated environment within a computer or device that is used to protect sensitive information, such as cryptographic keys and private data. The issuer's enclave attestation metadata includes details about the type of enclave used by the issuer, the security measures and capabilities of the enclave, and any relevant security certifications or attestations. This information allows credential verifiers to assess the security of the issuer's enclave and determine the trustworthiness of the VCs issued by the issuer. In a verifiable credential system, the issuer's enclave attestation metadata may be included as part of the VC or stored separately and referenced by the VC. This allows credential verifiers to easily access and verify the metadata when verifying the authenticity of a VC.
2. **The Assertions** — Verifiable Credentials often include assertions, which are statements made by the credential issuer about the information contained in the VC. Assertions specify the subject of the credential, the type of information being asserted, and the value of the information being asserted. They are written in a domain Specific Language (DSL) that contains a set of steps to describe how to fetch and calculate data. The DSL is a type of code that can be automatically generated into Parachain runtime codes or TypeScript SDK codes. It enables flexibility, isolation, and standardization of data, as well as improves the verifiability of assertions (data, values, information, etc.) through trustless, automatically generated codes. Assertions have their own execution flow that defines how they are executed based on data that matches the VC registry. They are used to define the standard of the Litentry context based on the data provided by data providers. In other words, assertions help ensure that VCs contain accurate and trustworthy information that can be easily verified.



Assertions

3. **The signature proof** — this is provided by the issuer enclave's private key and it is a type of digital signature that is used to authenticate the identity of the issuer and the integrity of the issued document. In essence, the signature is a cryptographic function that takes the issuer's private key and the document's content as inputs and produces a unique signature as output. This signature can then be verified using the issuer's public key, which is typically shared with the recipient of the document. This allows the recipient to confirm that the document has not been tampered with and that it was indeed issued by the issuer.

Each of the JSON file components is shown below:



Components of a VC JSON file

# Parts of a VCMP

Explanation of the different components that makes up the VCMP

## VC Registry

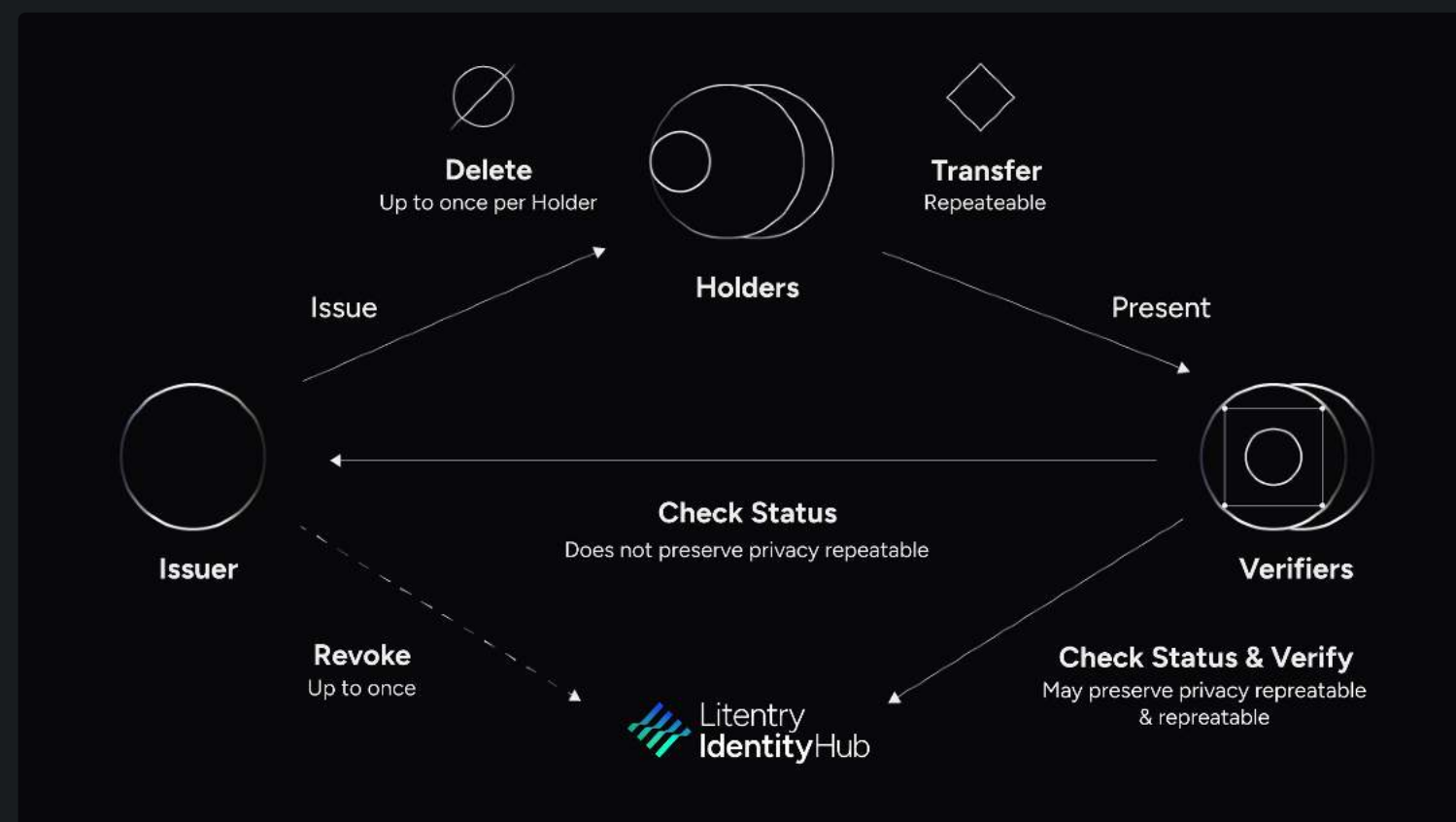
The VC registry is a component of the VCMP that allows users to store and manage their verifiable credentials, and make them available to others who need to verify their identity or other attributes.

The VC registry is an onchain and public storage on the VCMP that maintains the `VC Index` <> `VC Hash` so that credentials can be verified. It contains only a map from the VC index and VC hash which is used to check the validity of the VC content. When a VC is generated, the Parachain VCMP extrinsic `vc_issued` is called (one of the parameters is the generated VC, encrypted by the user's shielding key; and the other parameters are `vc_index`, `vc_hash`, etc), and VCMP outputs an `VC_issued` event. At the same time, the VCMP inserts a new record into the VC Registry, the key is `vc_index`, and the value is `vc_context` (this includes subject, `vc_hash` and `vc_status`). The default `vc_status` is active.

The registry does not store all the content of a VC. Rather, it stores only the VC index and context to ensure that the privacy of users is protected. It is responsible for maintaining a list or directory of issued credentials. The registry typically includes the VC subject (the individual to whom the credential was issued), the VC status (active, revoked, expired, etc.), and the VC hash. This information can be accessed by verifiers to verify the authenticity and validity of a credential presented by a holder.

The registry can also provide additional services, such as allowing holders to manage their credentials, enabling issuers to publish and share information about their credentials, or withdrawing access to the credential.

Summarily, the cycle of the sequence of events in a verifiable credential ecosystem is as follows:



1. Issue. The first event in the sequence is the issuance of a VC. This involves an issuer issuing a verifiable credential to a holder.
2. Transfer. Once received by the holder, the holder may transfer one or more of its VC to another holder or may optionally present its VC(s) to a verifier.
3. Verify. The verifier determines the authenticity of the VC and this includes a status check for revocation of the verifiable credential.
4. An issuer might then decide to revoke a verifiable credential; or
5. A holder might delete a verifiable credential.

Although the order of the actions above is not always fixed, the most common sequence of action is usually:

- An issuer issues to a holder
- The holder presents to a verifier
- The verifier verifies.

## Map, VC Index, and VC Context

The VC registry is made up of a **Map** and it consists of Index as a key, and Context as a value. The map is typically represented as a JSON object that contains a key-value pair for each context aim in the credential.

The **VC Index** is the key that provides a way for issuers and verifiers to find and share credentials with each other in a secure and efficient manner.

The **VC Context** is an important component of the registry it helps to ensure interoperability and consistency in the creation, verification, and management of VCs. It doesn't contain any original VC content to protect user privacy but it acts as the value and is made up of the following three fields; Subject, Hash, and Status.

- **Subject**

A subject is an entity to which the VC is issued and the VC is about. The subject can be an individual person, an organization, or any other entity that can be described using a set of claims or contexts.

- **Blake2\_256 Hash of VC:**

The Blake2\_256 hash of a verifiable credential is a cryptographic digest generated using the Blake2\_256 hashing algorithm. This hash can be used to verify the integrity and authenticity of the credential, as any tampering with the contents of the credential will result in a different hash value.

- **Status**

Status refers to the current validity or state of the VC registry. The status can indicate whether the credential of a user is active, disabled, or revoked. The status of a verifiable credential can change over time, depending on various factors, e.g, if the issuer revokes or suspends the credential for some reason.

The status of a verifiable credential is an important aspect of its validity and can be used by relying parties to make informed decisions about whether to trust and rely on the claims contained in the verifiable credential.

# VCMP Extrinsic, Events, and Error Events

## VCMP Extrinsic

VCMP extrinsic is critical to ensuring the security, privacy, and usability of verifiable credentials. The common VC management extrinsic includes request, issuance, verification, revocation, and management. Below are the Litentry VCMP extrinsics and their callers:

- `request_vc` - VC is requested onchain- called by the user
- `disable_vc` - VC is disabled onchain- called by the user
- `revoke_vc` - VC is revoked onchain- called by the user
- `vc_issued` - VC is issued - called by the TEE

## VCMP event

VCMP events are the different events that occur during the lifecycle of a verifiable credential, such as its creation, issuance, revocation, and update. These events are important for ensuring the integrity and trustworthiness of the VC.

By tracking these events, issuers and credential holders can ensure that their verifiable credentials are up-to-date and accurate. Here are some of our VCM events:

- `VCRrequested`
- `VCDisabled`
- `VCRrevoked`
- `VCIssued`
  - Account
  - `VCIndex`
  - VC (by AES encrypted with User Shielding Key)

## VCMP error event

A VCMP Event can be explained as any unexpected situation or issue that occurs during the process of issuing, managing, and using a VC.

It is crucial to address these errors to ensure the integrity and security of the VC management pallet. This can be achieved through implementing proper error handling mechanisms and protocols and regularly monitoring and updating the system to prevent errors from occurring in the first place. Some of the VC errors that can be encountered include:

- `VCNotExist` - the ID doesn't exist
- `VCAAlreadyExists` - the VC already exists
- `VCSubjectMismatch` - The requester doesn't have permission (because of subject mismatch)
- `VCAAlreadyDisabled` - The VC is already disabled
- `HttpRequestFailed` - { reason: ErrorString }
- `RequestVCHandlingFailed`
- `ParseError`
- `Assertion1Failed`,
- `Assertion2Failed`,
- `Assertion3Failed`,
- `Assertion4Failed`,
- `Assertion5Failed`,
- `Assertion6Failed`,
- `Assertion7Failed`,
- `Assertion8Failed`,
- `Assertion10Failed`,
- `Assertion11Failed`,



Previous  
Parts of a VCMP

Next

Assertion definitions and parameters



# Assertion definitions and parameters

An Assertion is a **DSL**(Domain Specific Language) that contains a set of steps to describe how to fetch and calculate the final data. The DSL itself is code and can be auto-generated to Parachain runtime codes or TypeScript SDK codes.

- **pros**
  - enabling flexibility, isolation, and abstract module of assertions;
  - standardizes the data fetch and assertion calculate/validate logic;
  - it can cover most use cases for the issuer;
  - improve the verifiability of assertions with trustless auto-generated codes.
- **cons**
  - has a higher barrier for issuers to organize or write assertions;
  - has a larger json file size.

The supported `src`, `op` and `dst` in each `Rule` are described as follows:

- **Data type of `src` and `dst`**
  - NULL
  - Boolean
  - Number: int / long / float / double/decimal
  - String
  - Object
  - Array[Boolean/Number/String/Object]
- **Operator `op`**
  - +(add), -(subtract), \*(multiply), /(divide)
  - %, round()
  - `>`, `>=`, `<`, `<=`, `==`, `!=`
  - in, not in
  - contains, not contains
  - includes one
  - size, sum, average
  - owns
- **Data type and Operator Binding**
  - Boolean: `>`, `>=`, `<`, `<=`, `==`, `!=`, in, not in
  - Number: `>`, `>=`, `<`, `<=`, `==`, `!=`, in, not in
  - String: `>`, `>=`, `<`, `<=`, `==`, `!=`, in, not in
  - Object: (if comparable) `>`, `>=`, `<`, `<=`, `==`, `!=`,
  - Array: `==`, `!=`, in, not in, contains, not contains, includes one, size, sum, average

Left Value / Right Value	NULL	Bool	Number	String	Object	Array
NULL	==, !=	==, !=	==, !=	==, !=	==, !=	in, not in,
Bool	==, !=	==, !=				in, not in,
Number	==, !=		>, >=, <, <=, ==, !=,			in, not in,
String	==, !=			==, !=, in, not in, contains, not contains		in, not in,
Object	==, !=				>, >=, <, <=, ==, != (if comparable)	in, not in,
Array	contains, not contains	contains, not contains	contains, not contains	contains, not contains	contains, not contains	in, not in, contains, not contains, includes one

# Identity Management Pallet (IMP) ⋮

## Introduction

The **Identity Management Pallet (IMP)** is a powerful tool designed to facilitate the management of users' web2 and web3 identities. There are two types of IMP. [One](#) is located on the Parachain and it is the user portal with no storage functionality. It has extrinsics and events (normal and error events) and all input and output data are encrypted by user's shielding key. The [second](#) one is on the Trusted Execution Environment (TEE). It maintains the storage of users' shielding keys and IdGraphs and runs the actual identity verification business logic.

The pallet located in the TEE (enclave) is integrated in SGX-runtime, and the extrinsics are called by the enclave. When requesting *identity linking* (Web2 <> Web3, or cross-chain wallets linking) or *verifiable credential generation* in the IdentityHub, all data involved to complete the request will be stored and computed in the TEE environment. This includes the request itself, the relationship between different wallets, data fetched from a specific wallet that supports the claim in a VC, etc.

The TEE is secured by an isolated, cryptographic electronic structure that is resistant to malicious attacks and unauthorized access. The hardware manufacturer guarantees that no one — not even the system administrator or the operating system — has access to the keys or can read the memory stored within the TEE. This makes it a great choice for executing confidential tasks. Check here for more information about the [TEE](#).

The IMP provides a set of functionalities that enable users to create, manage, and revoke identities, as well as perform various other operations related to identity management. With the Identity Management Pallet, users can easily manage their identities, control access to their data, and ensure the security and integrity of their information. This technology transforms users' identities, enabling individuals and organizations to take greater control of their digital identities in a more secure and decentralized manner.



Assertion definitions and parameters

Previous

Next

Components of the IMP



# Components of the IMP

This section explains the different components of the Identity Management Pallet

Since the Identity management pallet directly interacts with the front end, it is made up of different components. These components include:

## Identity Struct

An Identity Struct is a JSON data structure that contains information about an individual's identity. This data structure is used to represent and manage the attributes and characteristics of a user's digital identity within the hub.

The IdentityHub supports linking, unlinking, and verifying multiple identities in batches to reduce the number of needed calls. It also supports linking multiple identities to one litentry account.

In order to achieve this, multiple fields are packed into one single `identity` struct and multiple `identities` as an array in the `identity` field. A typical example of a single identity struct in the pallet is shown below:

```
{
  "type": "..",
  "address": "..",
  "webtype": "..",
  "metadata": "..",
  "needVerification": "..",
  "web2ValidationData": {
    "link": ".."
  },
  "web3ValidationData": {
    "message": "..",
    "signature": "..",
    "timestamp": ".."
  }
}
```

Only `identity type` and `address` in the fields are required, other fields are optional:

`type` - Types of identities to be linked, e.g. Twitter, GitHub, discord, substrate, EVM

`address` - concrete address or handle, e.g. `0x1234`, `twitterHandle`

`webtype` - `web2` or `web3`

`metadata` - placeholder for potential metadata, it makes it more extensible as well.

`needVerification` - By default, the optional fields are false and only used when linking the identities. if the stated identity type and address need verification. When it's false, the given identity information will still be stored in TEE, but marked as "untrusted/unverified"; when it's true, a challenge code will be generated and broadcasted as events for further verification.

`web2ValidationData` - This is only used when verifying web2 identities e.g. the URL `link` to the public post where an encrypted message is included.

`web3ValidationData` - This is only used when verifying web3 identities, e.g calling extrinsics.

## ID Graph

An identity Graph is a data structure that represents the relationships between different identities that belong to the same individual. It represents the relationship between a user's different accounts and can be used to map out a user's aggregated identity through any of the associated identity data. The IdentityHub aims to provide the tool to generate an identity graph for the use of generating a user's verifiable identity data for both Web2 and Web3 data.

At its core, it is a collection of information that links together all of the different identities that belong to a single individual. This information includes the identity information in the `Identity Struct` such as identity type, address, web2 and, web3 data, and other identifying information. By linking all of this information together, an Identity Graph provides a comprehensive view of an individual's digital identity.

An identity graph is retrieved from multiple associated identity pairs with verifiable identity verification proofs. An identity pair is a pairing of two web3 addresses or a pairing of a web3 address and a web2 account, it proves the joint ownership of the two associated accounts.

### Data structure

An ID graph is composed of a list of the Web3 and Web2 accounts owned by the owner and their corresponding proofs.

### Identity pair

Each ID graph is extracted by ID pairs. An ID pair is made of two decentralized verifiable ownership claims. Each ID pair claims the joint ownership of two accounts, it can be a pairing of two web3 addresses or a pairing of a web3 address and a web2 account. Everyone can verify and trust the ID pair.

### Merging

An ID pair is the smallest combination of an ID graph. For ID graphs that have a common address, one ID graph will be merged into the other ID graph and keep only one.

## Shielding Key

A shielding key is a 256-bit AES-GCM cryptographic key pair that is generated randomly and protected by the user password. Its public key is used by the Litentry TEE worker to encrypt user-sensitive information when passing back the data. It is used to encrypt all the data in the communication between the user and the Litentry Parachain. The shielding key is generated in the user's local environment and used by the IDHLS to isolate sensitive user data from the IDH server and all other third parties.

Overall, it is an extra layer of security on top of your Substrate private key (Account) used exclusively to encrypt your data for transmission and ensure only you or the Enclaves can decrypt it.

It is important to note that there are two types of shielding keys;

User shielding key - This key is applied to the on-chain data returned by TEE. TEE shielding key- is used in the other direction (user -> TEE) and is publicly visible.

# IMP Extrinsic, Events and Error Events

## Extrinsic (called by user/extension)

IMP extrinsics refer to the set of instructions that can be executed on the parachain to update the IdentityHub. These extrinsics can include actions such as creating a new identity, linking a new identity, updating existing identity attributes, revoking access to an identity, or delegating identity management privileges to another user or entity. These extrinsics are called by the user or the IDhub and they are encrypted with a TEE shielding key. Below are the Litentry IMP extrinsics and their callers:

- `linkIdentity([payload])` - Request to link the given identities to the call origin (user). The payload is forwarded and packed in a trusted call of `call_worker` extrinsic in pallet-teerex.
- `unlinkIdentity([payload])` - Request to unlink(delete) the given identities from the call origin. Here, the payload is serialized JSON bytes.
- `verifyIdentity([payload])` - Request to verify the identities using the included validation data (within the identity struct). The payload here is also serialized JSON bytes.

No events will be emitted for these extrinsics, or something like `callForwarded()` at its maximum because the real business logic will be handled inside TEE which is carried out asynchronously.

## IMP Events

An Identity management pallet event is a message that is emitted by the pallet when a specific action related to identity management is performed on the pallet. An event may be triggered when a user creates a new identity, updates their identity attributes, or revokes access to an identity. These events typically include information such as the identity account involved, the type of action performed, and any associated metadata or attributes. The event emissions are triggered by the pub fn which is called by the pallet-teerex/enclave. The events are encrypted with the user's shielding key.

- `UserShieldingKeySet` - user shielding key is set { `who: T::AccountId`, `key: UserShieldingKeyType` }
- `ChallengeCodeSet` - challenge code is set { `who: T::AccountId`, `identity: Identity`, `code: ChallengeCode` }
- `ChallengeCodeRemoved` - challenge code is removed { `who: T::AccountId`, `identity: Identity` }
- `IdentityCreated` - an identity is created { `who: T::AccountId`, `identity: Identity` }
- `IdentityRemoved` - an identity was removed { `who: T::AccountId`, `identity: Identity` }

## IMP Error Events

Pallet error events are messages or notifications that are emitted by the pallet when errors or exceptions occur during the execution of an identity management action. Error events typically include information such as the type of error, the identity account or action involved, and any associated metadata or attributes.

For example, an Identity management pallet error event may be emitted if a user attempts to update their identity attributes with invalid or unauthorized information or if an identity already exists.

- `ChallengeCodeNotExist` - challenge code doesn't exist
- `IdentityAlreadyVerified` - the pair (litentry-account, identity) already verified when creating an identity
- `IdentityNotExist` - the pair (litentry-account, identity) doesn't exist
- `IdentityNotCreated` - the identity was not created before the verification
- `IdentityShouldBeDisallowed` - the identity should be disallowed
- `VerificationRequestTooEarly` - a verification request comes too early
- `VerificationRequestTooLate` - a verification request comes too late
- `RemovePrimeIdentityDisallowed` - remove prime identity should be disallowed



Previous  
Components of the IMP

Next

Teerex Pallet



# Teerex Pallet



## Introduction

Teerex pallet is the remote attestation registry and verification pallet for integritee blockchains and parachains. Attestation is the means for a remote user to ascertain that an application runs on real hardware in an updated Trusted Execution Environment (TEE) with the expected initial state. It proves the trustworthiness of the SGX enclave and remote attestation is defined when a user certifies a TEE running on a remote physical machine.

The Teerex pallet is a pallet for Integritee that acts as a verified registry for SGX enclaves. Its goal is to provide public auditability of remote attestation of SGX enclaves. Given deterministic builds of enclave code, this pallet closes the trust gap from source code to the MRENCLAVE of an enclave running on a genuine Intel SGX platform.

Without the need for a license with Intel, everyone can verify what code is executed by registered service providers and that it is executed with confidentiality. Since Litentry integrates this pallet, it, therefore, acts as a public registry of remote-attested services.

The Teerex pallet also supports the upgrade of enclaves. A new enclave can be upgraded at a specified height while preserving the integrity of the original data. Apart from this, it supports Enhanced Privacy Identifier (EPID) and Data Center Attestation Primitives (DCAP) remote attestation.

EPID is the attestation protocol originally shipped with SGX where the user application runs in an SGX enclave on a remote untrusted machine whereas the end user waits for the attestation evidence from this enclave on a trusted machine. The DCAP is a software infrastructure provided by Intel as a reference implementation for remote attestation. It is a special SDK that allows for launching enclaves with Intel's remote infrastructure and it is backed by the DCAP-enabled SGX driver.

The pallet also functions as an indirect-invocation proxy for calls to the confidential state transition function executed in SGX enclaves off-chain.

Overall, the Teerex pallet is mainly responsible for packing the payload into `trustedCall` in `call_worker` e.g calling `link_eth`, which is extrinsic from the SGX account linker pallet. Or query the encrypted data in SGX.

For each extrinsic call in the Identity Management Pallet (**IMP**), a `trustedCall` needs to be constructed with the caller's address and payload, e.g.

```
call_worker([TrustedCallSigned::
```

The `caller-address` is the caller's parachain account address, it's:

- used as the primary key for storing ID-graph in TEE
- used to verify the `web2/web3validationData` as the correct owner when verifying identities

It is important to note that `payload` is already encrypted, so it's double-encrypted (but with the same TEE's shielding key)

The IMP has `extrinsics/pub` functions that only serve the purpose of broadcasting events. For every such method, an extrinsic in the Teerex pallet is required to call it. This extrinsic should have identical parameters and internally it only calls its counterpart in the IMP. However, it should only allow privileged origin, which is the enclave signing origin.

This means that only the enclave calls these methods even if they are declared as "extrinsics". These extrinsics include:

- `codeGenerated(<user-account>, <code>)`
- `identityLinked(<user-account>, <identity-type>, <identity-address>)`
- `identityUnlinked(<user-account>, <identity-type>, <identity-address>)`
- `identityVerified(<user-account>, <identity-type>, <identity-address>)`

In conclusion, the Teerex pallet in Parachain enables TEE workers to *register*, *discover*, and *communicate* with one another. Its main features are:

- Acting as a verified registry that allows remote verification of SGX enclaves — providing public auditability.
- Designed with confidentiality at its core to bridge the trust gap between the enclave and enable anyone to verify the codes that are executed.
- Acting as an indirect proxy for off-chain confidential state transition calls executed by SGX enclaves.
- Supports upgrade of enclaves
- Supports EPID and DCAP remote attestations.



IMP Extrinsic, Events and Error Events

Previous

Next

Token Bridge





# Make your identity tangible without being seen


## Token Bridge

Entry page for token bridge

### Overview


The [token bridge](#) is deployed by the Litentry team to allow the transfer of LIT tokens between **Ethereum** (ERC20 tokens) and **Litmus/Litentry parachains**.


### Outlines

 [Tokenomics revisit](#)

 [Architecture model](#)

 [Token Bridge Commission](#)

 [Previous Teerex Pallet](#)

[Next Tokenomics revisit](#) 

# Tokenomics revisit

This page revisits the tokenomics of LIT

**LIT** token is the native cryptocurrency of the Litentry network and is issued by the Litentry Foundation. Litentry tokens LIT would be the driving force in the circulation in the DID ecosystem. LIT is currently issued as ERC-20 token and BEP-20 token. [The Binance project research about Litentry](#) describes the LIT tokenomics and allocation.

After Litentry launches a parachain(Litmus) on the Kusama network, Litmus will have LIT natively, alongside the rest on the Ethereum blockchain, which means partial LIT token migration will be executed.

We'd like to revisit the LIT tokenomics:

- Litentry's economic model and initial supply will be retained (**100,000,000 LIT**).
- Litentry plans to burn a portion of ERC-20 LIT and issue the same amount of native LIT on the parachain to keep the initial supply unchanged.
- Litentry will launch a token bridge between ERC-20 LIT and Litmus LIT, ERC-20 LIT holders can decide whether to migrate tokens to Litmus or not.
- The migration ratios shall be **1 ERC-20 LIT = 1 Litmus LIT**.

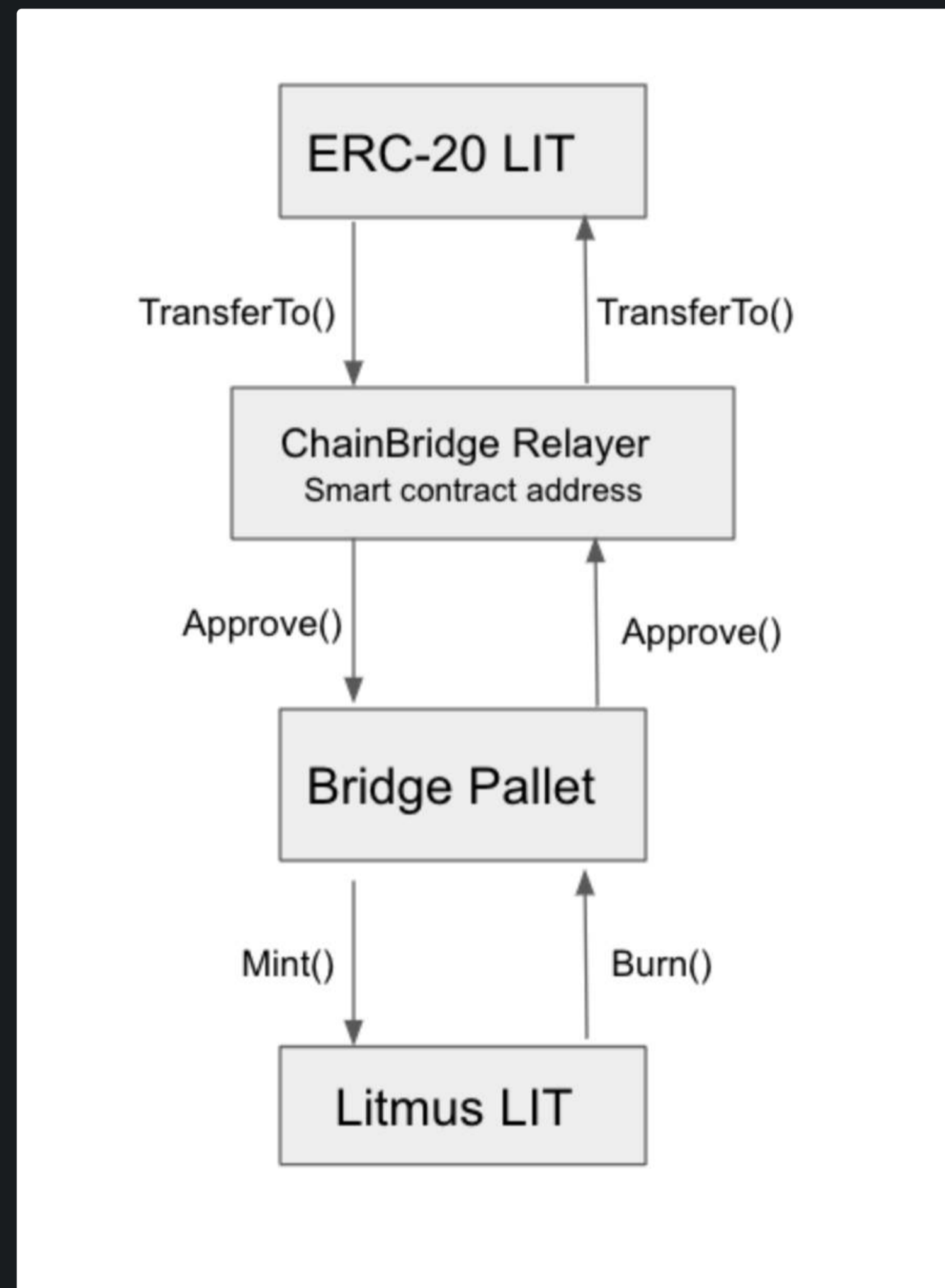
← Previous **Token Bridge** Next **Architecture model** →

## Architecture model

Since the LIT token was already issued as ERC-20 token and BEP-20 token at the beginning of 2021, a token bridge is required to transfer LIT token across different chains: between Ethereum and Litmus parachain in this case.

To achieve it, Litmus takes use of the [ChainBridge](#) solution, a modular multi-directional blockchain bridge built by [ChainSafe](#) and designed to interact with multiple networks including Ethereum, Ethereum Classic and Substrate-based chains.

The workflow of the token bridge is demonstrated in the following diagram:



workflow of token bridge

In the diagram above,

- **ChainBridge relayer** contains a bridge smart contract that is deployed by the official Litentry team on the Ethereum network.
- **Bridge pallet** is a pallet configured in the Litmus runtime. It will verify the message from the relayer and mint tokens if required.

Imagine Alice wants to transfer LIT tokens from ERC20 on Ethereum to Litmus parachain:

- Alice calls the smart contract in the ChainBridge relayer with her Litmus parachain address and the number of LIT tokens she wants to transfer.
- Alice transfers the desired amount of LIT tokens to the specified smart contract address
- After successful verification of the transaction above, the ChainBridge Relayer issues an acknowledgment to Bridge-Pallet on Litmus parachain, with all the needed information as parameters.
- Bridge-pallet gets the approval message and mints the same amount of LIT tokens to Alice provided address on parachain.

More technical details will be added later.

We also added a [step-by-step guide](#) to illustrate the whole process.

# Token Bridge Commission

This treatise explains the Litentry crosschain bridge token commission fee to give our users an understanding of how it works.

## Introduction

Cross-chain bridge serves as a neutral zone that enables users to deploy assets for fast and easy transactions irrespective of the blockchain network. It also reduces operational difficulties by allowing access to multiple blockchains through the same network.

Litentry has developed a Crosschain Bridge that works in the same way and all its components are built in a standardized fashion to help unlock the superior liquidity in the DOTSAMA ecosystem. The crosschain bridge allows users to seamlessly send and receive LIT tokens between Polkadot and Kusama (Parachain LIT) and the Ethereum network (ERC-20 LIT).

To achieve this, Litentry uses a ChainBridge Relay model that calls smart contract functions based on the transaction direction (LIT  $\rightleftharpoons$  ERC-20 LIT) and allows the Bridge Pallet to mint or burn LIT for the appropriate transaction to be carried out.

## Bridge Transfer Fee Calculation(Parachain LIT $\rightarrow$ Erc20-LIT)

For every Bridge transaction that occurs within the Litentry protocol, there are three ChainBridge relayers that are responsible for the approval or rejection of the transaction.

<b>Relayer node</b>	Litmus token bridge eth account
prod1	0xd6f10585d96458cf4106a9d02467d7d74fc23714
prod2	0x8229E43EcB9b215Bff71eC22D3BE0F9208fB976e
prod3	0xCC78B88248DFFd80C0C5B872114C2bb84AD8Ddd7

Litentry Relay Network/Address Info

We use 3 relay servers to submit and execute proposals for smart contracts and a proposal needs to get 2/3 votes to get executed. The reason for it is to prevent a single point of failure and minimize the risk of hacking and malicious attacks

So basically cross chain transactions from parachain(both Litmus and Litentry) to ERC20-LIT would result in 6 transactions on Ethereum, that's where the commission comes from. For every transaction, there are **six relayer executions** and at least two of the relayers must successfully execute the following two triggers (**a total of 4 executions**) for a transaction to be approved:

- **Vote Proposal**
- **Execute Proposal**

While the opposite is the case for failed transactions.

The remaining two triggers (**to make 6 executions**) are:

- **Failed Vote Proposal**
- **Failed Execute Proposal**

It is important to note that the smart contract charges different fees for each of the triggers executed by the Relayers as evident below:

Title	Cost	Execution status
Initial Gas Price on Ethereum	25 Gwei	
Success VoteProposal Fee	0.002552 ETH	2 times
Failed VoteProposal Fee	0.0005425 ETH	1 times
Success ExecuteProposal Fee	0.00123 ETH	1 times
Failed ExecuteProposal Fee	0.000559ETH	2 times
LIT/ETH Ratio	0.000773 LIT/ETH	
Fee based on LIT	10.34253 LIT	

The Commission Fee is calculated based on the following methods:

$$\text{CommissionFee} = (\text{SuccessVoteProposalFee} \times 2 + \text{FailedVoteProposalFee} + \text{SuccessExecutePropo})$$

The total fee charged by the smart contract for each of the Relayer triggers is then divided by the LIT/ETH ratio provided by Binance which amounts to approximately 10 LIT for Crosschain Bridge transactions.

**⚠ IMPORTANT: Token Bridge Commission Fee Varies with Network Congestion ⚠**

Our Token Bridge commission may vary based on network congestion. We strive to keep it reasonable and transparent. You may want to plan transactions during low-activity periods for smoother transfers.

# Identity is fragmented.

## TEE

Entry page for TEE

### Overview

A **Trusted Execution Environment** (TEE) is an environment for executing code, in which those executing the code can have high levels of trust in the asset management of that surrounding environment because it can ignore threats from the “unknown” rest of the device.

Trusted applications running in a TEE have access to the full power of a device's main processor and memory, whereas hardware isolation protects these components from user installed applications running in the main operating system. Software and cryptographic isolations inside the TEE protect the different contained trusted applications from each other.

To achieve identity aggregation, Litentry has a requirement of storing sensitive user data, like user's Ethereum account and computed credit score. And the TEE has been chosen as a fundamental approach to guarantee the security of data storage and data processing. Litentry builds a side chain, which is composed of multiple TEE equipped nodes, to make sure of storing and processing data in a distributed way, without exposing user's private data.


This chapter:


- introduces the background knowledge of TEE
- explains the architecture of Litentry solution via diagrams
- depicts the core components that are involved in the architecture
- provides a guide to building and executing code examples (WIP)

### Outlines:

 [Background](#)

 [TEE - FAQ](#)

 [Architecture diagram](#)

 [Core components](#)

← [Token Bridge Commission](#)

[Background](#) →

# Background ⋮

Brief introduction to TEE background

To perform the account linking from various providers and an aggregated DID solution in a secure and privacy-preserving way, TEE is used to wrap around the core computational service.

**TEE** (trusted execution environment) is a secure area of the main processor which guarantees code and data loaded inside to be protected with respect to confidentiality and integrity. In Litentry, TEE also represents the whole solution for data protection in parachain, the encrypted data processing in the side chain and all Dapps.

The most important technology stacks among them are:

- [Intel SGX](#) (Software Guard Extension)  
A new instruction set in Skylake Intel CPUs since autumn 2015. Every node in the side chain must support SGX
- [Rust SDK](#)  
The TEE device is bare metal, so the SDK is different from SDK based on OS, which provides services like a system-level library, file system and so on. A dedicated rust SDK was provided by rust community and now it is open source and becomes the Apache incubated project.
- [Substrate](#)  
A next-generation framework for blockchain innovation. Substrate takes a modular approach to blockchain development and defines a rich set of primitives that allows developers to make use of powerful, familiar programming idioms.
- [Integritee](#)  
The most scalable public blockchain solution for securely processing sensitive business or personal data. It harnesses the speed and confidentiality of trusted execution environments, combined with the trust of a decentralized network.

In the next chapter, we will present the overarching architecture and workflow for the Litentry solution.

← Previous **TEE** Next **TEE - FAQ** →

# TEE - FAQ ⋮

Common questions about the use of Trusted Execution Environments

## Why does Litentry Use TEE's?

To allow users to aggregate their fragmented identity, Litentry has a requirement of storing sensitive user data, such as a user's Polkadot or Ethereum account, Twitter account and credit scores. Trusted Execution Environments have been chosen as a fundamental approach to guarantee the security of data storage and data processing.

## What is a TEE?

A Trusted Execution Environment is a secure area or enclave on a computer's processor, separate from the main operating system. It stores and processes data with complete integrity and protects data from any possible tampering from the outside. Computation within a TEE is totally invisible from the outside.

## How does a TEE protect data privacy?

Litentry provides these closed environments or enclaves (TEE's) in which only the user has control over their data and sharing authorisations. Besides the specific hardware design of a TEE, any input and output, such as a user's sensitive account relationships or their credentials, are encrypted with cryptographic keys.

## How does a TEE create trust?

The trusted execution environment is known as a secured machine running a known piece of open source code. Everyone can verify the TEE's functionality and results. It functions as an independent 3th party. Everyone can also verify that the inputs and outputs of a TEE act according to the open source code.

## How do you make sure that a TEE can be trusted?

Every TEE goes through an attestation process to ensure the code is running on a genuine and secure TEE from the hardware manufacturer. This process verifies the TEE's code is untampered and verifies the dedicated cryptographic keypair of the specific TEE. These keys allow the TEE to sign its own messages as a means of verification that a specific credential was issued by a specific TEE.

## How does Litentry use's TEE's?

Litentry uses Trusted Execution Enclaves to protect the sensitive relations of identity owner. A user can store & communicate the relationships between their web2 & Web3 accounts safely since it is protected by the TEE and encrypted during communication. Our TEE's also verify the on-chain information that is related to those accounts as an independent trustworthy observer and help users practice selective disclosure of their credentials.

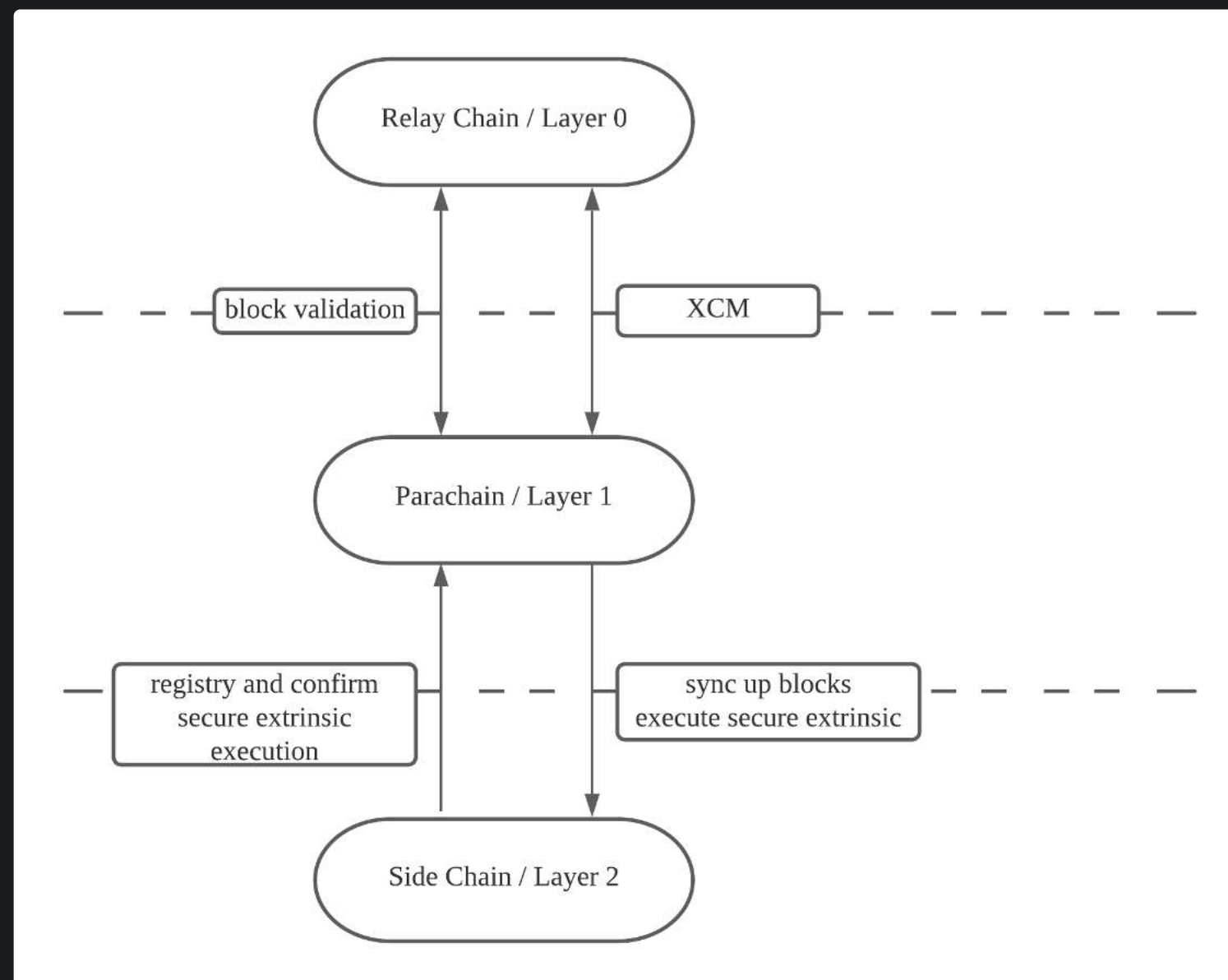
## How does the TEE allow for 'Selective Disclosure' of credentials?

Since the TEE acts as an independent trustworthy and verifiable observer it can issue claims and credentials about the accounts being stored inside its enclave. This privacy preserving middle layer allows the user to manage the amount of information they select to disclose or allows to share. A user might prefer to share the possession of a token but not its amount or purchase date.

← [Previous Background](#) [Next Architecture diagram](#) →

# Architecture diagram

A diagram representation of the main architecture



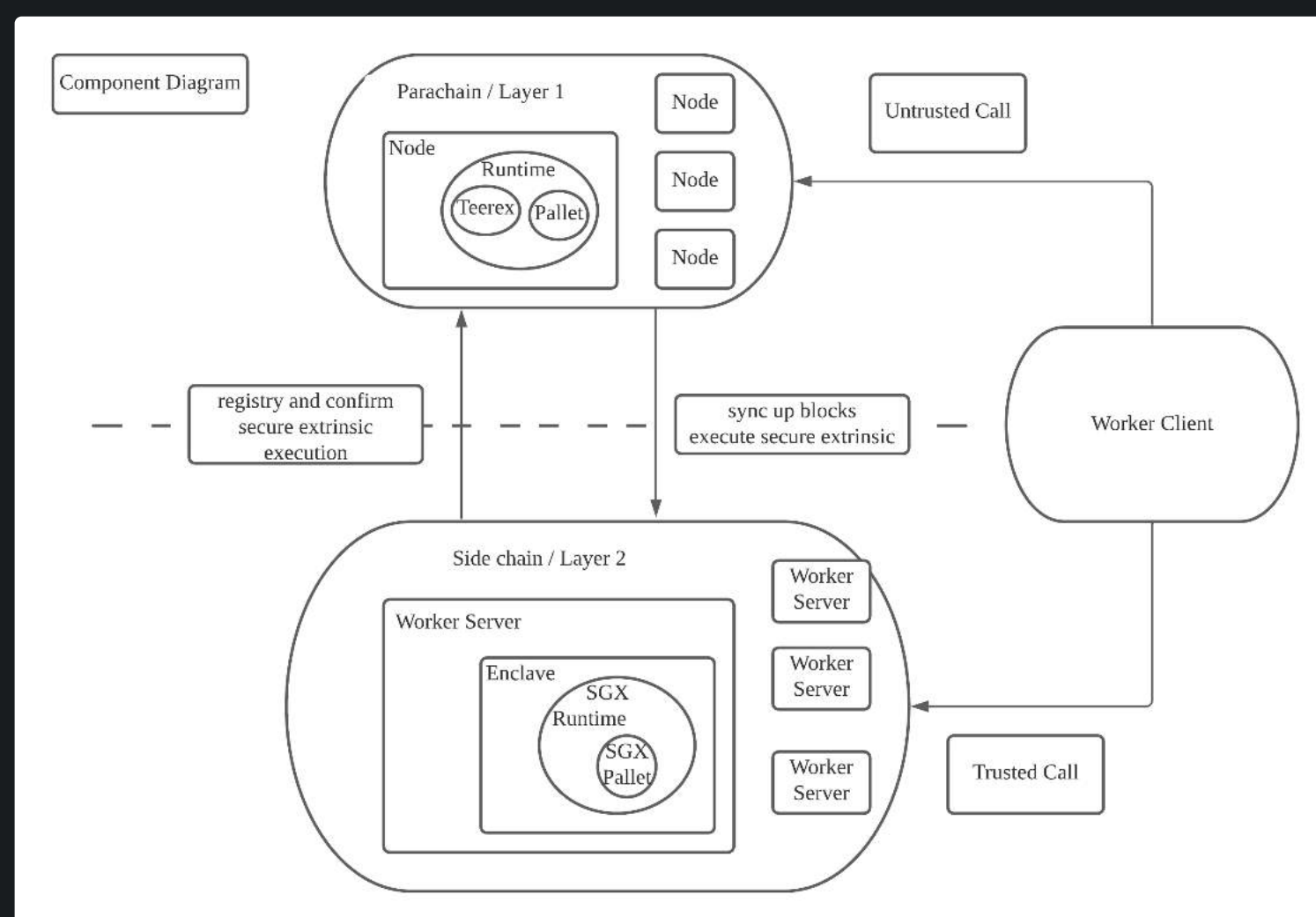
overall architecture

In the diagram, there are three layers of blockchains, each layer has different responsibilities:

- Layer 0** - The main net of relay chains, for example Kusama or Polkadot. It is responsible for providing the shared security of the whole relay chain and parachain network, also works as a router for XCM messages.
- Layer 1** - Litmus or Litentry parachain. In this scenario, it serves as an application-specific blockchain, which connects to the relay chain by fitting itself into the parachain slot. The relay chain will validate its blocks and handle the XCM messages.
- Layer 2** - The TEE side chain that is supported by Integritee. It provides an environment for the runtime to be executed in the SGX (secure run environment). It deviates from the Layer 1 parachain where every state and extrinsic are public and known.

The most magical part of the architecture is that all three layers of blockchain are all based on the Substrate framework. By adapting to different runtime configurations and execution logics, the customized blockchain can play different roles.

There are 5 major software components, **SGX runtime**, **teerex pallet**, application-specific pallets or **sgx pallets**, **worker client** and **worker server**. Their interaction is shown below in the component diagram:



component diagram in TEE



# Make your identity tangible without being seen

## Core components

The chapter elaborates on all the important components in the previous diagram and as well as features we have implemented so far:

- [Pallets in parachain](#)
- [SGX runtime](#)
- [Worker client](#)
- [Worker server](#)

### Pallets in parachain

#### Teerex pallet

To communicate with the side chain, the parachain must include this pallet. Teerex has two important interfaces.

- verify Intel's SGX node and accept the registration if verification is done.
- get the request to call some extrinsic in SGX runtime in SGX. call `sgx-runtime` extrinsic includes a parameter, the type is `Call` that could be any byte code encoded with pallet id, extrinsic index, and parameters from SGX runtime.

Note: In the development environment, we can skip the verification via a compilation feature:

```
cargo build --features skip-ias-check
```

#### Credit score pallet

Litentry provides the service to compute customer's credit score based on the linked accounts and data related to these accounts. All accounts are stored in the SGX, the data fetch and score computing happened in SGX to avoid information leakage. This pallet is responsible to get the score from SGX side chain, the result may be encrypted by customer's public key, or used by Dapp based on Litentry SDK.

#### SGX runtime

We define the runtime executed in the SGX in this repo. SGX runtime is similar to Substrate runtime, composed of pallets. The runtime can be compiled to a WASM blob or binary. The difference is SGX runtime depends on `tee-sgx-sdk` as mentioned before.

For Substrate runtime execution, we need two packages. First one is `sp-externalities` which provide the execution environment. Another one is `sp-io` for runtime to access the db, file systems, blockchain state and so on. Because SGX runtime is executed in the SGX, so both packages from Substrate not applicable in the SGX.

There are two packages from SGX runtime [repo](#), are `sgx-runtime/sp-io` and `sgx-runtime/sp-externalities`, with the same name `sp-io` and `sp-externalities`. They make it possible for runtime executed in the SGX.

#### SGX pallets

If the pallet includes some privacy data that need to be stored in the SGX node, we should put it in the SGX runtime. For example, Litentry first pallet `account linker`, the users links their Ethereum or Bitcoin addresses via `link-eth` and `link-btc` extrinsic. It may expose all these privacy data if we put the `account linker` in parachain.

So we put the `account linker` pallet into SGX runtime now, the code is not changed. Then all extrinsic parameters and linked addresses will be encrypted in the parachain, only SGX nodes know the private key, decrypts the data in the SGX, and dispatch the call to SGX runtime.

In the future, we will migrate more pallets from parachain to side chain.

#### Worker client

The client interacts with both parachain and side chain via RPC / WSS connections. According to destination, we have different call types.

- Untrusted call: client interacts with parachain node, it is similar to js client. Client includes some default pallets like `teerex`, we can use untrusted call to send transactions or some queries. For example, a user can call `balance transfer` via the client, it is an untrusted call.
- Trust call: client interacts with worker server, which is counterpart to parachain node. The server also provides the RPC service. For example, a user can call `link_eth`, which is a extrinsic from `sgx account linker` pallet that is part of SGX runtime. Or query the encrypted data in SGX.
- Direct call: client can call the extrinsic defined in the SGX runtime. the same use cases as trust call.
- Indirect call: client wrappers the SGX runtime call, then send it to `teerex` pallet in parachain. worker node sync up the blocks from parachain, then identify the specific `call_work` extrinsic, parse the call from parameter and dispatch it to SGX runtime. The details could be found in the diagram in Litentry solution section.

#### Worker server

The server is the most complicated part of whole solution, the major features are as follows:

- get the verification report from Intel verification service and register itself to parachain
- provide the execution environment for SGX runtime in the trusted node
- sync up the blocks from parachain, decrypt and parse the data from `call_work`
- generate side chain's block, sync up and consensus between nodes
- provide RPC and WSS service
- send the response to parachain via extrinsic

From a software running point of view, the server has a boundary between trusted and untrusted parts. Trusted code is executed in the SGX. Untrusted code includes start-up the process, RPC server, initializing the enclave, and so on.

The sharding is supported from the beginning of side chain design. The server node joins one shard, each call both direct and indirect has a default parameter `shard` identity. The server node just execute the call with the same shard that it joined. the benefit of shard is as follows

- the state for the different shard is isolated, the different shard nodes can't see each other private data
- shard nodes can skip the call from another shard, it saves the resource, makes it faster to execute less extrinsic in the block
- sharding make it possible to use our solution at a large scale, at the same time protecting the data

## Litentry Identity Registrar

Explanation of Litentry (Kusama) identity registrar - a service that leverages cryptographic design to provide judgment on user identity while preserving privacy.

### Registrar Background

Decentralized Identity (DID) consists of a triangle of Trust. The attester (registrar), the credential holder, and the verifier. The attester (verifies/attests/judges) that the credentials for the identity are correct. Credentials could be email addresses, Twitter accounts, or any other data that could be used as identity credentials. The credential holder is the person, business, or IoT device that controls the credentials. The verifier is the entity to whom the credentials are being presented.

For example, if you want to use a Dapp or service and they would like to verify that you are identified by your email address and/or Twitter account. They could ask for these credentials. As a user that needs to use this service, you could choose to hand these credentials over to them. Because the credentials are attested by a trusted attester, the verifier will accept the presented credentials and allow you to use the Dapp or service.

Litentry is a registrar in the Polkadot ecosystem (an attester in self-sovereign identity). This means that Litentry can attest to (issue a judgment) the validity of your email address, Twitter account, and any other identifying information. This section covers How to verify your identity with Litentry and the implementation details of Litentry.

### Litentry and Polkadot

Polkadot provides a service that allows participants to add personal information such as email addresses and Twitter accounts to their on-chain accounts. The user can then ask for verification of this information by a registrar. Litentry is a registrar. A user can request a registrar to make a judgment on their claims. The user can select a fee that they are willing to pay. Registrars are accepted by submitting proposals to the democratic process in Polkadot and Litentry has been accepted as a Registrar for Kusama.

### Implementation Details

GitHub Repository: <https://github.com/litentry/litentry-registrar>

This document highlights the implementation details of how Litentry works as a Polkadot/Kusama registrar.

### Context

When a user has set their identity in Polkadot or Kusama they can request judgment from a registrar. In Polkadot, the registrar can support up to six levels of confidence in their attestation. At the moment Litentry supports four judgment levels and in the future, we would like to support `KnownGood` by integrating with well-known KYC organizations in the future. The `LowQuality` level will never be supported by Litentry.

In the implementation details section, the Litentry Registrar Architecture consists of Validators, Event Listener, ProvideJudgementService, and Database Service. We also introduce a secure method using JWT (JSON Web Token) to construct the verification protocol.

### Judgment Levels & Criteria

Registrars on Kusama can provide their judgment according to six levels of confidence for users' identity:

- `Unknown`: The default value, no judgment made yet.
- `Reasonable`: The data appears reasonable, but no in-depth checks (e.g. formal KYC process) were performed.
- `KnownGood`: The registrar has certified that the information is correct.
- `OutOfDate`: The information used to be good, but is now outdated.
- `LowQuality`: The information is low quality or imprecise, but can be fixed with an update.
- `Erroneous`: The information is erroneous and may indicate malicious intent.

There is another temporary confidence level used by Polkadot/Kusama.

- `FeePaid`: The judgment has been requested by a user and the information verification is in progress.

In Litentry we add additional clarification for each of these levels.

`Reasonable`: If a user's `displayName`, `email`, and `twitter` identity are verified. Litentry will update the user's identity as `Reasonable`.

`OutOfDate`: Litentry registrar keeps track of the user's identity to see whether it's out of date or not updated regularly enough. If a user doesn't update his identity timely, his identity will degrade to `OutOfDate`.

`Erroneous`: If any attempt is made by the user to attack the Litentry registrar, e.g. DDOS attack, the Litentry registrar will provide judgment with `Erroneous` and refuse to provide a new judgment for him in a specific period.

`LowQuality`: The Litentry registrar will never provide a judgment of `LowQuality`.

The Litentry registrar will automatically provide hints to guide the user to update his identity. After all the information is correctly verified, the user will receive a `Reasonable` judgment. In this way, a user can not only save fees (since we only provide one judgment for him) but also save time (since the Litentry registrar will point out an imprecise or low-quality identity as and when it is captured).

At the current phase, the Litentry registrar does not support providing a `KnownGood` judgment level since this would require the cooperation of third-party KYC services. We are working on partnering with the appropriate organization so that we can support this level in the future.

### Registrar Architecture

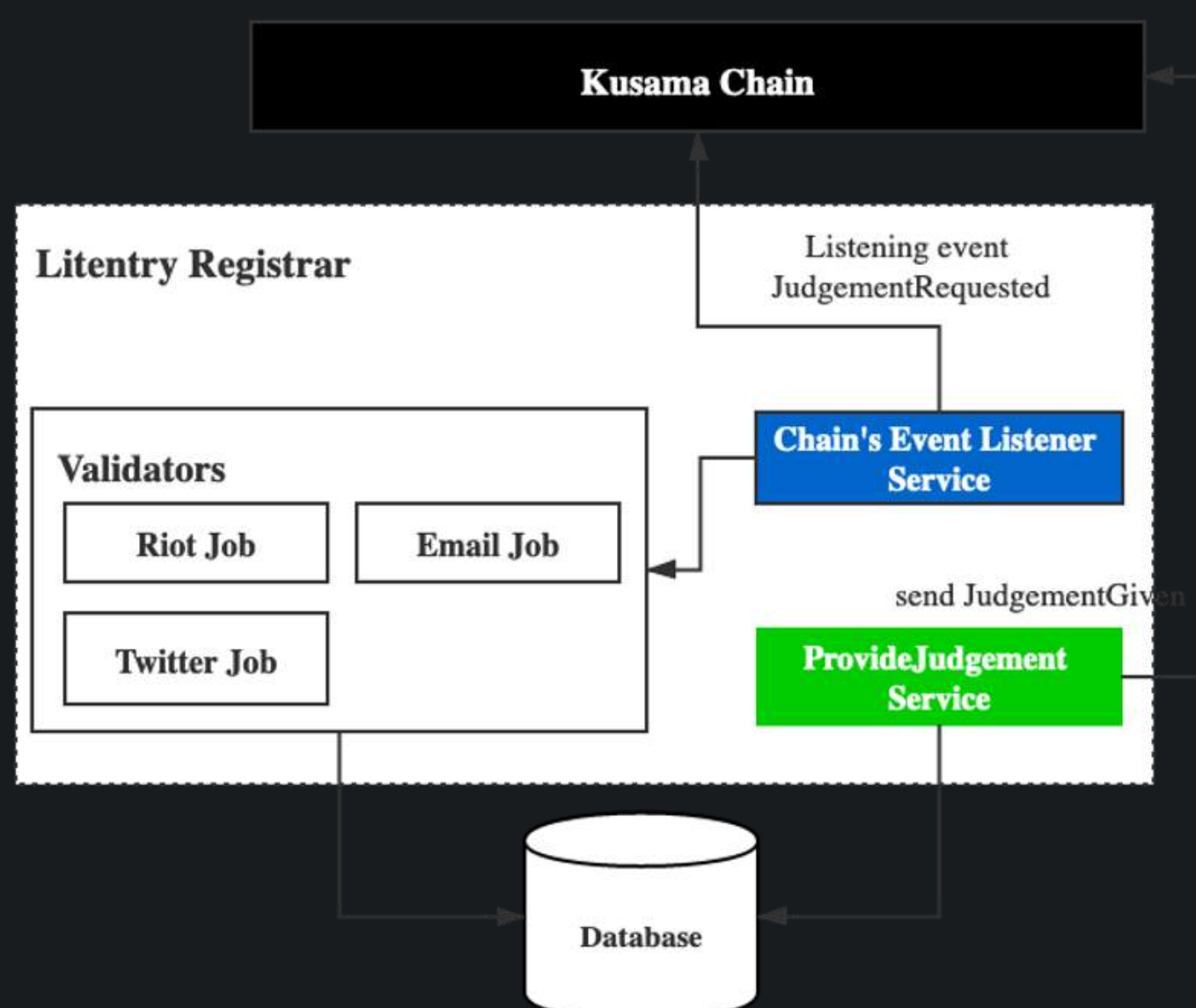


Figure 1.1 The Architecture of the Litentry Registrar

The Event Listener listens to all events coming from the Kusama chain. Once a `JudgementRequested` event is triggered on Kusama and the `JudgementRequested` indicates to use the Litentry registrar, the Event Listener service will invoke Validators starting the verification process.

At the current stage, the Validators consist of three verification services, Email, Element, and Twitter verification. After receiving the verification request from the Event Listener, the Validator will invoke those verification jobs. They will send a verification link to the users' provided accounts and wait for user confirmation from their accounts. The email address and Twitter account will be verified by Litentry by sending a challenge message that the user must pass to prove their control of the account. As soon as the user confirms all verification links, the `ProvideJudgement` service will complete the final step by providing judgment for the user. The implementation details will be introduced in the next section.

Once the user proves the ownership of the Email and Twitter accounts, the `ProvideJudgement` service will send a `JudgementGiven` transaction on Kusama to confirm the ownership of the accounts that the user provides.

The Database service will temporarily store users' data, e.g. Kusama account, email, and Twitter account so that we can recover services from an unpredictable crash. After completing the verification service, those data will be removed from the server permanently.

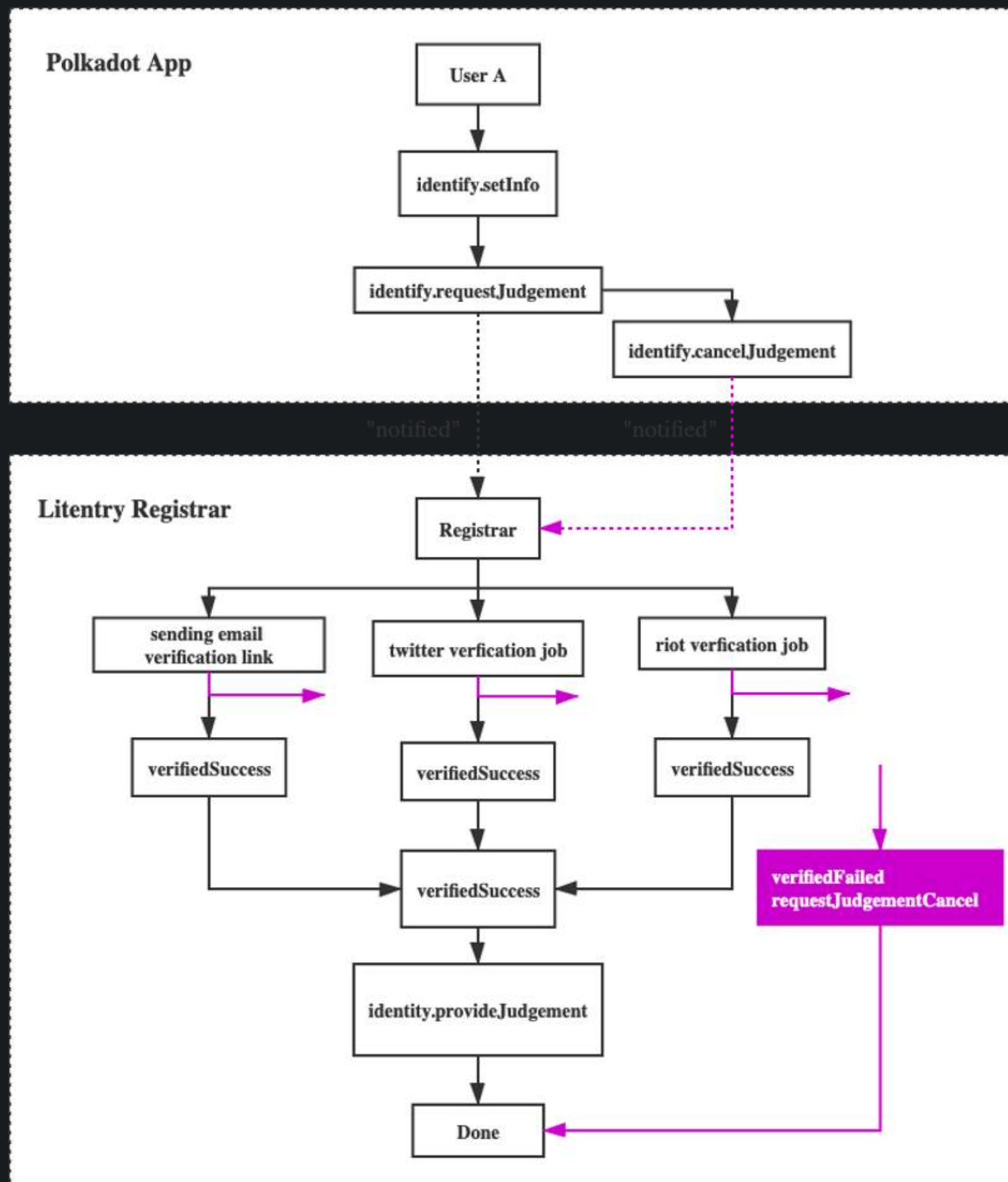


Figure 1.2 The main Workflow of the Verification process

### Security and Availability

We use JSON Web Token (JWT) to construct the verification protocol. A nonce and an `objectId` (from MongoDB) are used to generate the JWT token to ensure the security of the Litentry registrar. In this implementation, only the user who requests identity judgment, which implies his/her ownership of this Kusama account, will receive this encrypted token. Malicious users/her construct this token because of an unknown encryption secret, since nonce and `objectId` are encrypted. And the malicious user has no way to replay the attacks.

On the other hand, the `WebSocket` (TCP connection) can be easily reset by the remote peer due to long-time idle. In this situation, the events from Kusama would never be captured due to the disconnection between Kusama and Litentry. To prevent this situation, we capture the events from the underlying `WebSocket` connection and reconnect to the Kusama automatically whenever the connection is reset by a peer.

# Build parachain

How to build our parachain manually

Similar to polkadot, different chain-specs/runtimes are compiled into one single binary. In our case it's:

- [litmus-parachain-runtime](#) (on kusama)
- [litentry-parachain-runtime](#) (on polkadot)
- [rococo-parachain-runtime](#) (on rococo)

## build parachain manually

1. make sure [cargo](#) is installed, preferably via [rustup](#)
2. `git clone https://github.com/litentry/litentry-parachain`
3. `cd litentry-parachain`

To build the litentry-parachain **raw binary** manually:

```
make build-node
```

To build the `litentry/litentry-parachain` **docker image** manually:

```
make build-docker
```

To build the litentry-parachain runtime wasm manually:

```
make build-runtime-litentry
```

The wasms should be located under `target/release/wbuild/litentry-parachain-runtime/`

Similarly, use `make build-runtime-litmus` to build the litmus-parachain-runtime.

## pre-built docker images

You can find all the parachain images on the github [release page](#) or directly on [docker hub](#). Among them, `litentry/litentry-parachain:latest` is pushed by our [github CI](#) and should always contain the latest stable build from the [default branch](#).

## pre-built binary releases

You can find all the pre-built binary releases on the github [release page](#).

⚠ The binary releases are only built for x64 Linux platforms, to get a binary for other platform, you have compile it manually.



Parachain - Previous  
**Developer Documentation**

Next

**Launch a local network**



# Launch a local network

How to get our parachain running on localhost

A minimum local dev network consists of **2** relay chain nodes and **1** parachain node. To launch it, there're two ways (taking litentry as an example, litmus is similar):

## using docker images (preferred)

1. make sure **docker** and **yarn** are installed
2. `git clone https://github.com/litentry/litentry-parachain`
3. `cd litentry-parachain`
4. `make launch-docker-litentry`

Both `parity/polkadot` and `litentry/litentry-parachain` images will be pulled from upstream automatically. Additionally, `parachain-launch` will be installed and used to generate chain-specs and docker-compose files.

The generated files will be under `docker/generated-litentry/`.

Wait a while until you see such logs:

```
[+] Running 6/6
  :: Volume "generated-litentry_relaychain-alice"    Created
  :: Volume "generated-litentry_relaychain-bob"     Created
  :: Volume "generated-litentry_parachain-2013-0"   Created
  :: Container generated-litentry-parachain-2013-0-1 Started
  :: Container generated-litentry-relaychain-bob-1 Started
  :: Container generated-litentry-relaychain-alice-1 Started

-----
waiting for parachain to import blocks ...
parachain imported #1

-----
checking parachain block production ...
parachain produced #1, all good. Quit now
```

logs showing that the network is launched successfully

✔ Up to this point the network is launched successfully.

You should be able to connect to the chain websocket via browser:

- <https://polkadot.js.org/apps/?rpc=ws%3A%2F%2F127.0.0.1%3A9944#/explorer> for relay chains
- <https://polkadot.js.org/apps/?rpc=ws%3A%2F%2F127.0.0.1%3A9946#/explorer> for parachain

ⓘ When finished with the dev network, run `make clean-docker-litentry` to stop the docker containers and tidy things up

## using raw binaries

Only when option 1 doesn't work and you suspect the docker-image went wrong. In this case we could try to launch the dev network with raw binaries.

### On Linux host:

- you should have the locally compiled `./target/release/litentry-collator` binary.
- run `make launch-binary-litentry`

### On Non-Linux host:

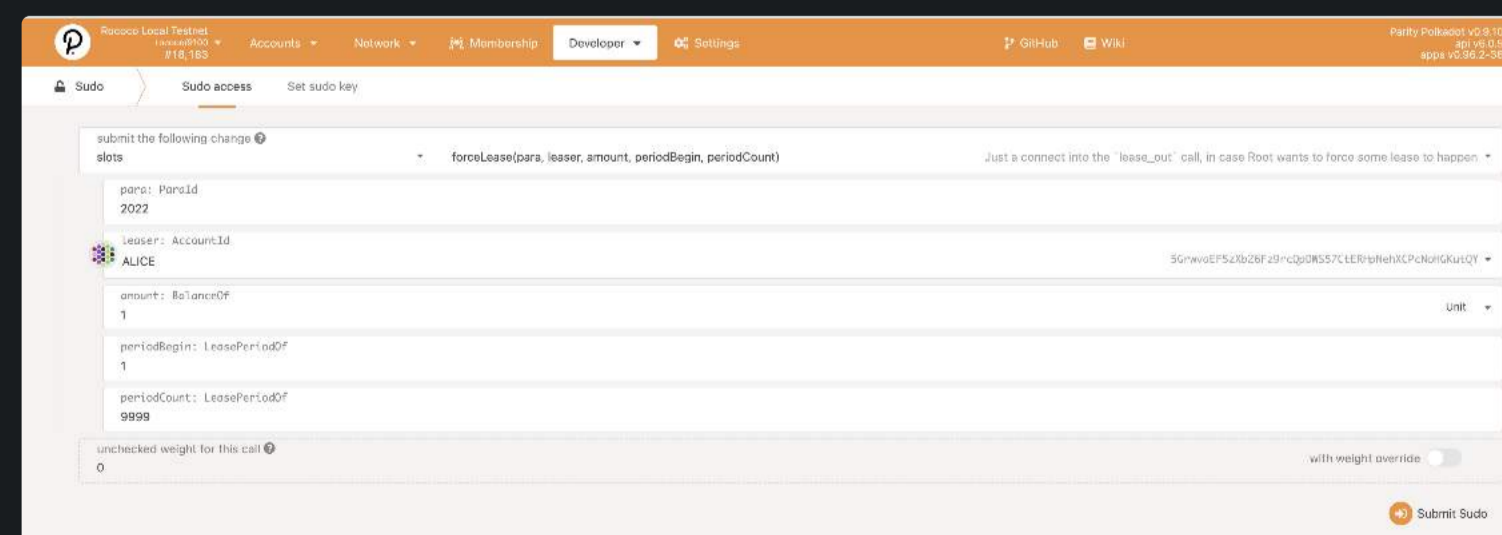
- you should have locally compiled binaries, for both `polkadot` and `litentry-collator`
- run `./scripts/launch-local-binary.sh litentry <path-to-polkadot-bin> <path-to-litentry-parachain-bin>`

ⓘ Likewise, when finished with the dev network, run `make clean-binary` to stop the processes and tidy things up.

Note this command should work for both litentry and litmus.

## extend the leasing period

The default leasing duration for parachain is 1 day, in case you want to extend it (even after it's downgraded to [parathread](#)), simply do a `forceLease` via `sudo`, it should be upgraded to parachain soon again and start to produce blocks.



polkadot-js extrinsic to extend the leasing period

ⓘ You can also refer to README on <https://github.com/litentry/litentry-parachain>

# Run tests

You might want to try out the pre-written tests locally.

## run ts-tests locally

To run the ts-tests (the test under `ts-tests/` folder) locally, similar to launching the networks, it's possible to run them in either docker or binary mode:

```
make test-ts-docker-litentry
```

or

```
# if on Linux
make test-ts-binary-litentry

# otherwise
./scripts/launch-local-binary.sh litentry path-to-polkadot-bin path-to-litentry-parachain
./scripts/run-ts-test.sh
```

Be sure to run the clean-up afterwards.

**i** The `make test-ts-*` command above will also launch the dev network before running the test, so you don't have to do it again.

If you already have a running network, simply run `./scripts/run-ts-test.sh` to execute ts-tests.

## run runtime integration test

There're also integration tests for Litmus and Litentry runtime which are written in rust. To run them:

```
# for Litmus
cargo test --release -p litmus-parachain-runtime --lib

# for Litentry
cargo test --release -p litentry-parachain-runtime --lib
```



Previous  
Launch a local network

Next - Parachain  
How-to guides

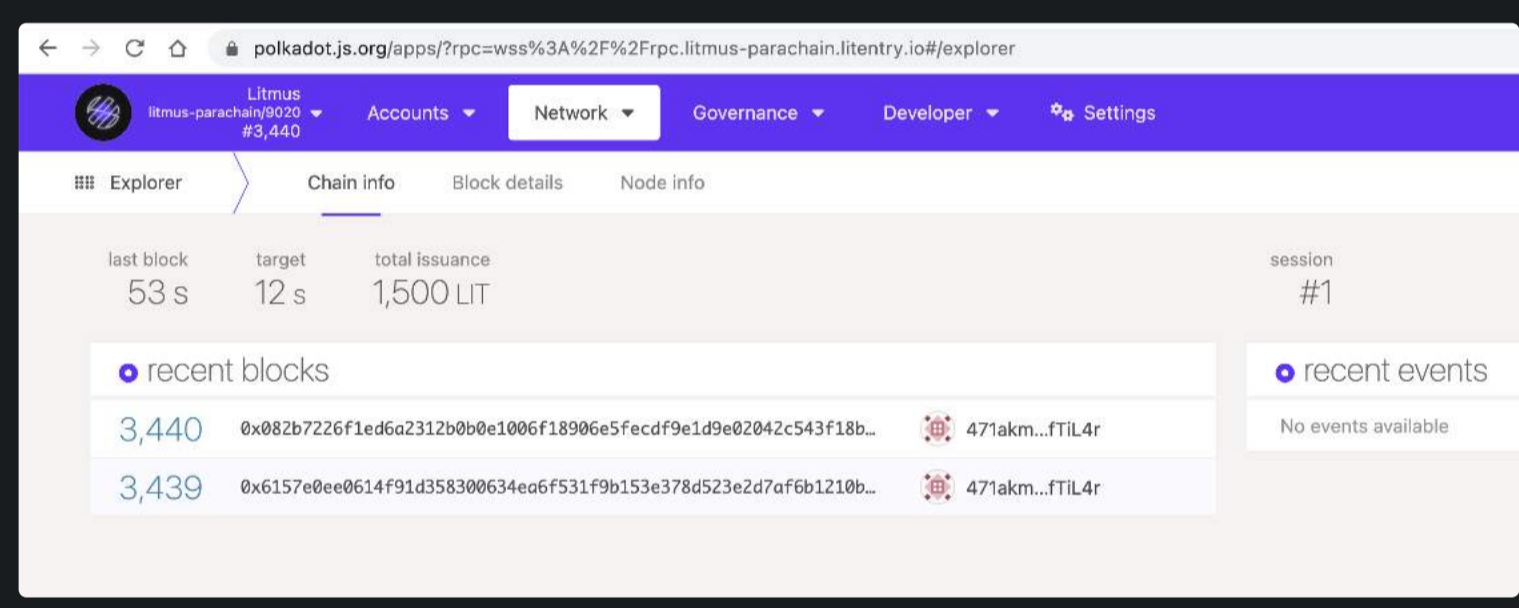




# Interact with parachain

## Main Litmus RPC endpoint:

<https://polkadot.js.org/apps/?rpc=wss%3A%2F%2Frpc.litmus-parachain.litentry.io#/explorer>



Litmus RPC endpoint

Users can query chain state, execute read-only RPC calls and perform extrinsics to read or write the chain state.

## Polkadot{.js} Browser Plugin

Litentry parachain use the Substrate-based chain address format SS58.

The quickest way for generating a Litmus/Litentry account is by Polkadot{.js} [browser extension](#).

## Blockchain Explorers

- [Polkadot Explorer Explorer](#) - Litentry console dashboard block explorer, can be configured to connect to other remote or local endpoints.
- [Polkascan](#) - Blockchain explorer for Polkadot, Kusama, and other related chains.
- [Statescan](#) - Blockchain explorer for Substrate chains, including Litmus/Litentry.

*More commonly used interactions will be added later*

← Parachain - Previous  
**How-to guides**

Next  
**Claim crowdloan rewards** →

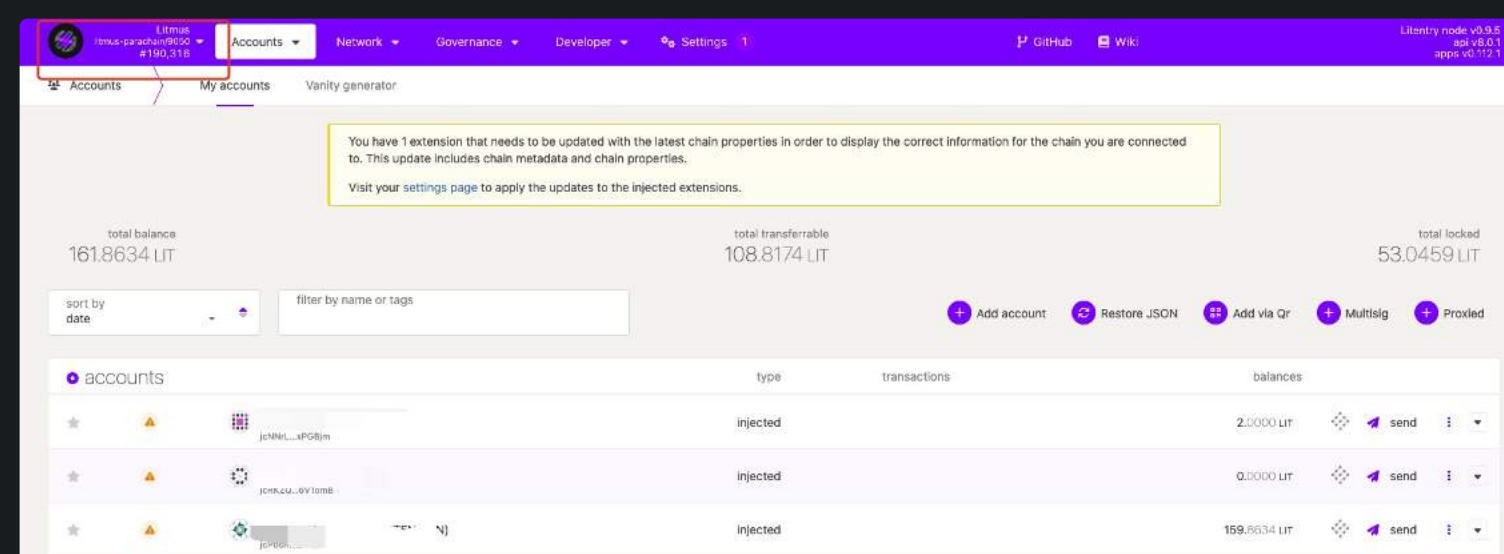
# Claim crowdloan rewards

Litmus and Litentry crowdloan rewards claims are now online. Please claim your vested LIT following the claim guide below.

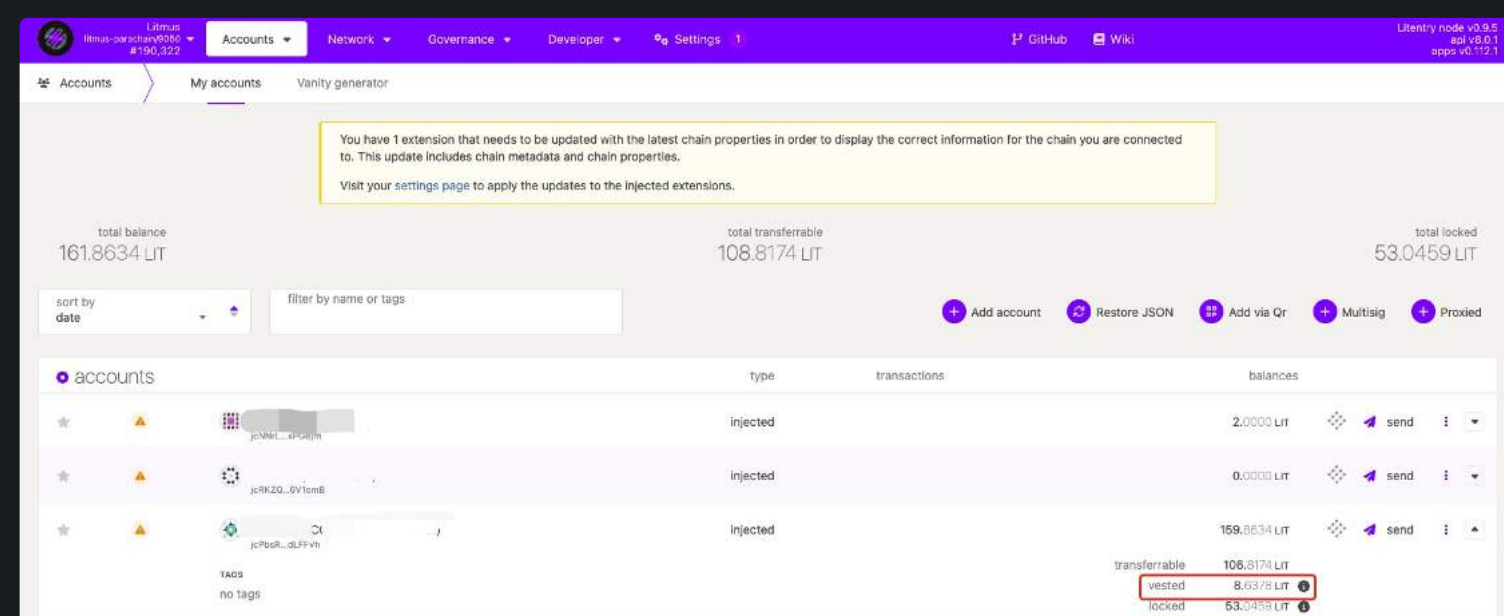
There are two approaches to claiming your crowdloan rewards. The first is via Polkadot.js and the other one is via the [Litentry webapp](#).

## Using Polkadot{js} Extension

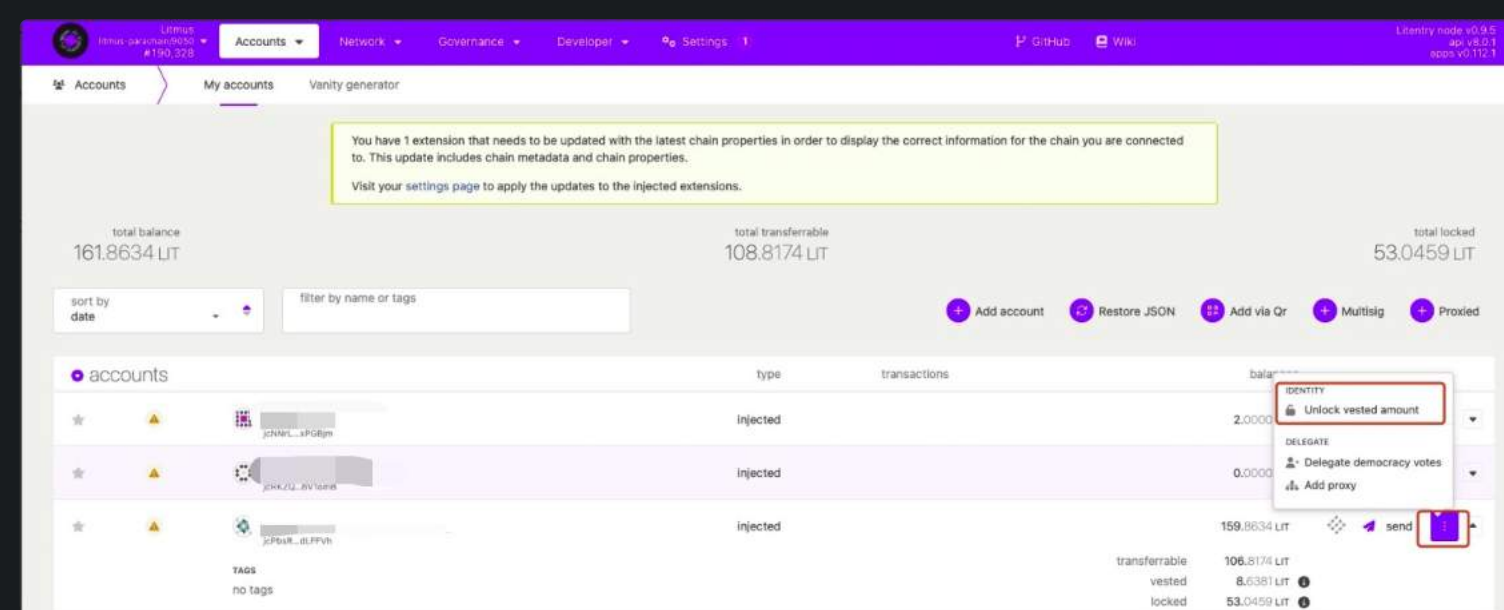
1. Navigate to the [Polkadot.js](#) Website. Connect your Polkadot{js} Extension, using the same account that participated in the crowdloan event, and follow the prompts to complete the process.



2. Select the account that was used in the Litmus/ Litentry crowdloan and check the vested LIT.



3. Click the **hamburger icon** (three dots) on the right side of the account and submit the **UnLock vested amount** button.



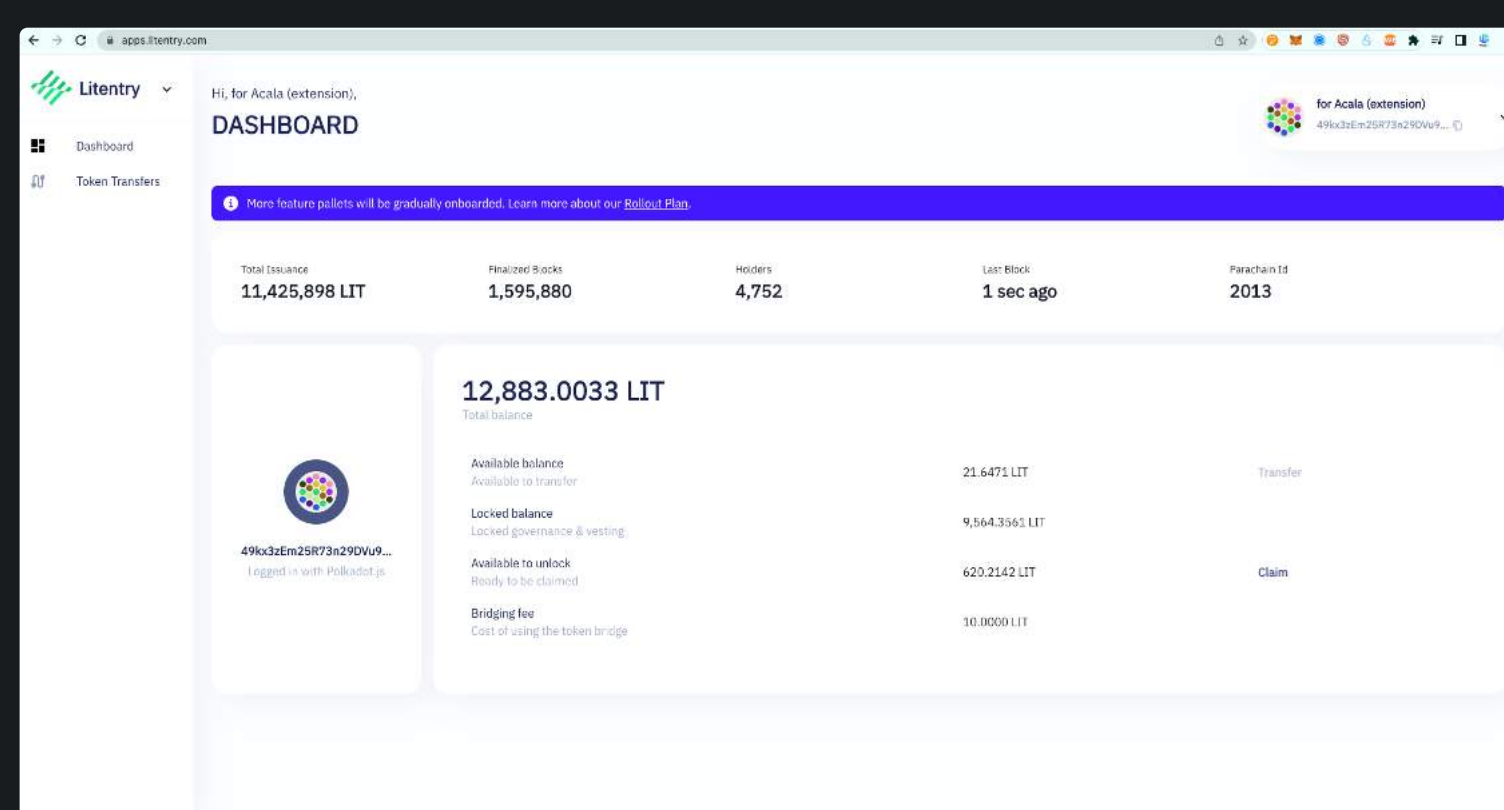
4. Enter your password to sign the transaction and you have successfully claimed your reward.

You'll be charged a small fee to execute the transaction, so make sure you have some balance in your wallet.

5. To confirm, check `vesting()` in the transaction history on [Statescan](#) and click the corresponding Extrinsic ID.

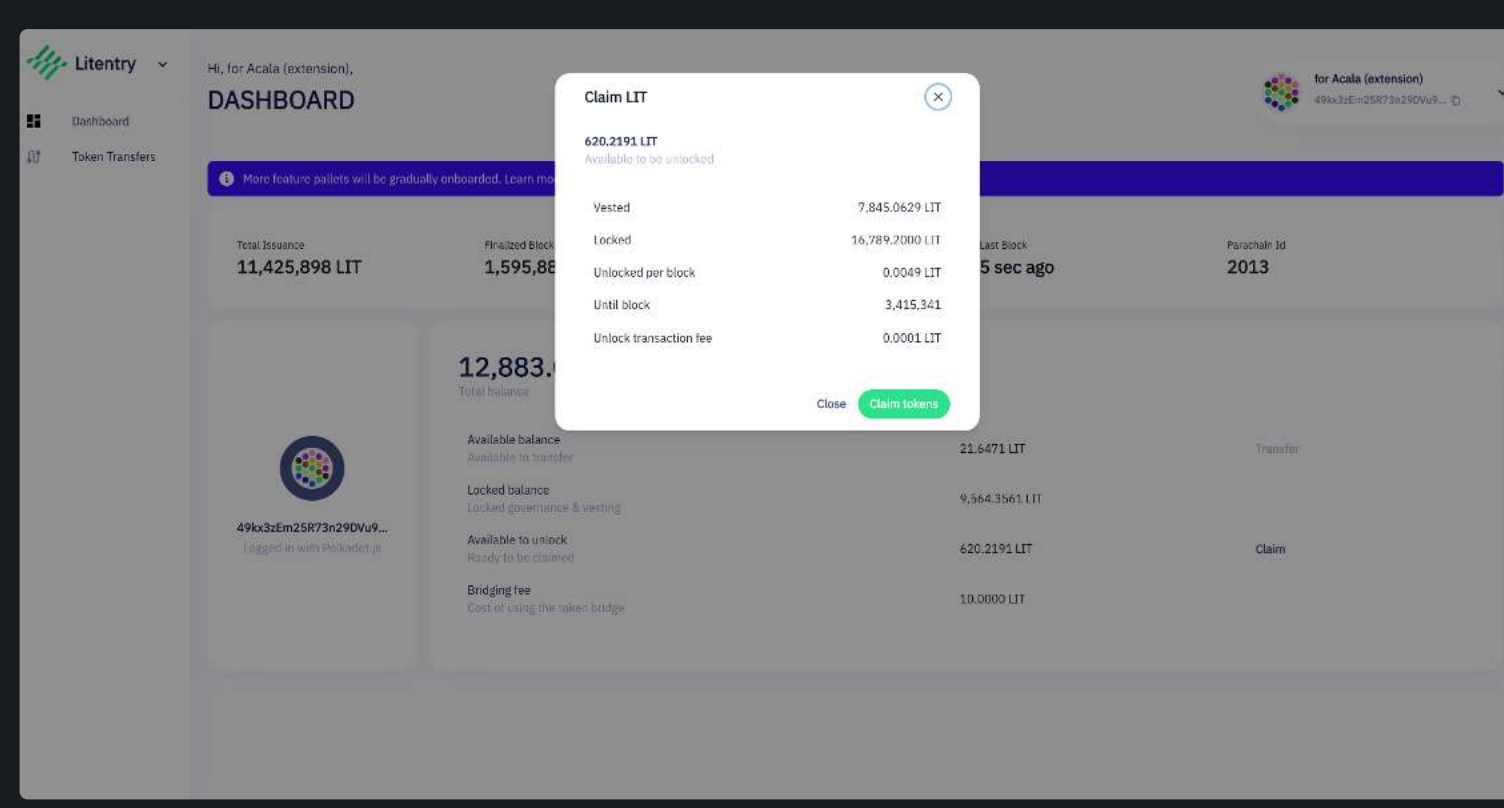
## Using the Litentry Webapp

1. Visit the Litentry Webapp via <https://apps.litentry.com> and sign in by connecting your wallet on the top right-hand corner of your screen.

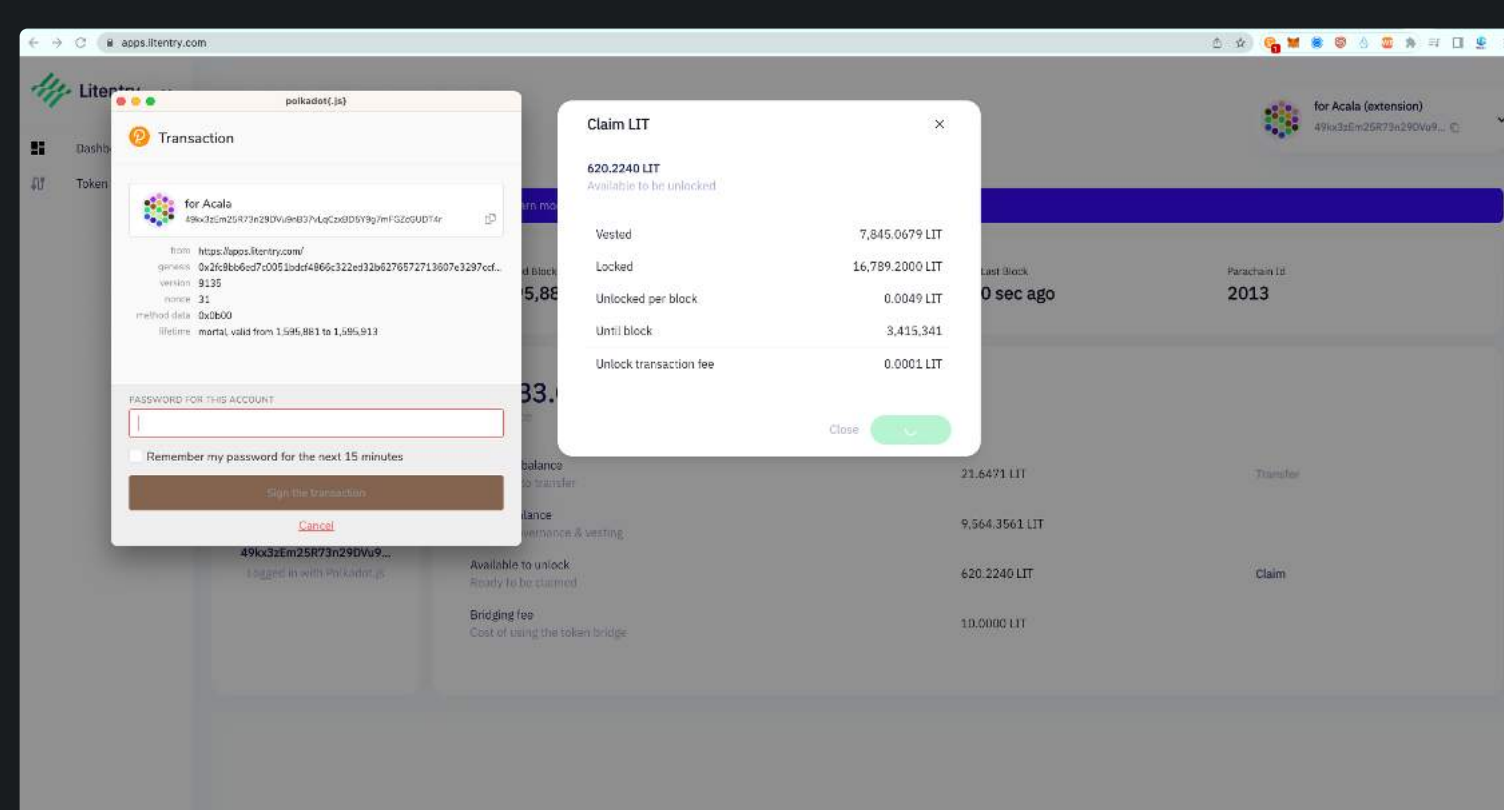


Note: Ensure you connect the wallet you used to participate in the crowdloan.

2. Next, click the "Claim" button right in front of the "Available to unlock" balance and click "Claim tokens" in the pop-up window as shown below:



3. Sign your transaction by entering your wallet password and your crowdloan reward will be added to your balance.



Note: You'll be charged a small fee to execute the transaction, so make sure you have some balance in your wallet.

# Transfer LIT from Ethereum to parachain

The token bridge allows you to transfer your LIT token from the Ethereum network into the Litentry Network.

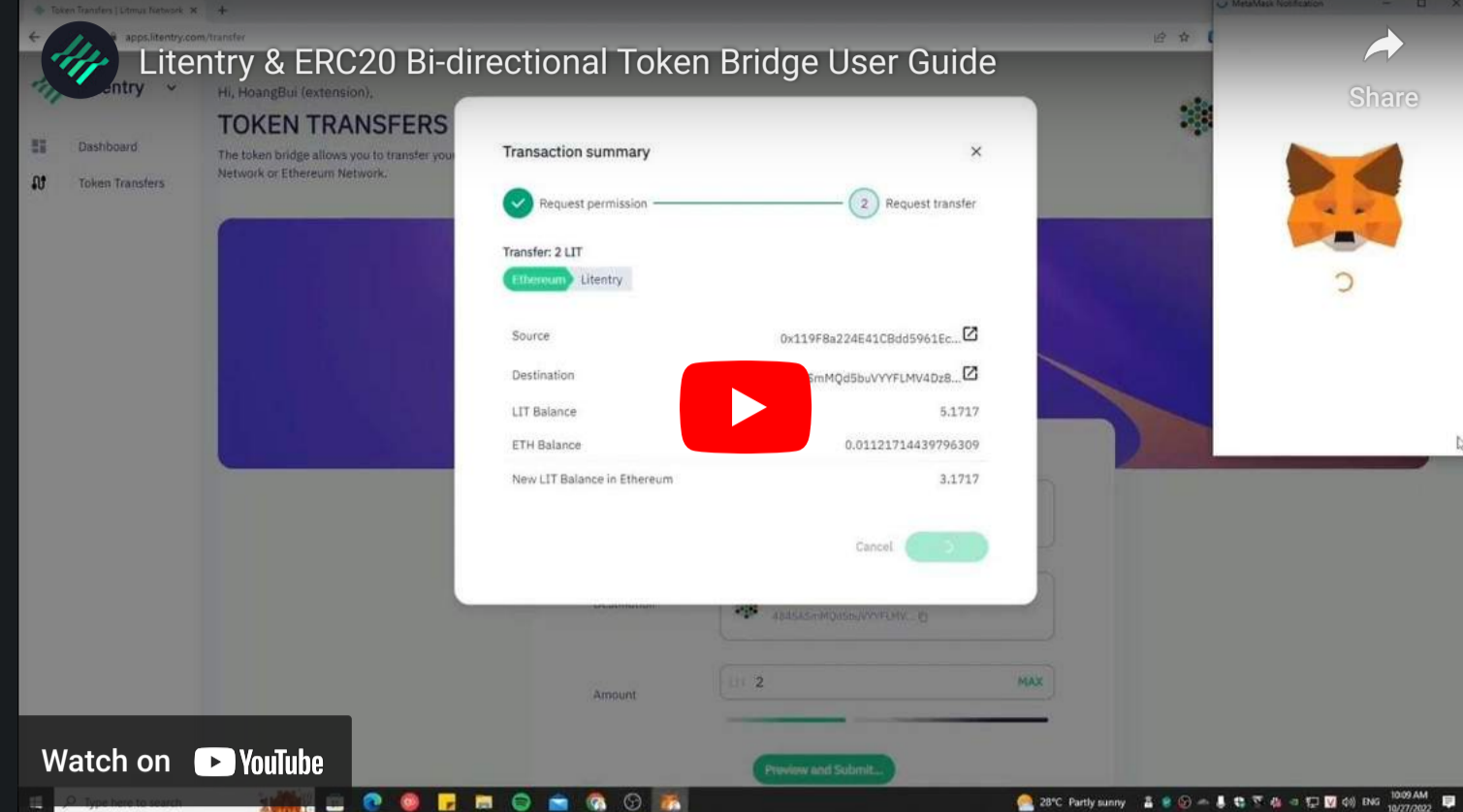
You can access our token bridge functionality by visiting <https://apps.litentry.com/transfer>

## An Introduction to the Token Bridge

Please refer to [Token Bridge](#)

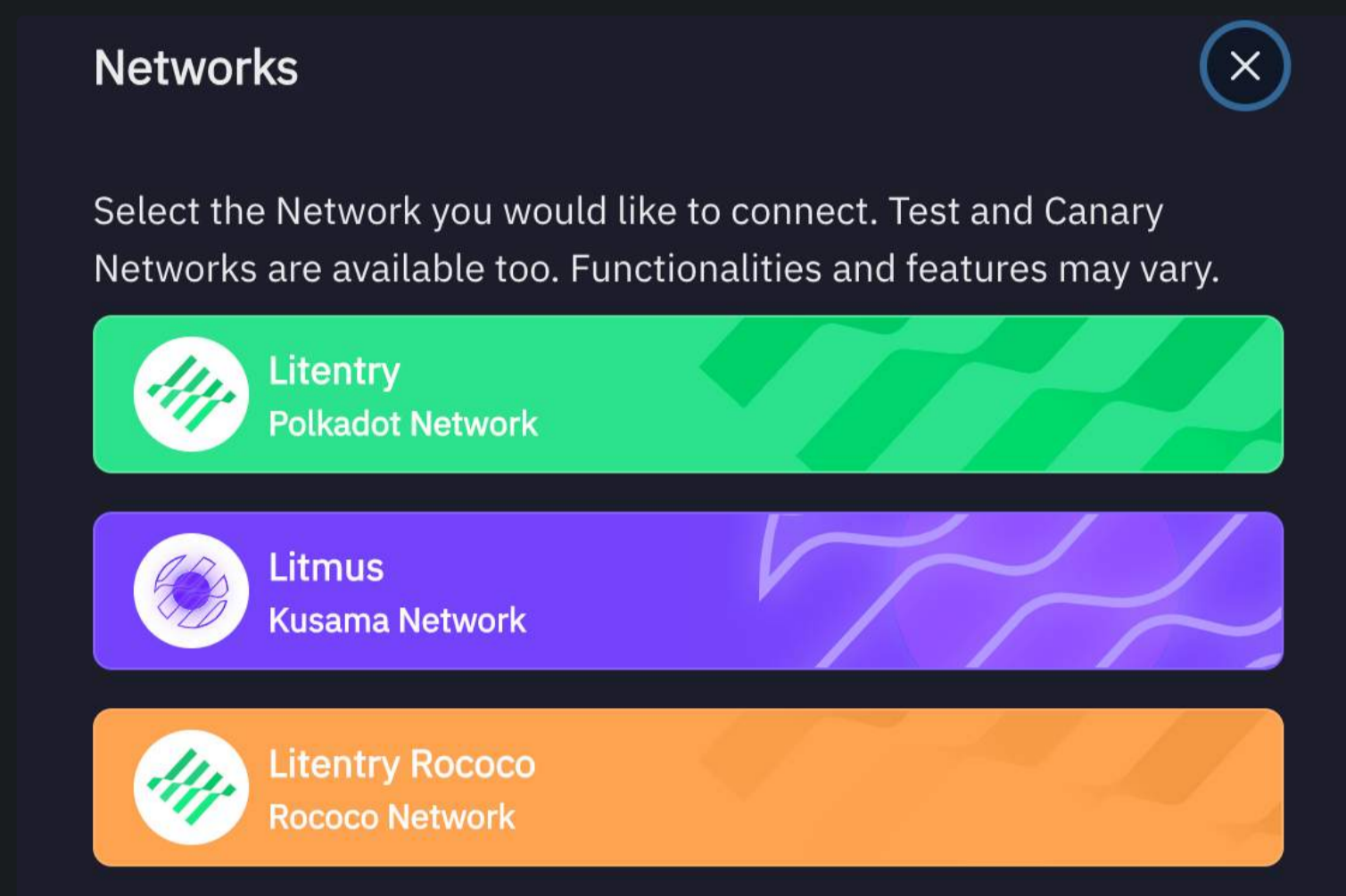
## Step-by-Step

This video guide takes **Ethereum => Litmus** parachain as an example, it also applies to **Ethereum => Litentry** parachain token bridge usage. You only need to switch to the corresponding network in the web app.

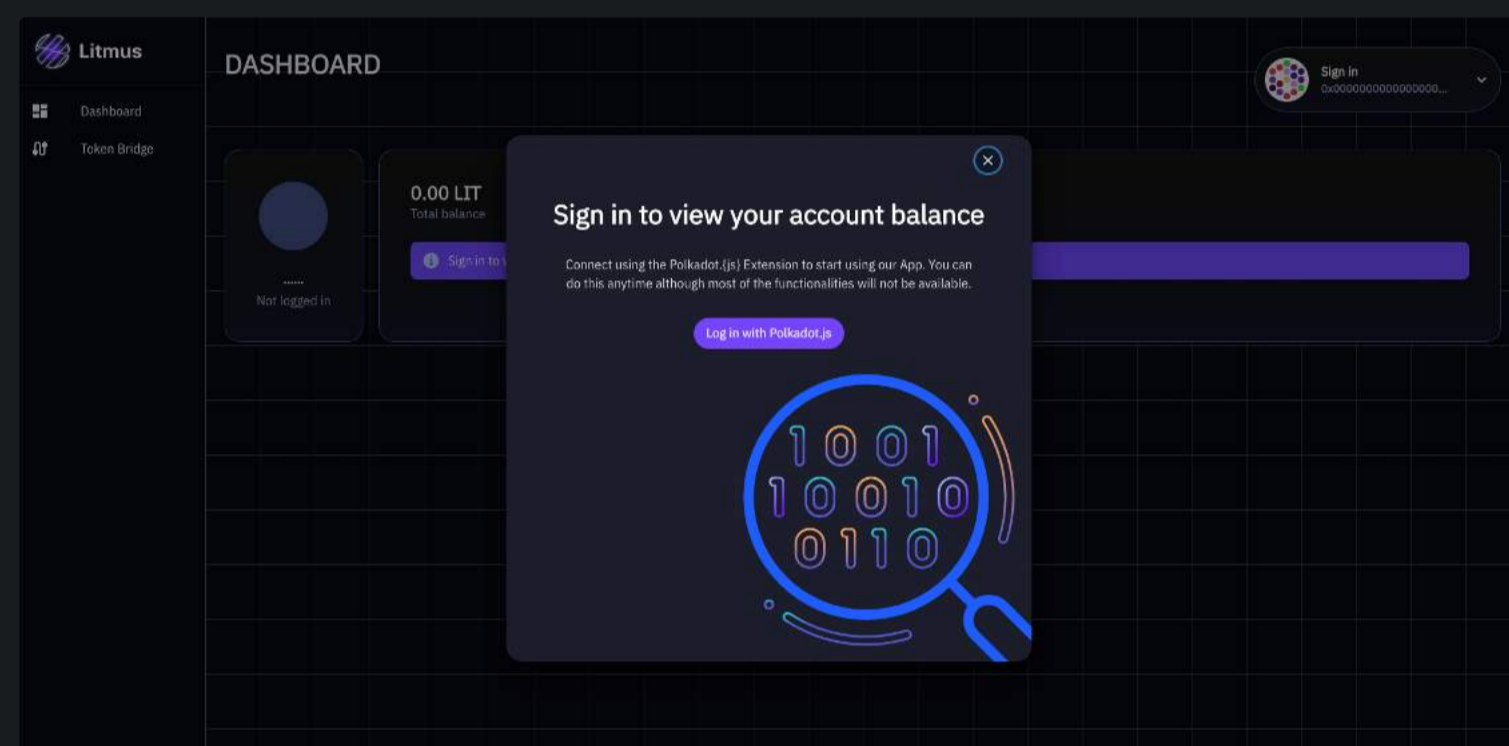


Alternatively, you can follow the step-by-step guide below:

1. Switch to the desired network in the upper left corner of the webapp:

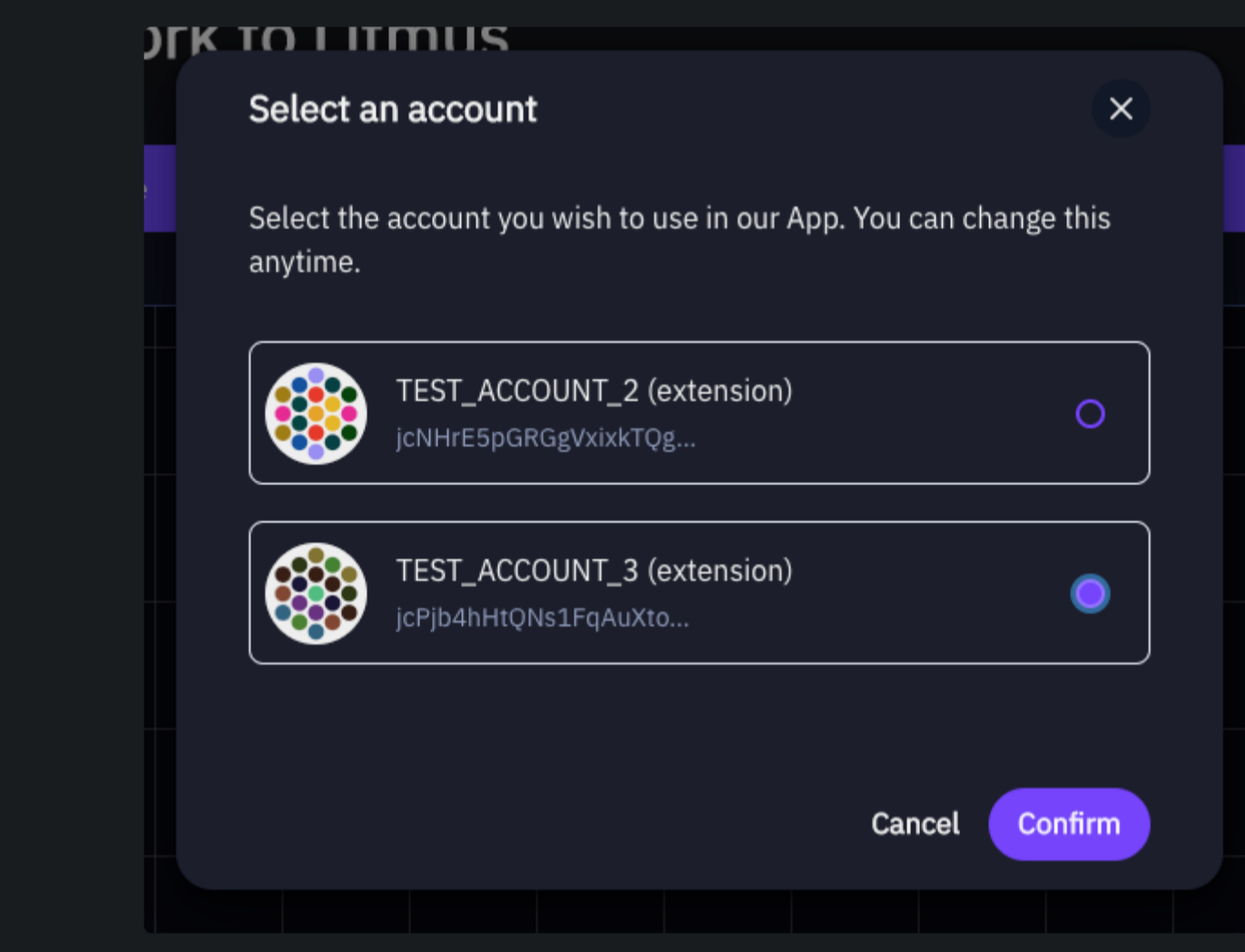


2. Connect using the Polkadot.(js) Extension to start using our app



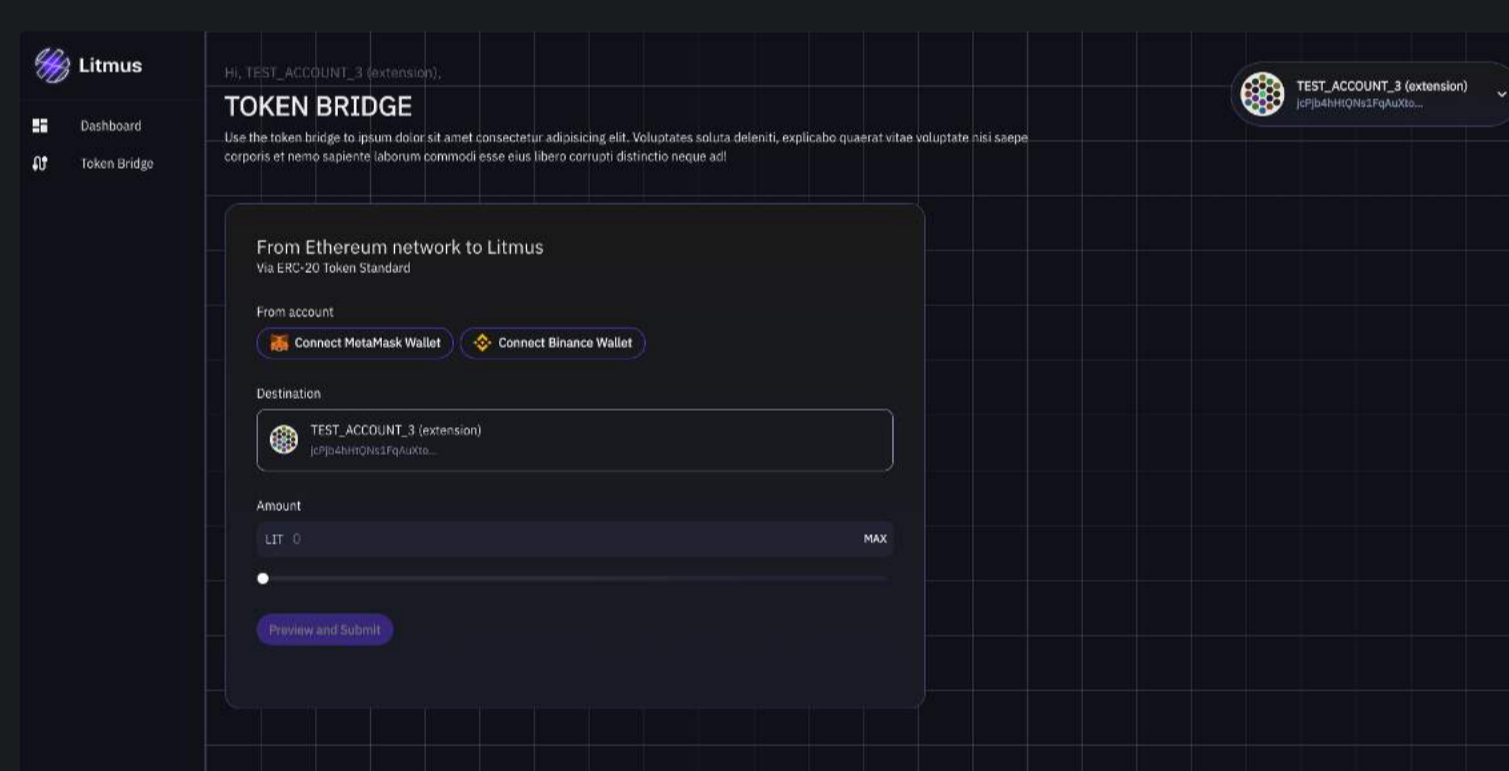
You can cancel this modal and do this anytime. However, please note that you must connect to a Polkadot account else most app functionality will not be available to you

3. Select the Polkadot Account you wish to connect to

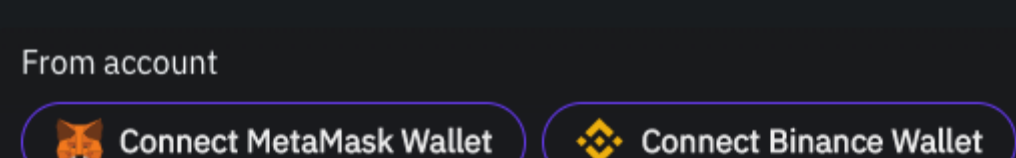


Select and press Confirm

4. Navigate to the **Token Bridge** screen



5. Choose which account you wish to transfer LIT from and connect to your Ethereum Wallet. Please note that your browser MetaMask or Binance extension will open ask you to sign in.



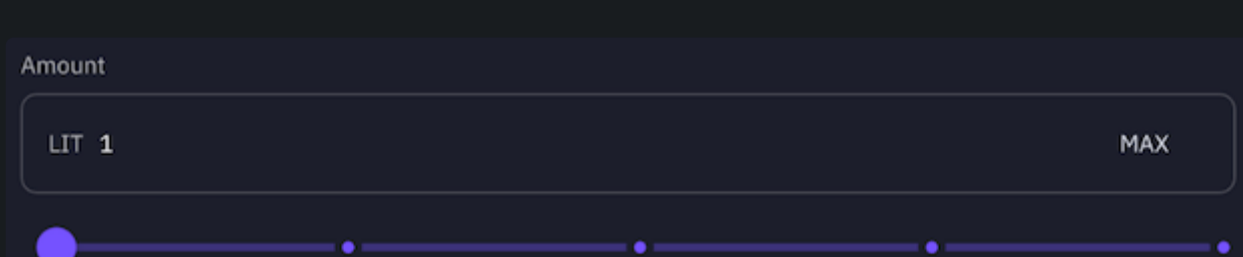
You can select connect to your Ethereum wallet on MetaMask or Binance

6. You can change the Destination and which Polkadot account you wish to transfer the LIT into by using the Polkadot account select control in the top right of your screen

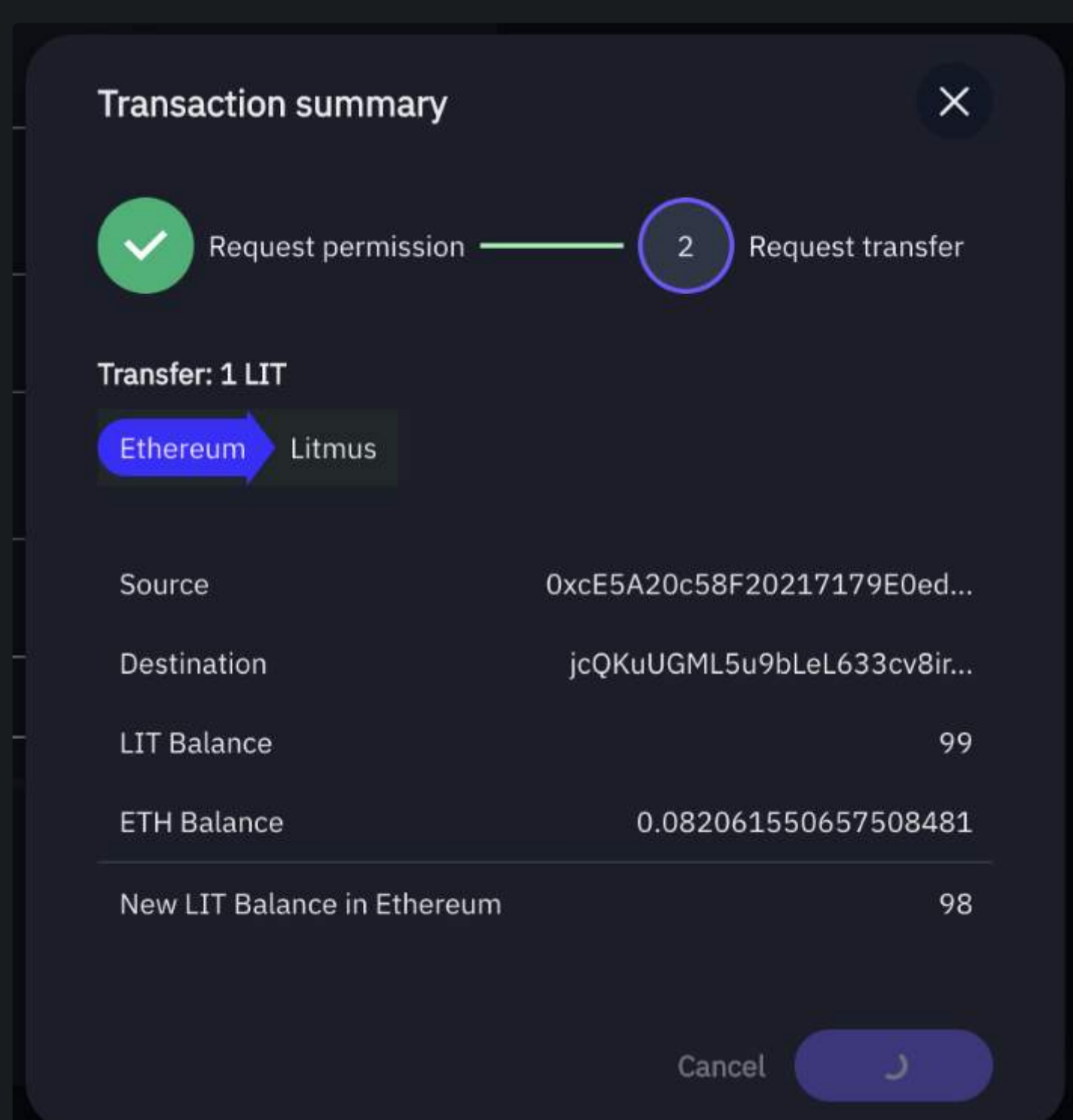


Just

7. Select the amount of LIT token you wish to transfer. You will be prompted and required to authorize the amount prior to being able to transfer it.



8. Press on **Preview and Submit** and you will be shown the **Transaction Summary** modal. Carefully review the information displayed and then press **Transfer tokens**



9. Depending on where you are transferring from. You will now be presented with a MetaMask or Binance modal. Please sign and approve the transaction and then wait for the transaction to be confirmed.

Congratulations! You have successfully transferred your LIT token from Ethereum to Litmus.

You can check if the transaction was successful by checking on Etherscan (<https://etherscan.io/>) and the Polkadot Network Explorer (<https://polkadot.js.org/apps/#/explorer>).



# Transfer LIT from parachain to Ethereum

The token bridge allows you to transfer your LIT token from Litmus Network into the Ethereum network as well.

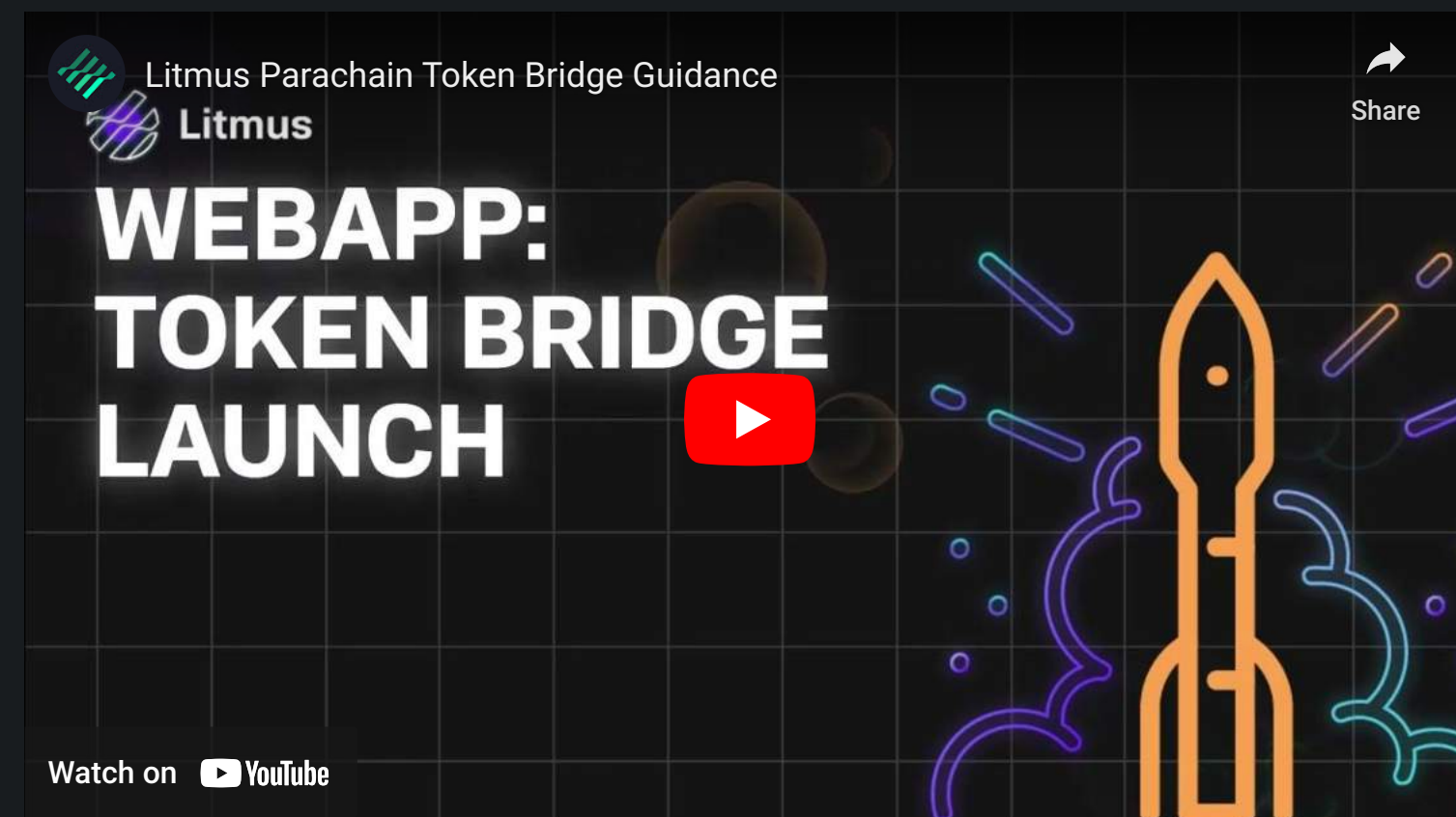
You can access our token bridge functionality by visiting <https://apps.litentry.com/transfer>

## An introduction to the Token Bridge

Please refer to [Token Bridge](#)

## Step-by-Step

This guide takes **Litmus => Ethereum** parachain as example, it also applies to **Litentry => Ethereum** parachain token bridge usage. You only need to switch to the corresponding network in the webapp.



✔ Congratulations! You have successfully transferred your LIT token from Litmus to Ethereum.

You can check if the transaction was successful by checking on [Etherscan](#) and the [statescan blockchain explorer](#).

ⓘ Notes:

1. Currently, the commission is set to be 10 LIT. It's calculated based on the average gas fee on Ethereum and the number of Ethereum transactions required for a token bridge transfer. It can be queried via `chainBridge.bridgeFee` from `polkadot-js`. The [Token Bridge Commission page](#) has more information about how the fee is calculated.
2. The waiting time normally takes around 5-10 minutes but varies depending on the network traffic condition of the Ethereum network.

← Previous  
Transfer LIT from Ethereum to para...

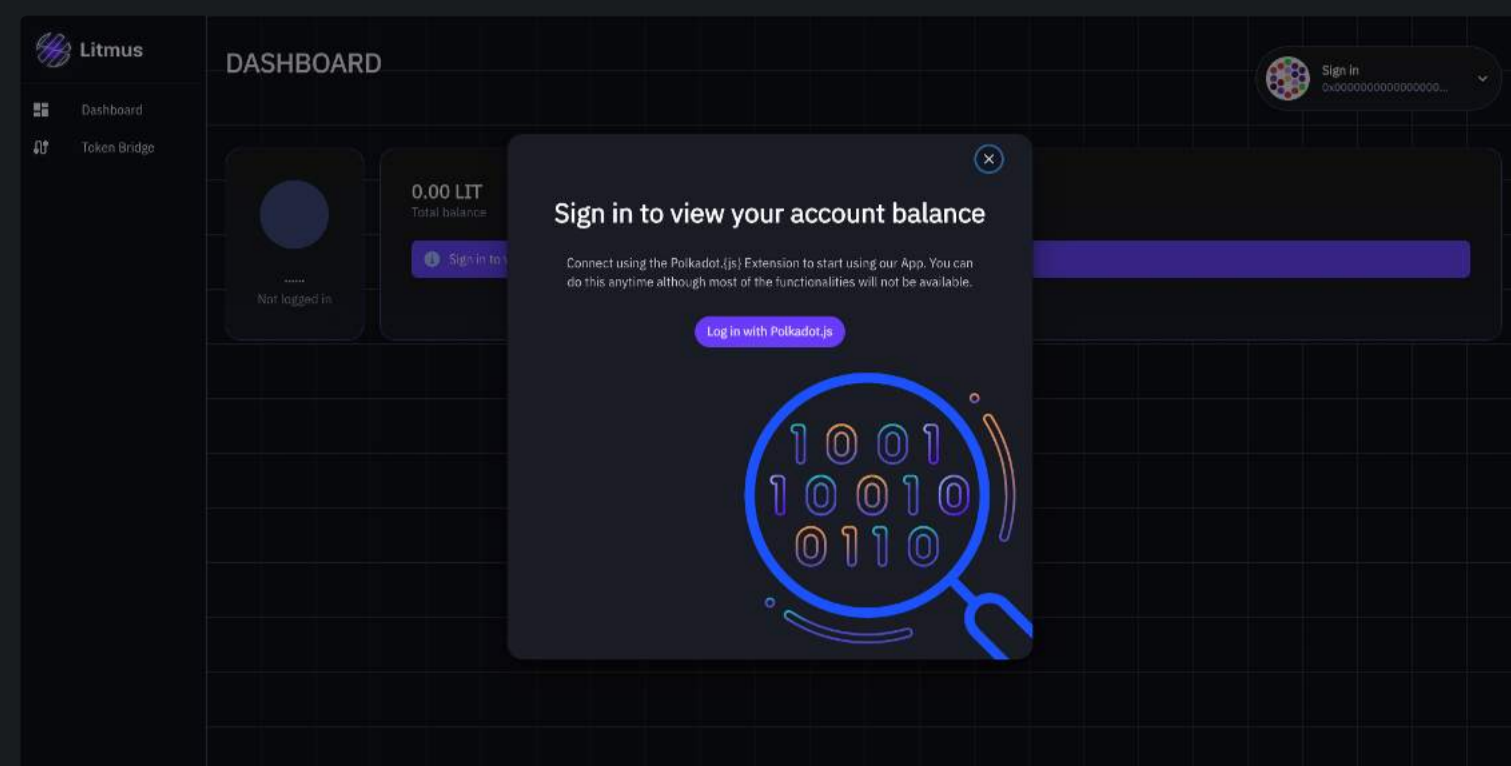
Next →  
Transfer LIT to other parachains wit...

# Transfer LIT to other parachains with XCM

The XCM allows you to transfer your LIT token from Litmus into a remote sibling parachain. You may visit the remote chain's XCM guidance for transferring them the other way around.

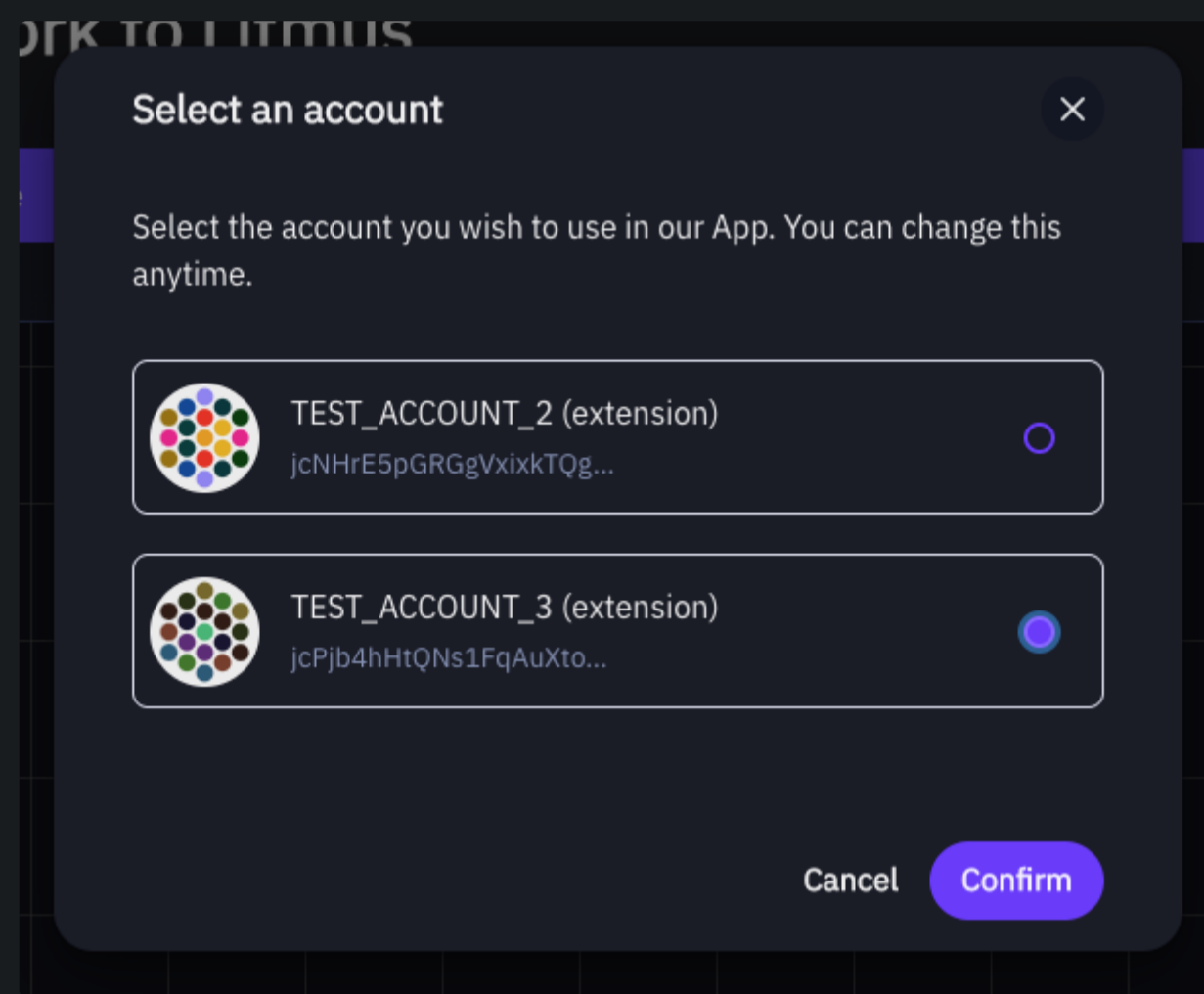
## Step-by-Step

1. Connect using the Polkadot.(js) Extension to start using our app



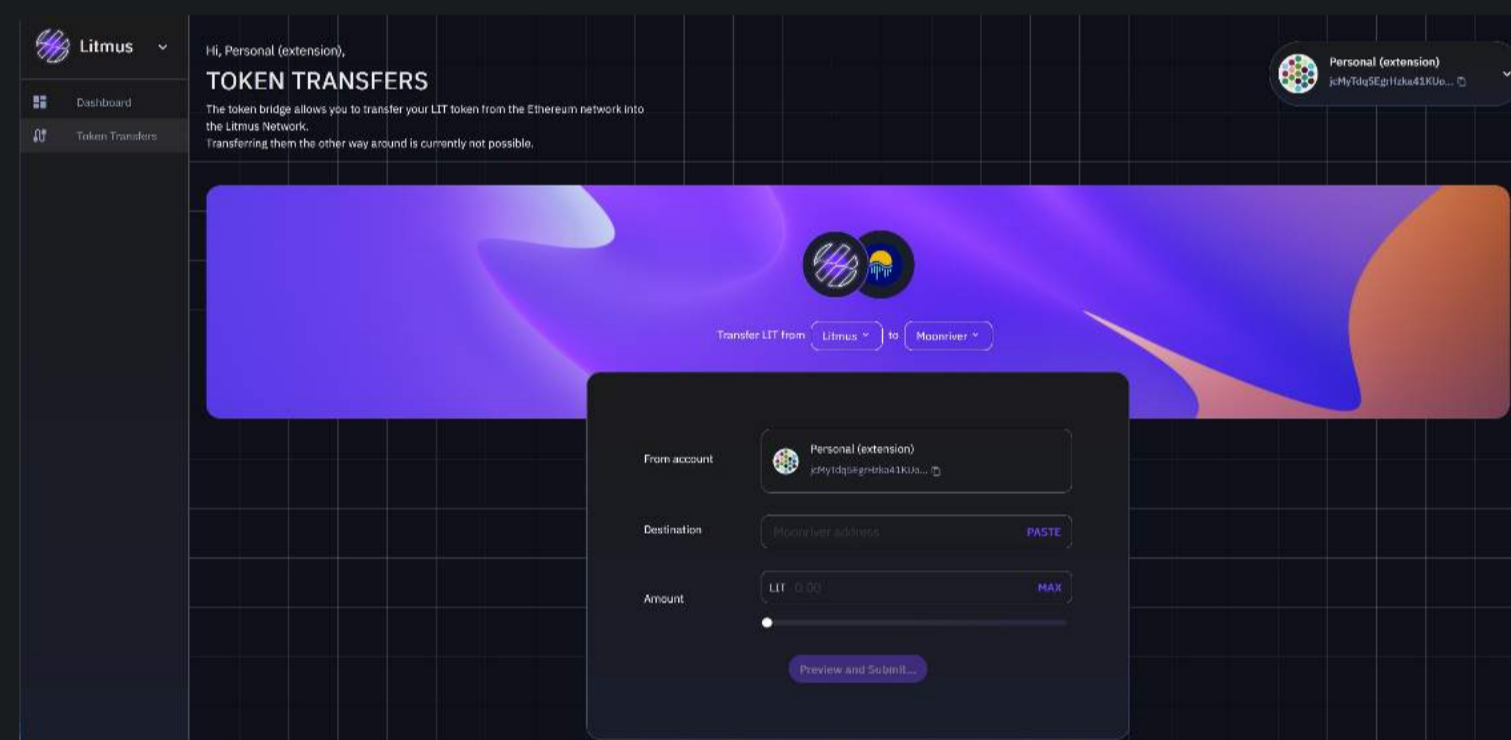
You can cancel this modal and do this anytime. However, please note that you must connect to a Polkadot account else most app functionality will not be available to you

2. Select the Polkadot Account you wish to connect to

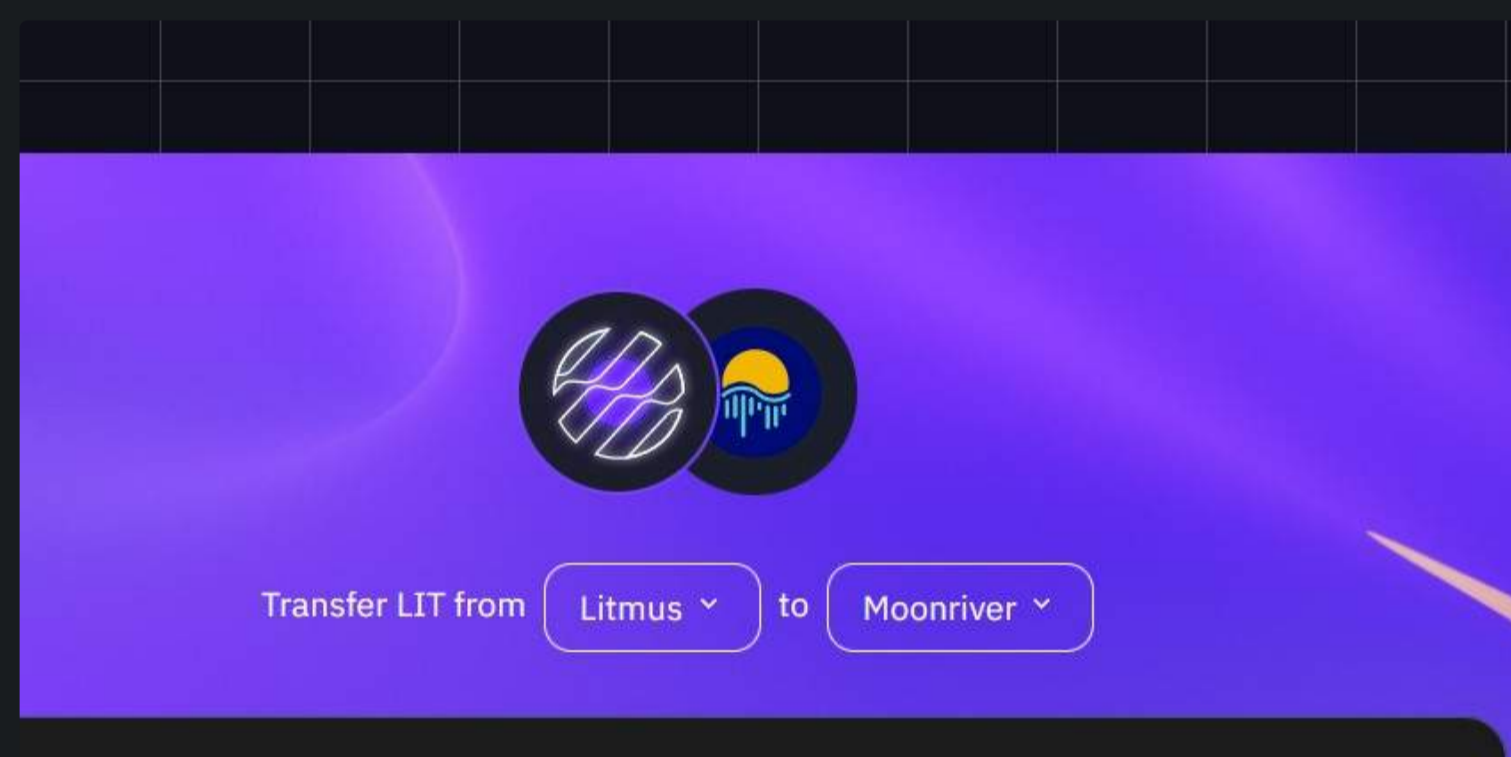


Select and press Confirm

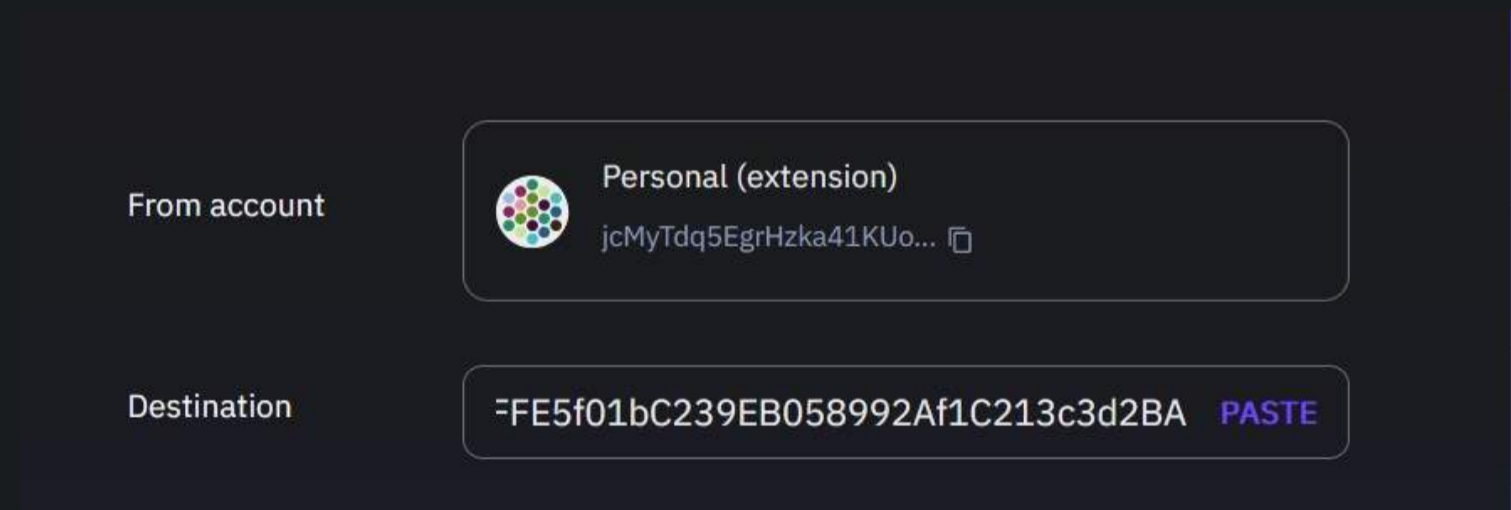
3. Navigate to the **Token Bridge** screen



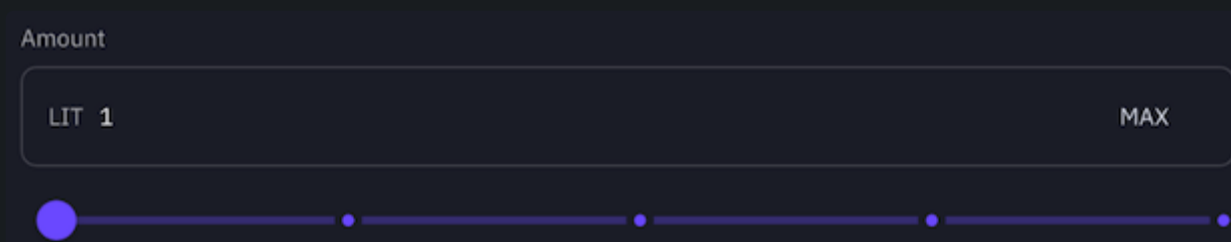
4. Choose the origin chain as Litmus and the target remote chain you wish to transfer LIT to.



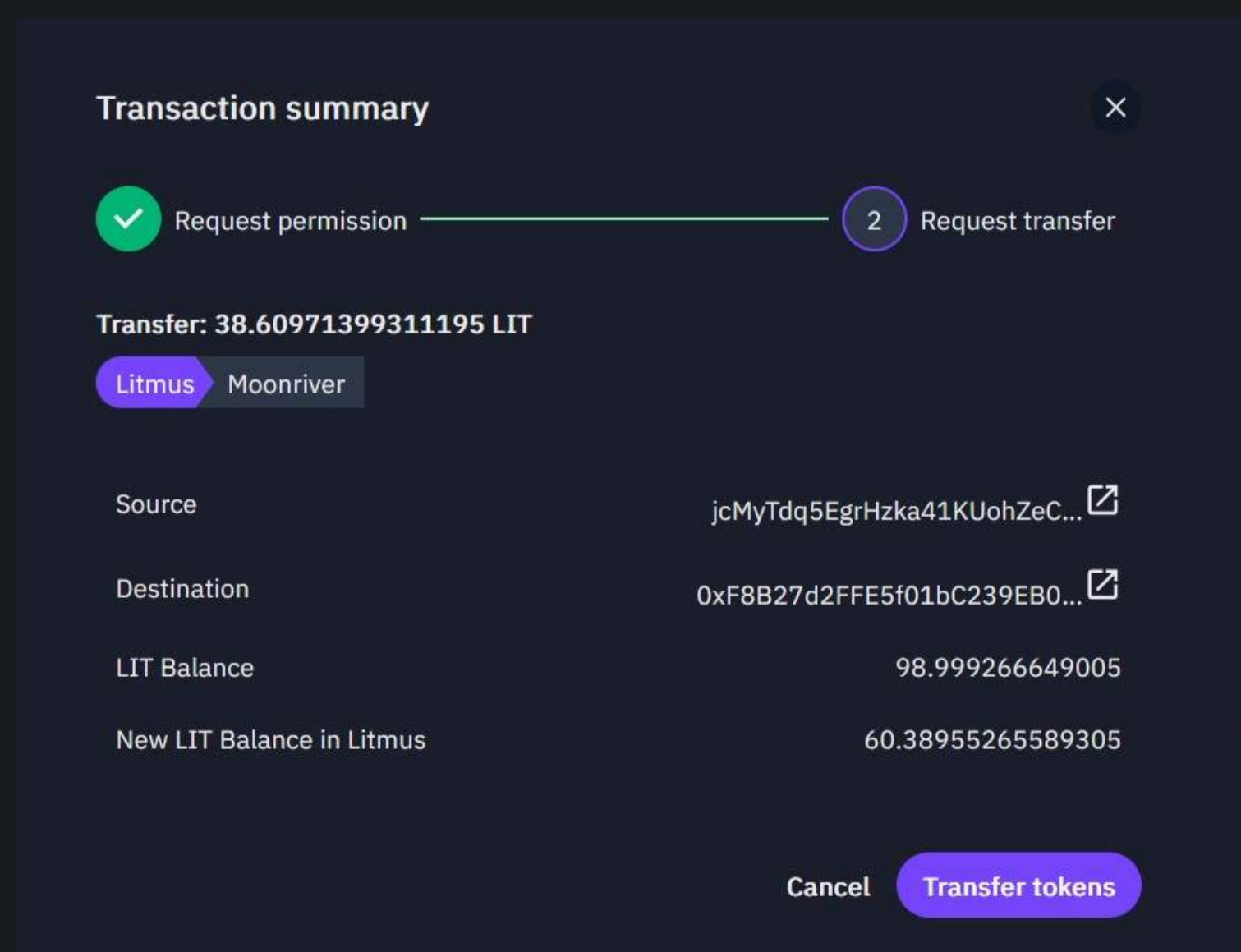
5. You can change the Destination and which Polkadot account you wish to transfer the LIT into by using the Polkadot account select control in the top right of your screen



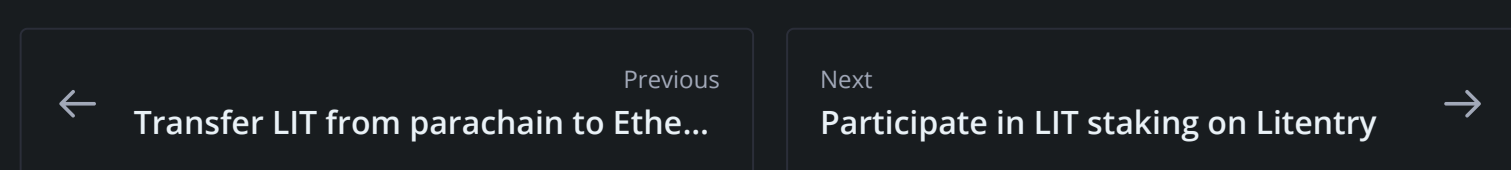
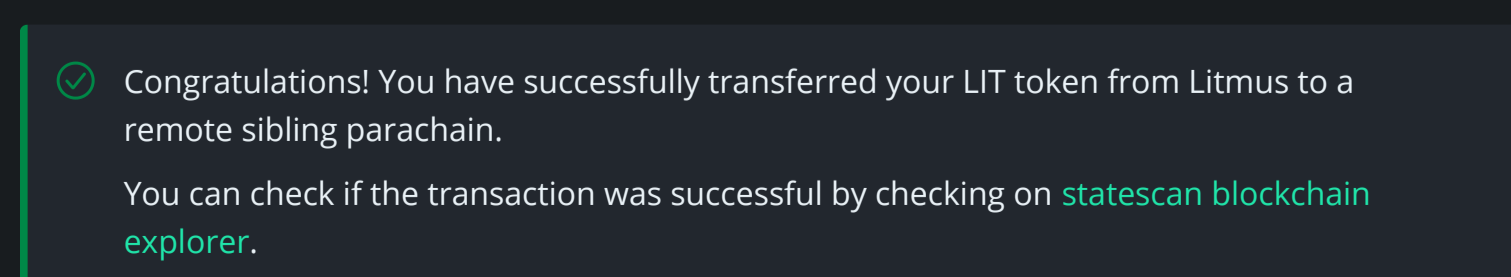
6. Select the amount of LIT token you wish to transfer. You will be prompted and required to authorize the amount prior to being able to transfer it.



7. Press on **Preview and Submit** and you will be shown the **Transaction Summary** modal. Carefully review the information displayed and then press **Transfer tokens**



8. Depending on wallet extension you use, you will get a wallet transaction pop-up. Please sign and approve the transaction and then wait for the transaction to be confirmed.



# Participate in LIT staking on Litentry

LIT staking is currently only available on Litentry - Polkadot parachain.

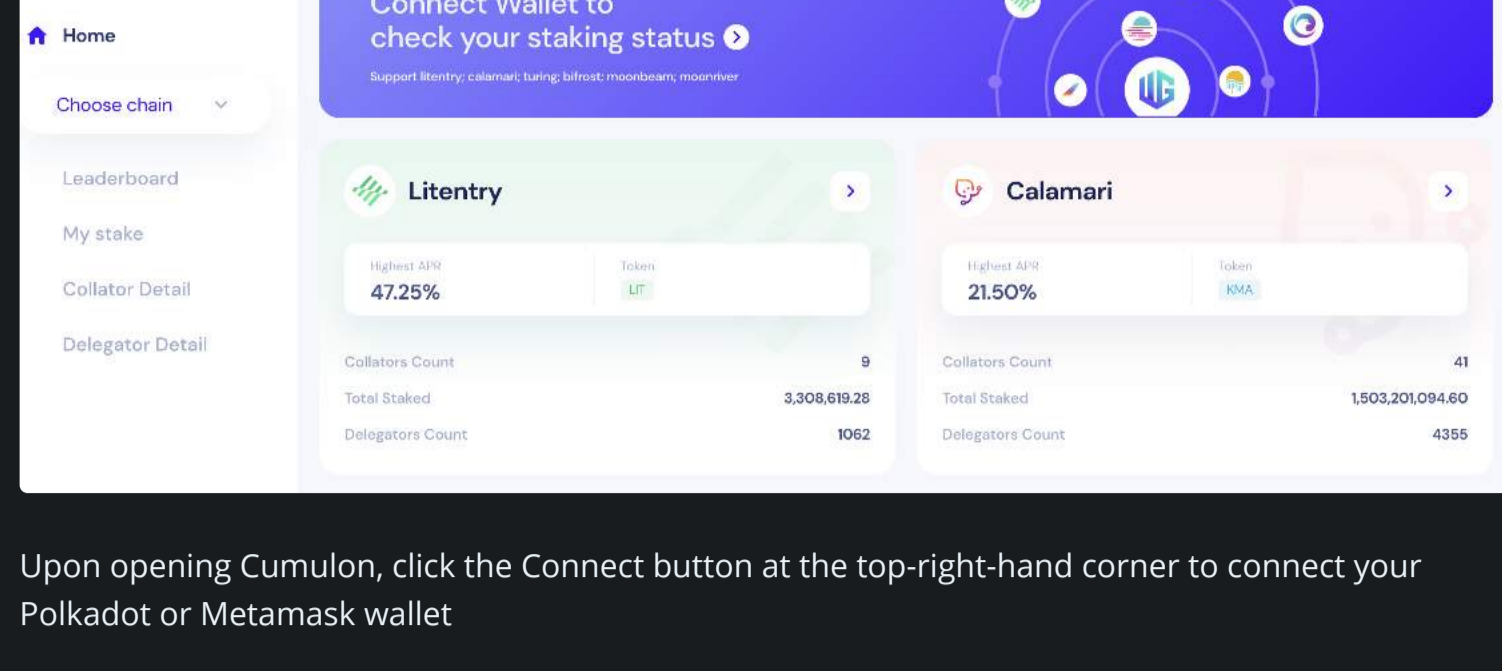
## Overview

Litentry uses a Delegated Proof of Stake (DPoS) consensus mechanism that allows LIT holders to stake their assets and delegate their desired collators through its parachain staking pallet. The staking pallet is designed to permit delegators and collators to share risks and rewards.

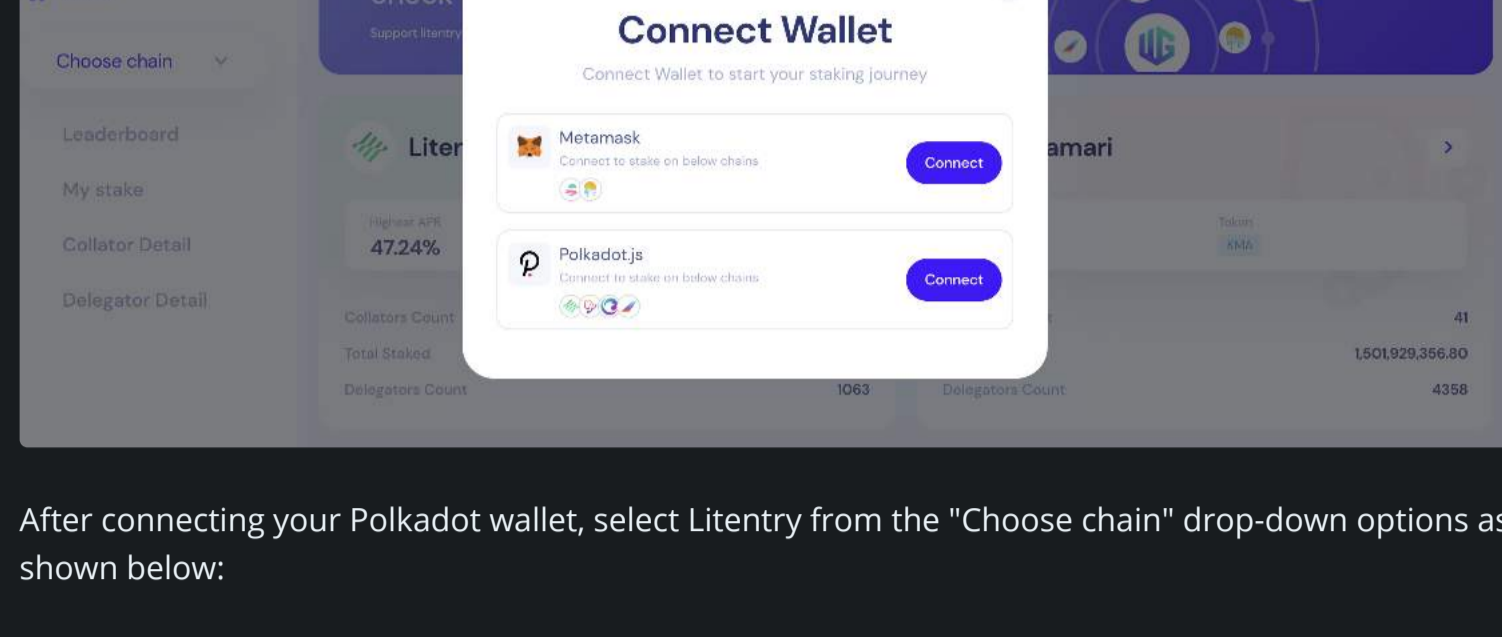
## How to stake

### 1. Cumulon (Previously "Web3Go")

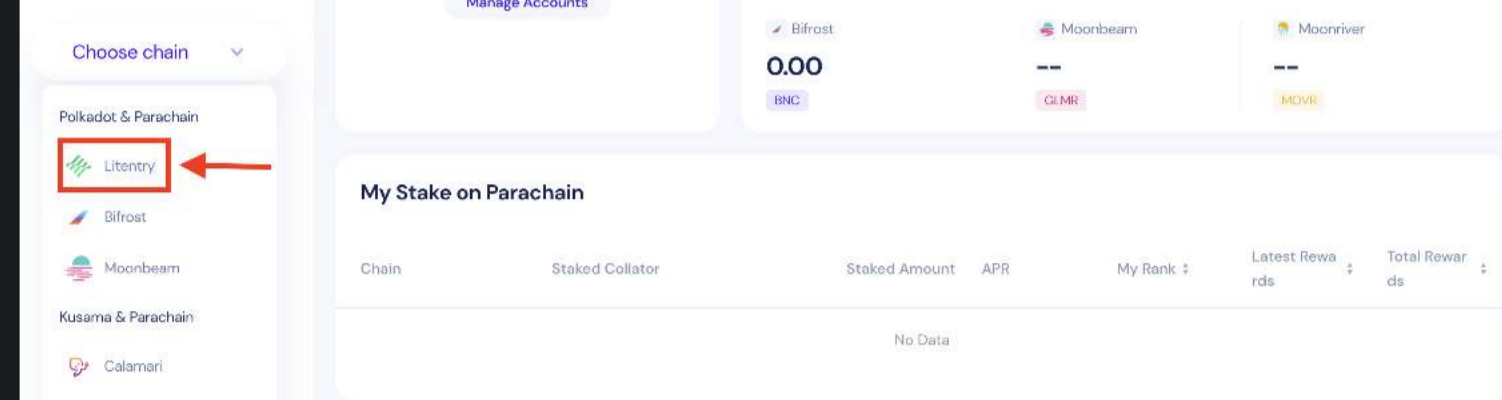
The easiest way to participate in LIT staking is through [Cumulon](#).



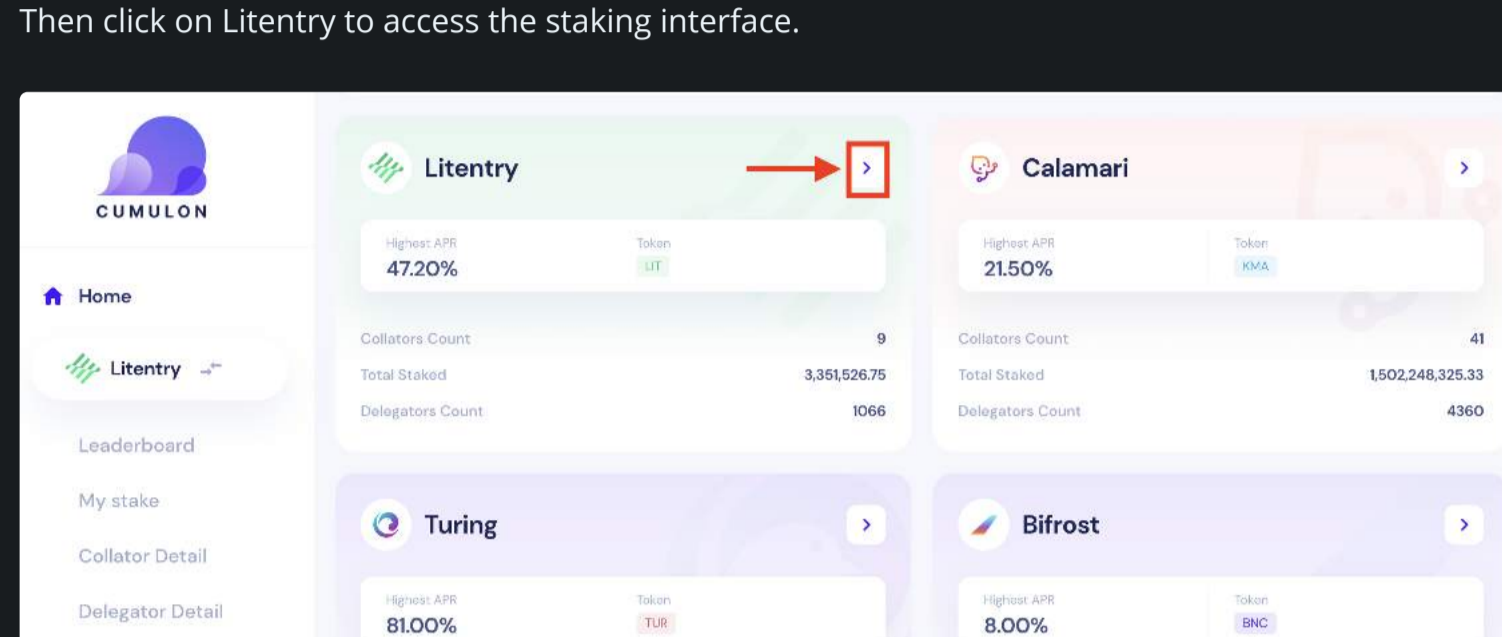
Upon opening Cumulon, click the Connect button at the top-right-hand corner to connect your Polkadot or Metamask wallet



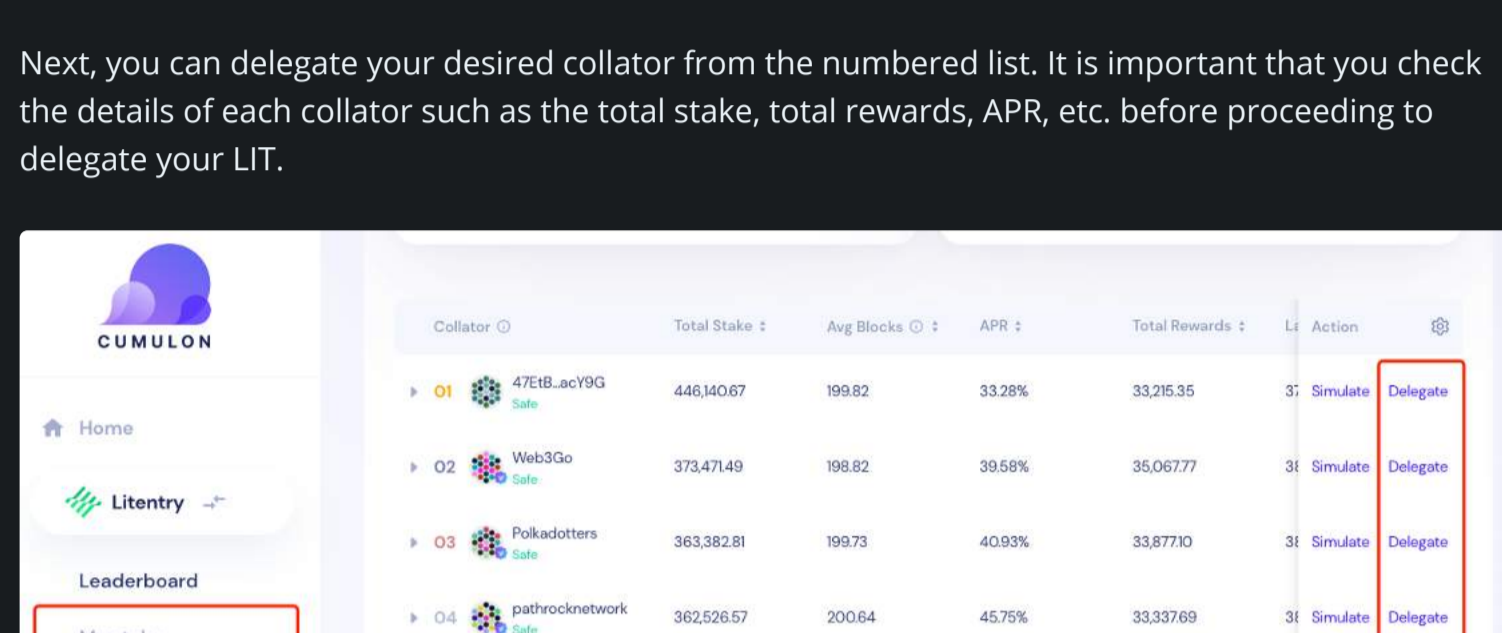
After connecting your Polkadot wallet, select Litentry from the "Choose chain" drop-down options as shown below:



Then click on Litentry to access the staking interface.

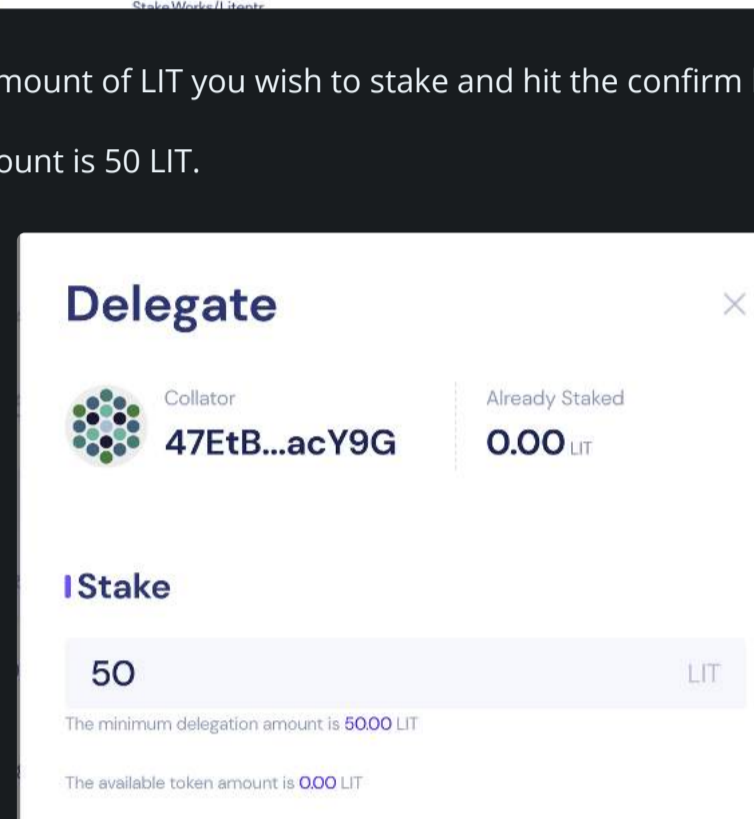


Next, you can delegate your desired collator from the numbered list. It is important that you check the details of each collator such as the total stake, total rewards, APR, etc. before proceeding to delegate your LIT.



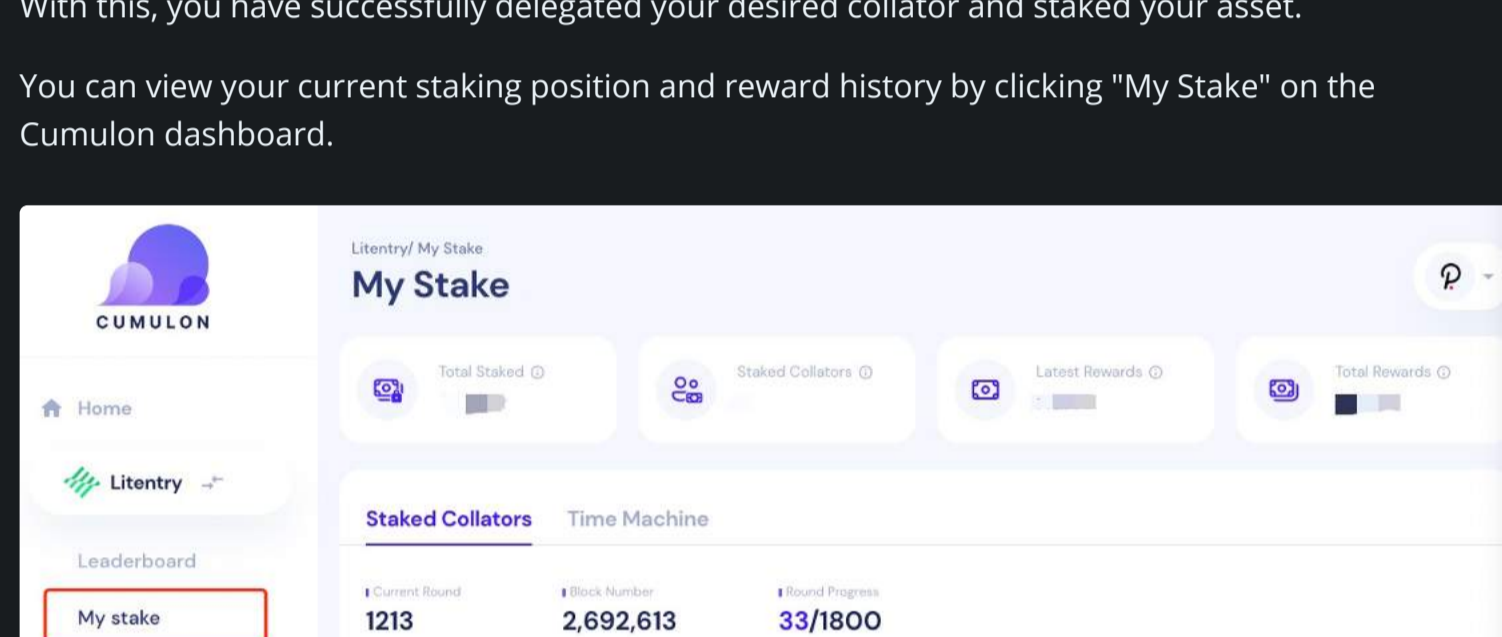
To delegate, enter the amount of LIT you wish to stake and hit the confirm button.

Note: The minimum amount is 50 LIT.



With this, you have successfully delegated your desired collator and staked your asset.

You can view your current staking position and reward history by clicking "My Stake" on the Cumulon dashboard.



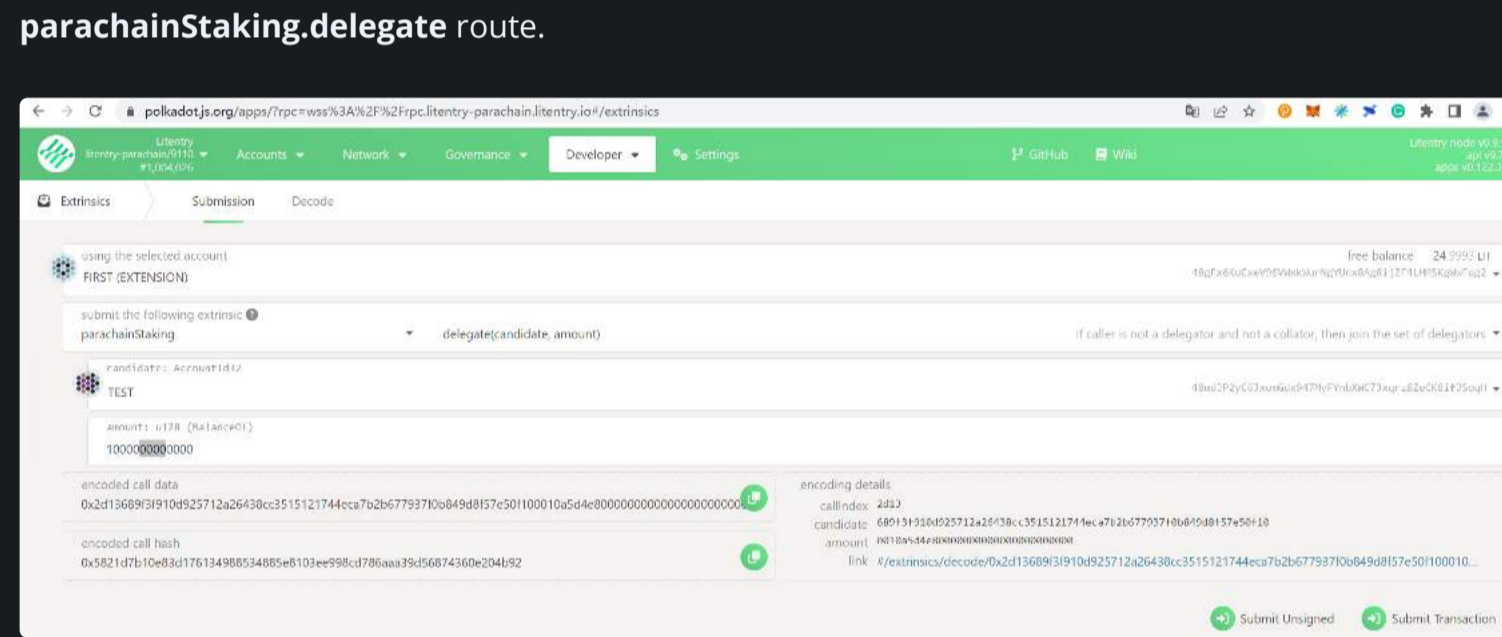
Please refer to Cumulon's [Official Website](#) for more staking dashboard features.

### 2. Polkadot.js

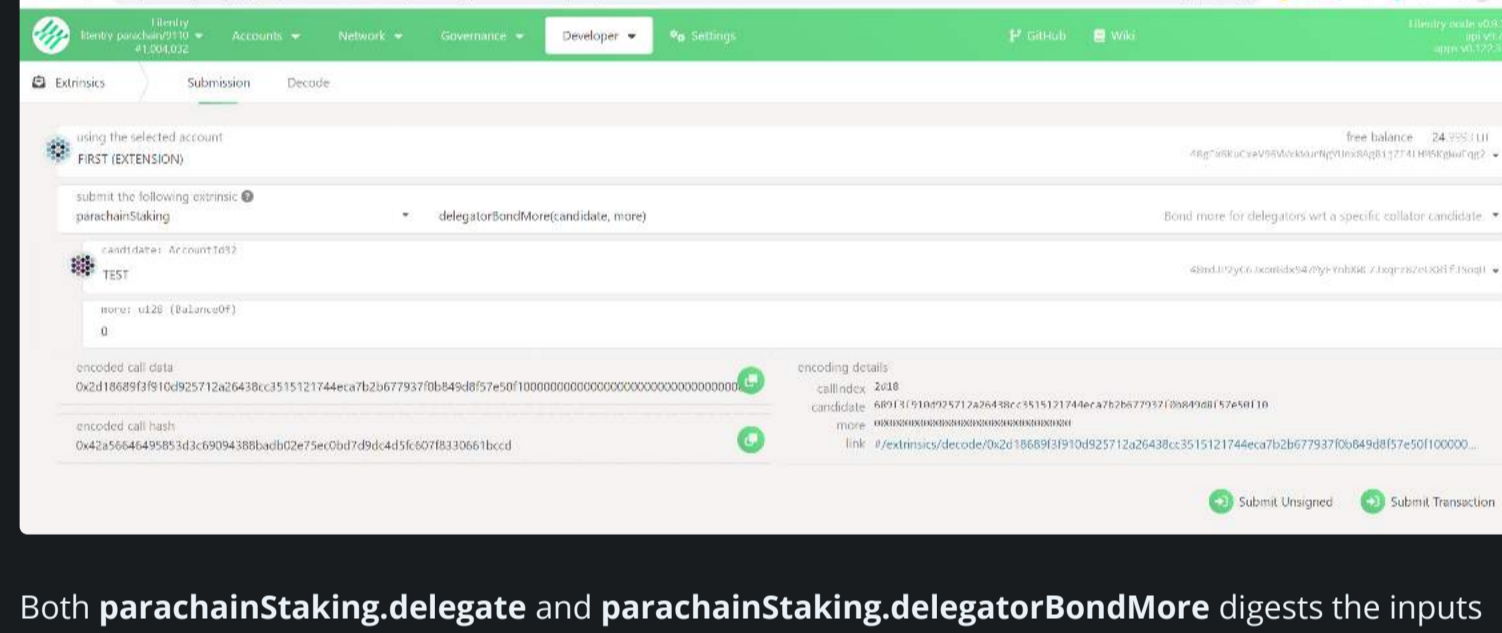
The second option is to use [Polkadot.js](#).

PolkadotJS remains the right point of call if you want to perform any potential Litentry parachain function.

For first-time users that wish to perform delegation staking, you will go through the [parachainStaking.delegate](#) route.



However, if you are an already existing user and intend to perform more delegation staking, you'll use the [parachainStaking.delegateBondMore](#) method.



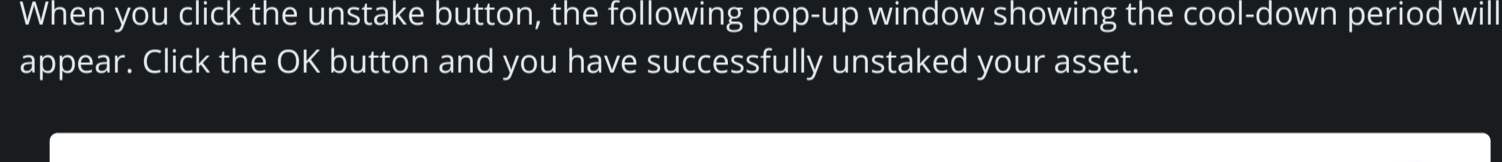
Both [parachainStaking.delegate](#) and [parachainStaking.delegateBondMore](#) digests the inputs of your target collator account address and your target staking amount in 10<sup>12</sup> form. (i.e. If your input amount is 100 0000 0000 0000, it means you want to stake 100 LIT).

## How to unstake

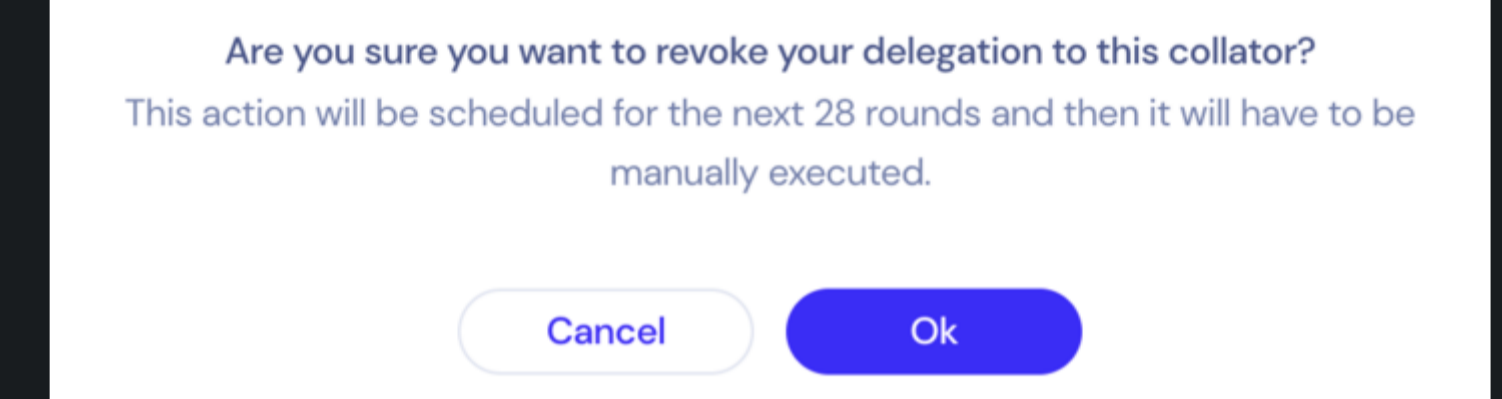
### 1. Cumulon

Upon opening your [staking dashboard](#), you will see two following buttons on the right side of the collators that you staked into:

- Delegate - to add more staking
- Unstake - to unstake the previous staking



When you click the unstake button, the following pop-up window showing the cool-down period will appear. Click the OK button and you have successfully unstaked your asset.



### 2. PolkadotJS

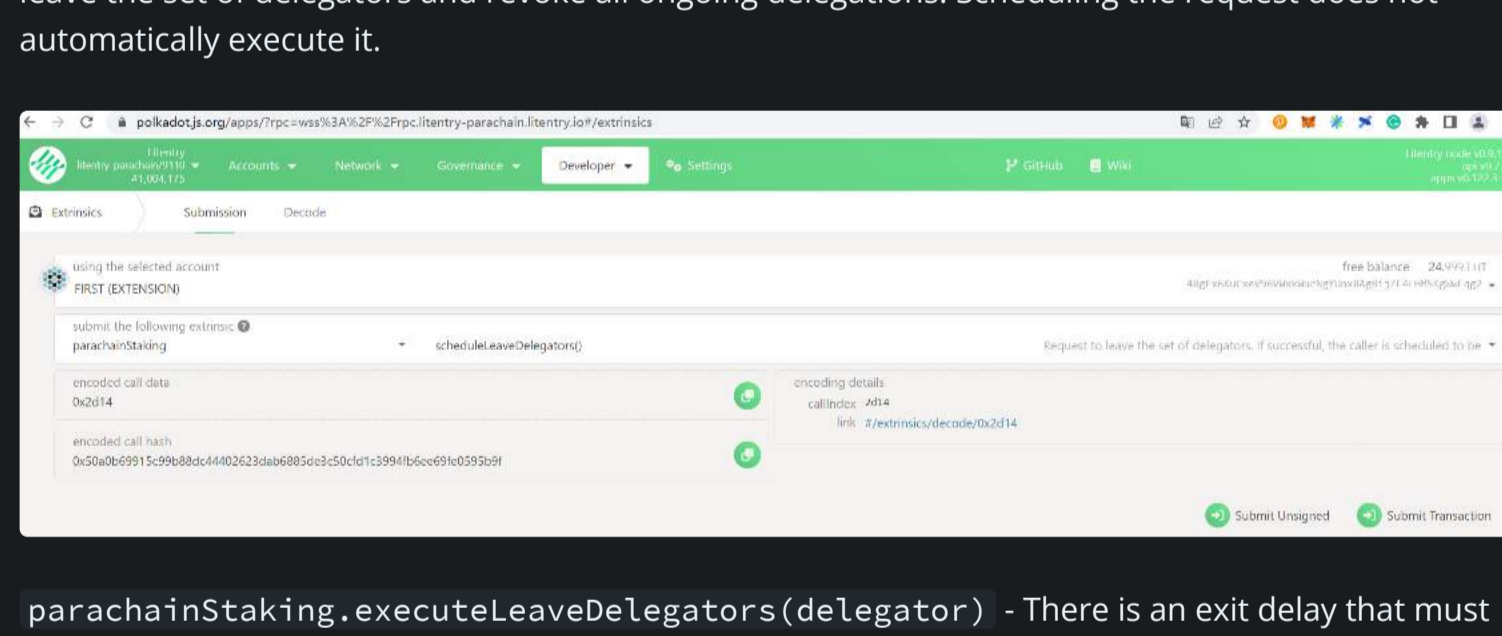
You can also use [Polkadot.js](#) for any unstaking/stake-less action with more technical flexibility.

You can unbond/undelegate your LIT at any time. However, it will take 7 days for your funds to become transferable. However, it is essential to note that, as a result of longer block times due to an issue on the relay chain, unstaking takes a bit longer than 7 days.

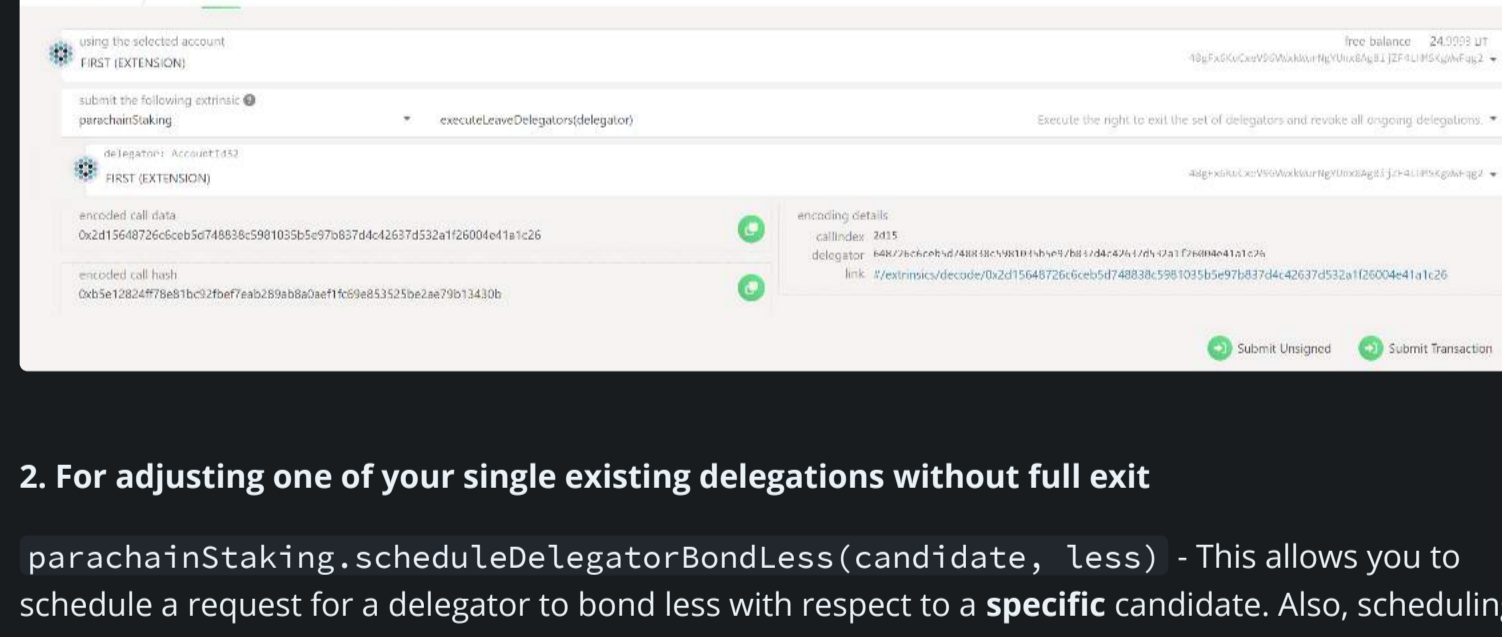
As a user who wishes to unstake his LIT, you need to schedule your unstake/stakeless command and execute it to claim your funds after the cooldown period expires. Below are the different scenarios to unstake your asset.

#### 1. For leaving all of your Collator Delegations at once

`parachainStaking.scheduleLeaveDelegators()` - This allows you to schedule a request to leave the set of delegators and revoke all ongoing delegations. Scheduling the request does not automatically execute it.

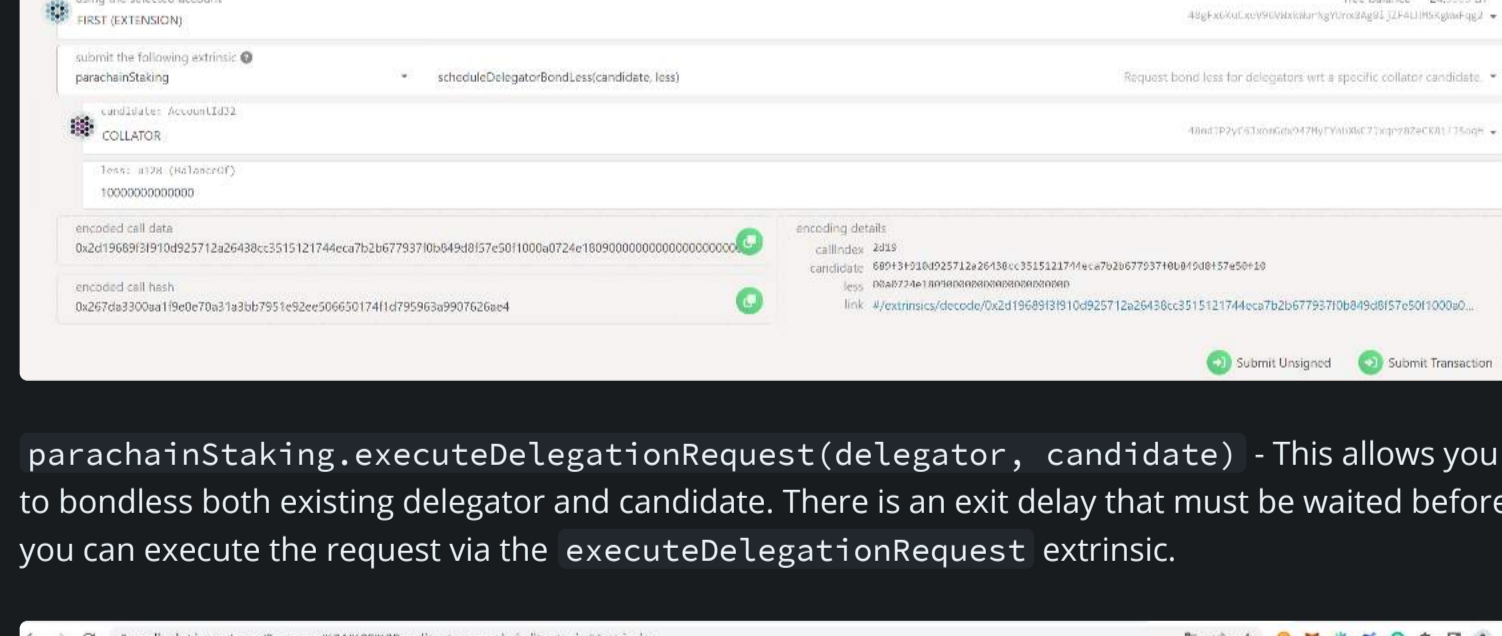


`parachainStaking.executeLeaveDelegators(delegate)` - There is an exit delay that must be waited before you can execute the request via the `executeLeaveDelegators` extrinsic.

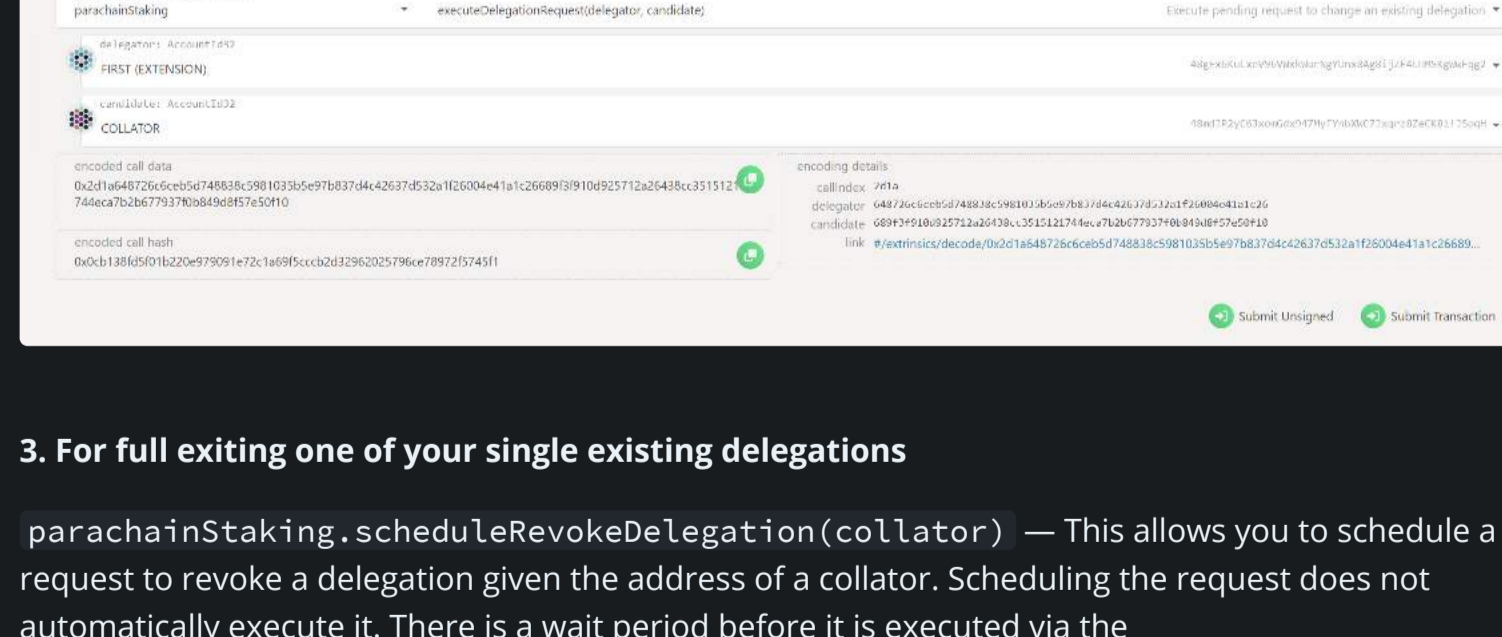


#### 2. For adjusting one of your single existing delegations without full exit

`parachainStaking.scheduleDelegatorBondLess(candidate, less)` - This allows you to schedule a request for a delegator to bond less with respect to a **specific** candidate. Also, scheduling the request does not automatically execute it like in `parachainStaking.executeLeaveDelegators` mentioned earlier.

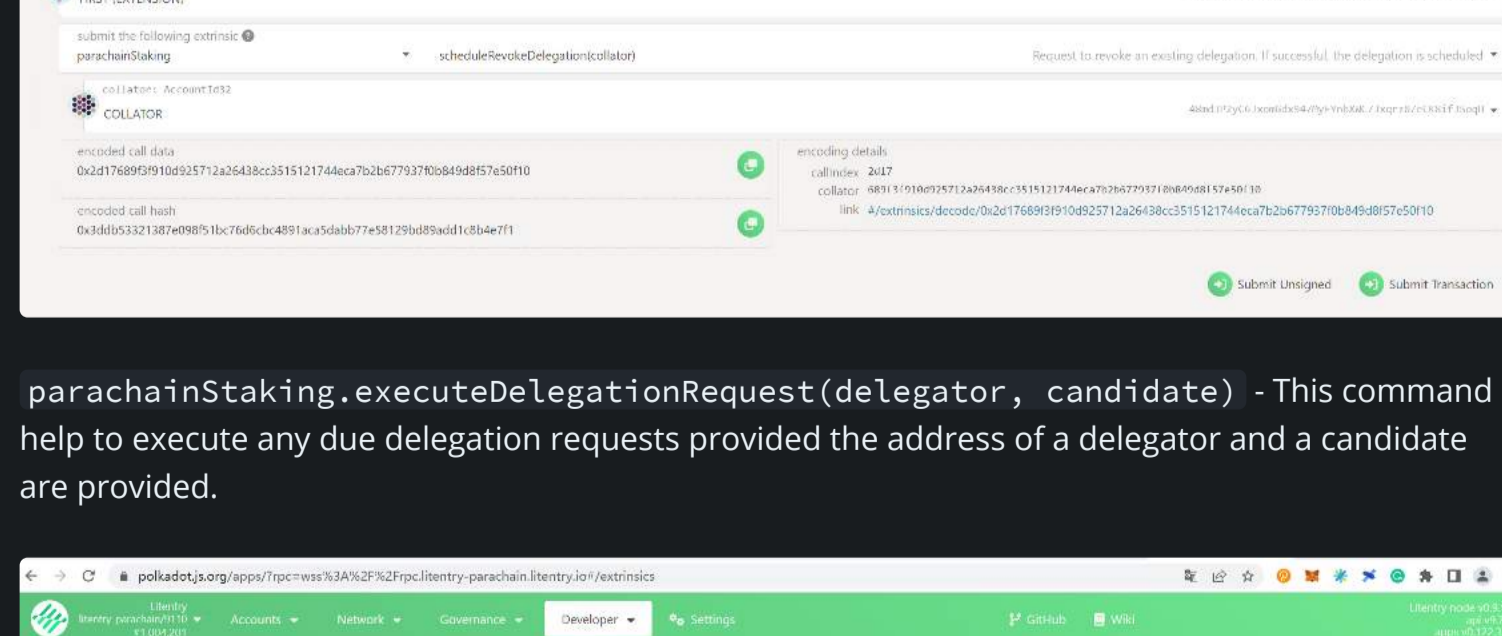


`parachainStaking.executeDelegationRequest(delegate, candidate)` - This allows you to bondless both existing delegator and candidate. There is an exit delay that must be waited before you can execute the request via the `executeDelegationRequest` extrinsic.

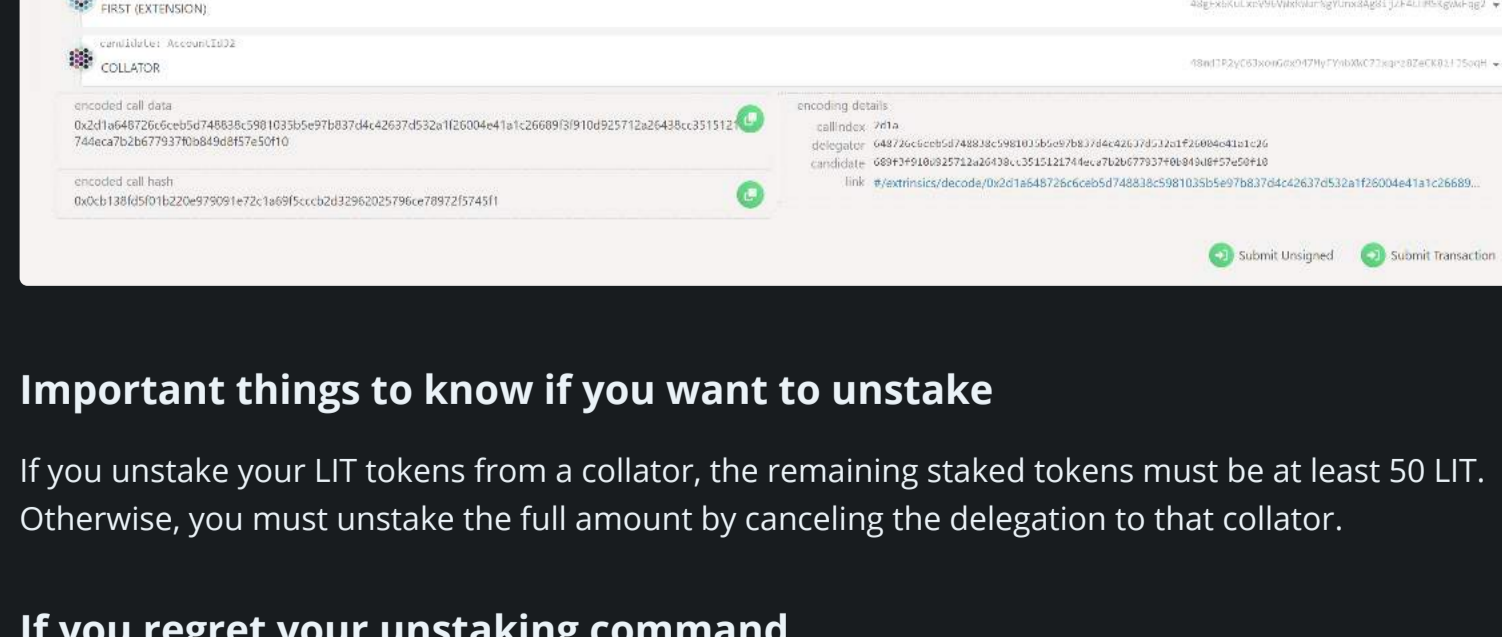


#### 3. For full exiting one of your single existing delegations

`parachainStaking.scheduleRevokeDelegation(collator)` — This allows you to schedule a request to revoke a delegation given the address of a collator. Scheduling the request does not automatically execute it. There is a wait period before it is executed via the `executeDelegationRequest` extrinsic.



`parachainStaking.executeDelegationRequest(delegate, candidate)` - This command help to execute any due delegation requests provided the address of a delegator and a candidate are provided.



## Important things to know if you want to unstake

If you unstake your LIT tokens from a collator, the remaining staked tokens must be at least 50 LIT. Otherwise, you must unstake the full amount by canceling the delegation to that collator.

### If you regret your unstaking command

Your tokens will not accrue rewards while unstaking is pending. You can cancel unstaking any time during the 7 days.

The following method is for canceling your unexecuted command:

`parachainStaking.cancelLeaveDelegators()` for condition 1 above - With this, you can cancel a pending scheduled request to leave the set of delegators.

`parachainStaking.cancelDelegationRequest(candidate)` for condition 2,3 above - This command help to cancel delegation request for all delegations.

# Auto-compound staking

Explaining Litentry Auto-Compound Staking feature

Users help to maintain the Litentry network by staking their LIT tokens to delegate collators through the parachain staking pallet. By design, staking rewards are automatically deposited into the users' wallets and the deposited token is available for the user to use based on their discretion.

However, Litentry being a protocol that is inclined towards increased capital efficiency and better utilization rate of assets for our users recently launched the auto-compound staking feature.

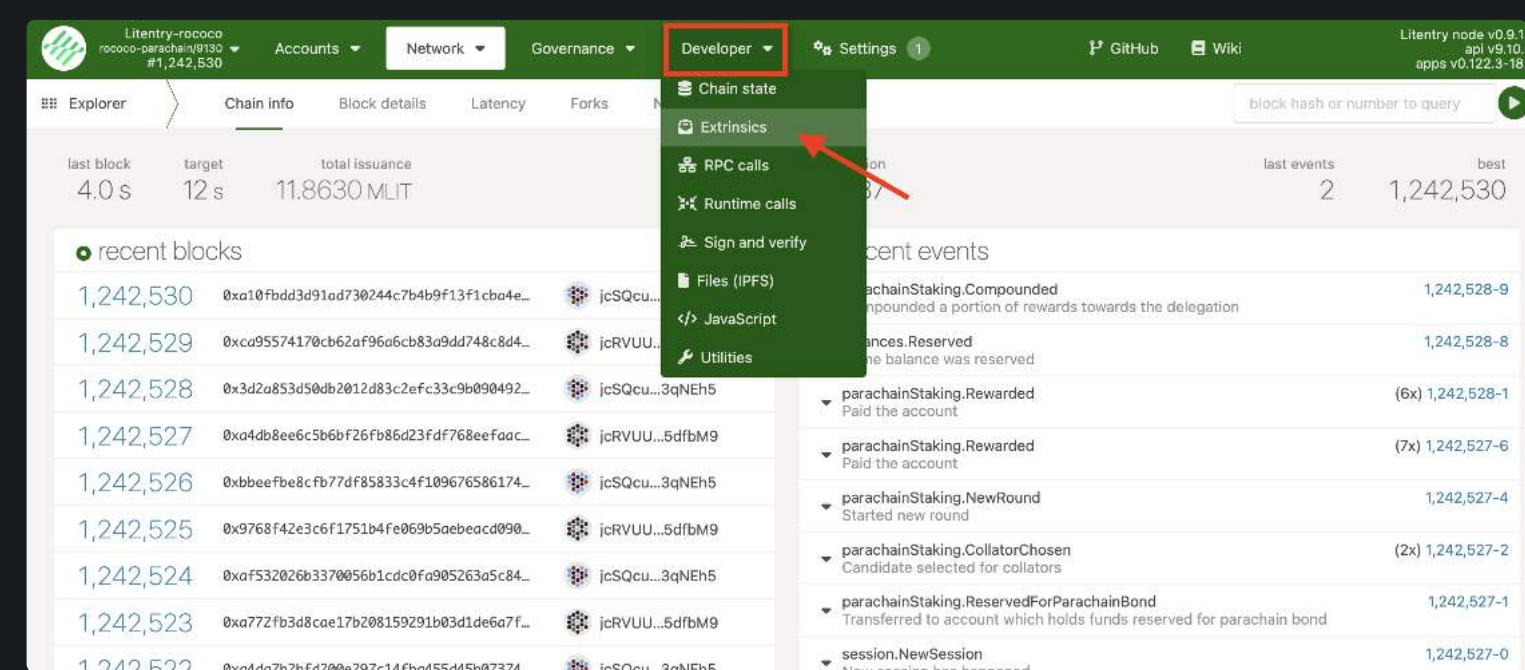
With this feature, the staking rewards earned by a user is automatically returned to the staking pallet at the click of a button so that the user can continue to earn more reward. The user determines the total amount (percentage) of the reward that is re-staked into the corresponding reward-generated staking position.

You may follow the guide below to explore the auto-compound staking feature.

## Polkadot.JS

The auto-compound staking feature is only available on Polkadot.JS for now.

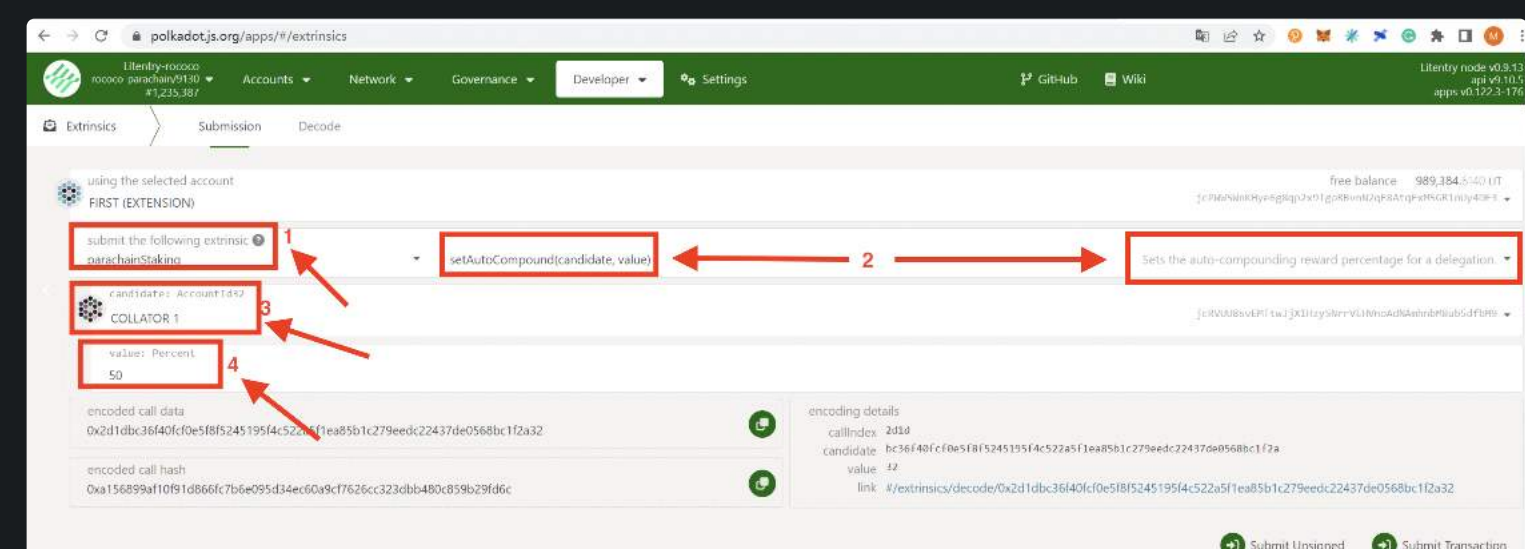
To get started, hover your mouse on the Developer button at the top of the page and select 'Extrinsics' from the drop-down options:



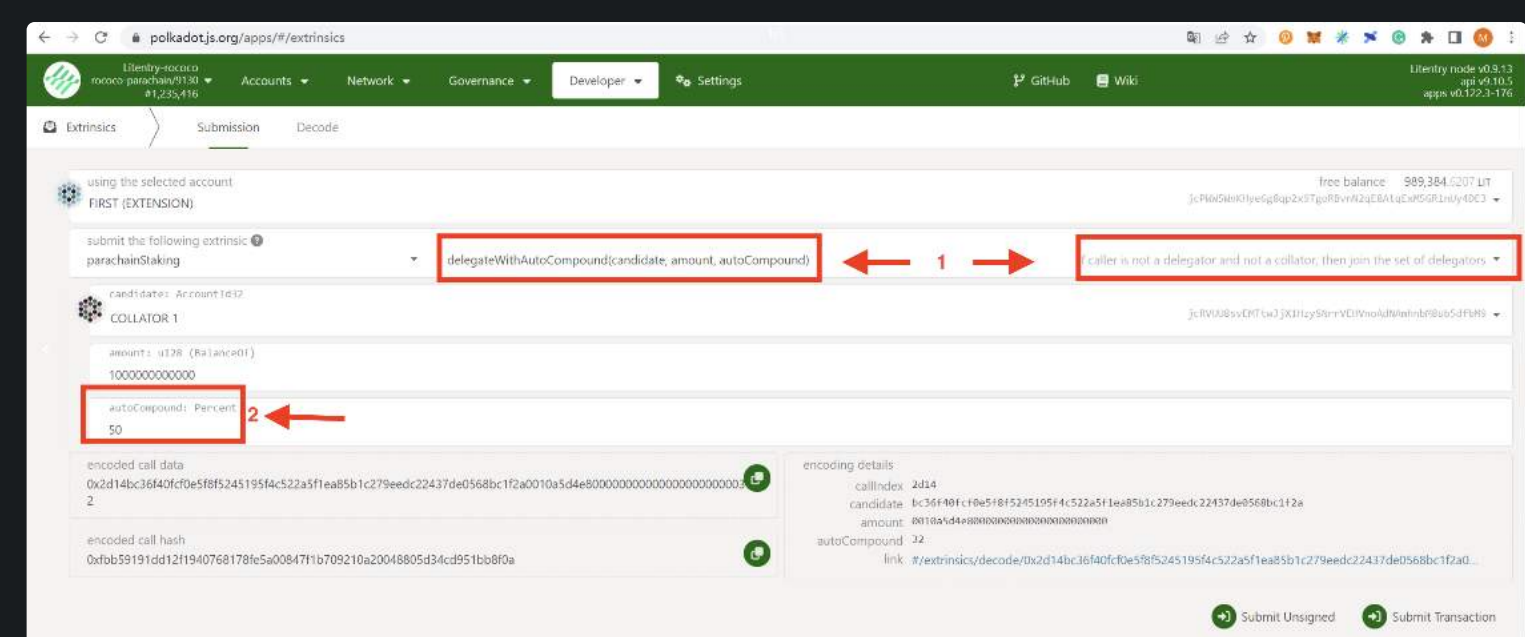
Once selected, choose `ParachainStaking` from the 'Submit the following Extrinsics' drop-down options as shown in 1 below.

By method, select `setAutoCompound` from the 'Add white list of candidates' drop-down options as shown in 2 below. It is important to note that the delegator's staking positions on every collator are separate. As such, you should continue if there is an existing position for you on the corresponding as shown in 3 below. Finally, set the percentage of your reward you'll like to re-delegate to the collator in the staking pallet as shown in 4 below and submit the transaction.

Based on the information provided in the extrinsics below, the user's (delegator) auto-compounded setting on Collator 1 will be 50% of his reward. It is effective immediately at the next reward distribution process and half of the reward will be automatically re-staked.



You may also directly stake a new position with a specified auto-compound staking setting if there is no existing position for you on the corresponding Collator. The method to select here under the 'Add white list of candidates' is `delegateWithAutoCompound`. After this, select your desired auto-compound percentage and submit the transaction. This will open a new staking position for you just like the traditional `delegate` method but with an auto-compound staking setting.



## Cumulon

As earlier mentioned, the auto-compounded staking setting is only available on Polkadot.JS and is currently not supported by Cumulon. We will update this documentation once the option is available.

# Identity is fragmented.

## Litentry Kusama Registrar Guide

A step-by-step guide on how to verify your identity with Litentry (Kusama) registrar.

### Introduction

While the anonymity that Web3 provides has a great advantage in user privacy, revealing partial personal information can help gain a higher reputation and trust in the Polkadot ecosystem. This document introduces a registrar service that is fully automatic and leverages cryptographic design to eliminate human interventions. The Litentry registrar focuses on providing judgment for users display name, email, and twitter while preserving user privacy. For more information about the registrar, visit this page.

In this treatise, we will walk you through the identity verification process step by step:

To get started, users need to set their identity information onchain; then, they may request the registrar to verify the identity. Users will enter a maximum fee they are willing to pay for the service. After that, the dedicated registrar can ascertain the identity.

### Step 1: Set an on-chain identity

- Click the 'Accounts' tab on the [PolkadotJS Apps](#)
- Make sure you're in the Kusama network by looking at the top left corner of the page. If not, click on the toggle to switch
- On the accounts list, click on the hamburger icon on the right side of your selected account and click 'Set on-chain identity'.

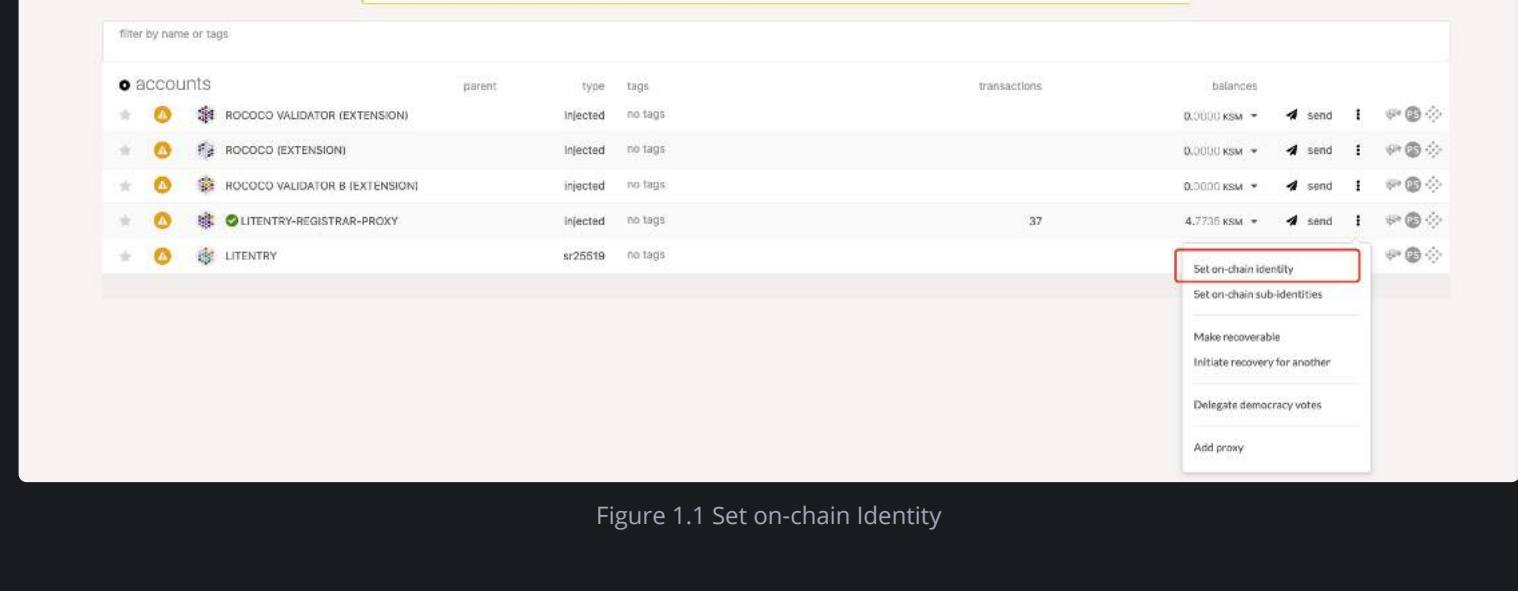


Figure 1.1 Set on-chain identity

- You'll see a popup window of 'register identity'. Click to turn on the include field of display name, email, and Twitter, and enter your information. Once done, click Set Identity to submit the transaction.

Figure 1.2 Submit Identity information

Now you have successfully submitted an identity! Since your information is not verified yet, you will see a profile icon next to your username.

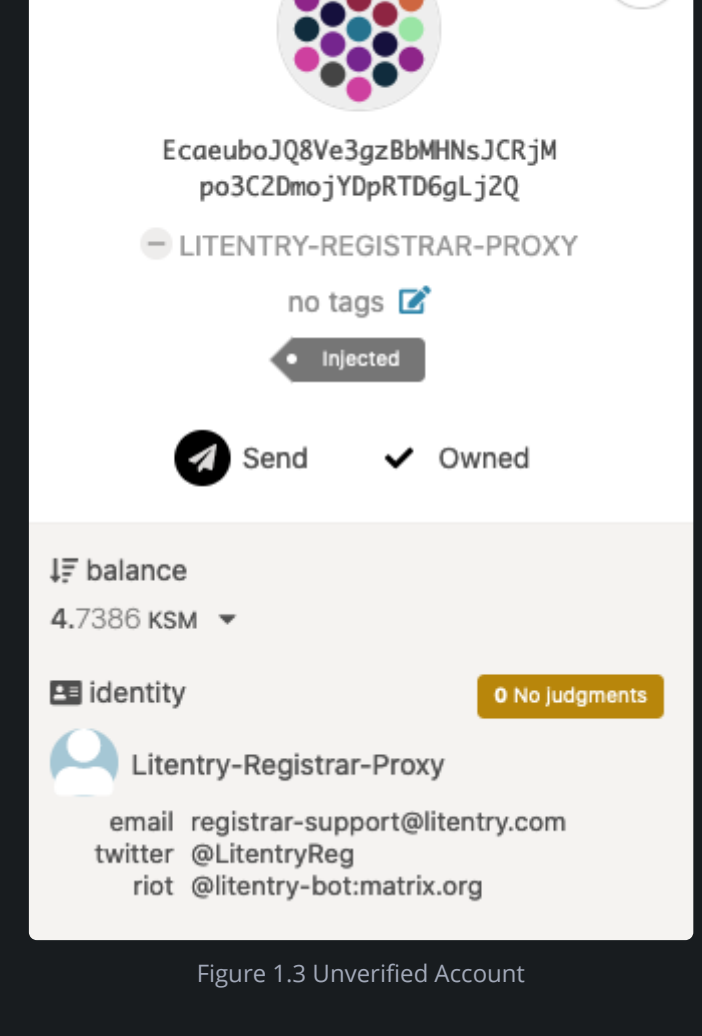


Figure 1.3 Unverified Account

### Step 2: Judgement Request

To request the registrar to validate your on-chain information:

- Go to Developer->Extrinsic, and select your account.
- Select identity under submit the following extrinsic, and requestJudgement (reg\_index, max\_fee) transaction.
- Enter '4' for reg\_index (index of the registrar)
- Enter '0.04' KSM for the service fee

As shown below:



Figure 1.4 Judgement Request

- Click Submit transaction and you have successfully requested the registrar to validate your on-chain information.

### Step 3: Email Verification

You should receive a verification email from Litentry. Click on "Verify Email Now" to complete the verification process (see figure 1.5). After that, you will receive another email that confirms that the verification is successful.

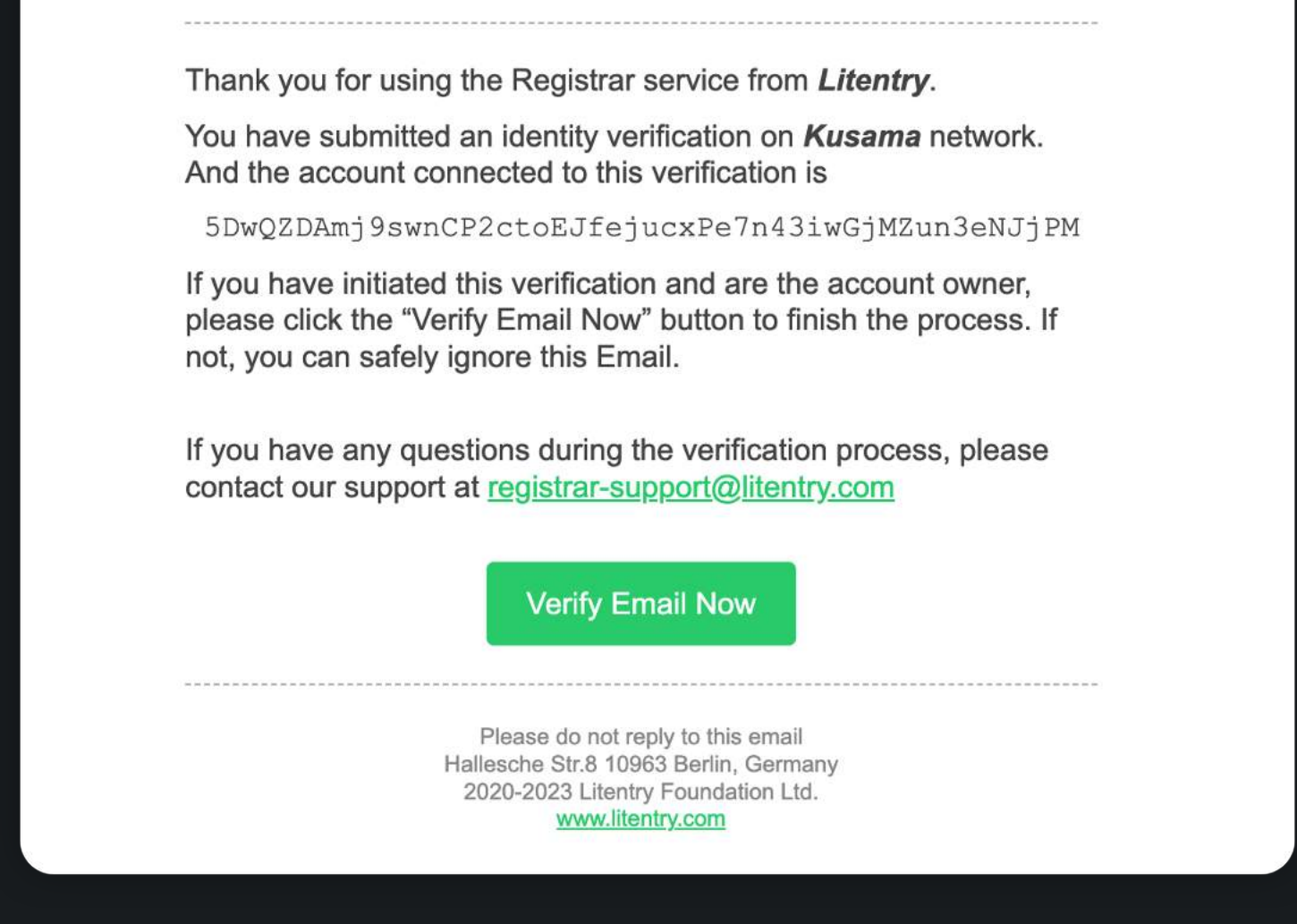


Figure 1.5 Email Verification Example

### Step 4: Element Verification (Optional)

- An invitation will be sent from "litentry-bot" on Element, accept the invitation
- Click on the verification link from "Litentry-Bot" to complete the verification of the element account. Once the verification process is completed, you will receive a confirmation message (see figure 1.6).

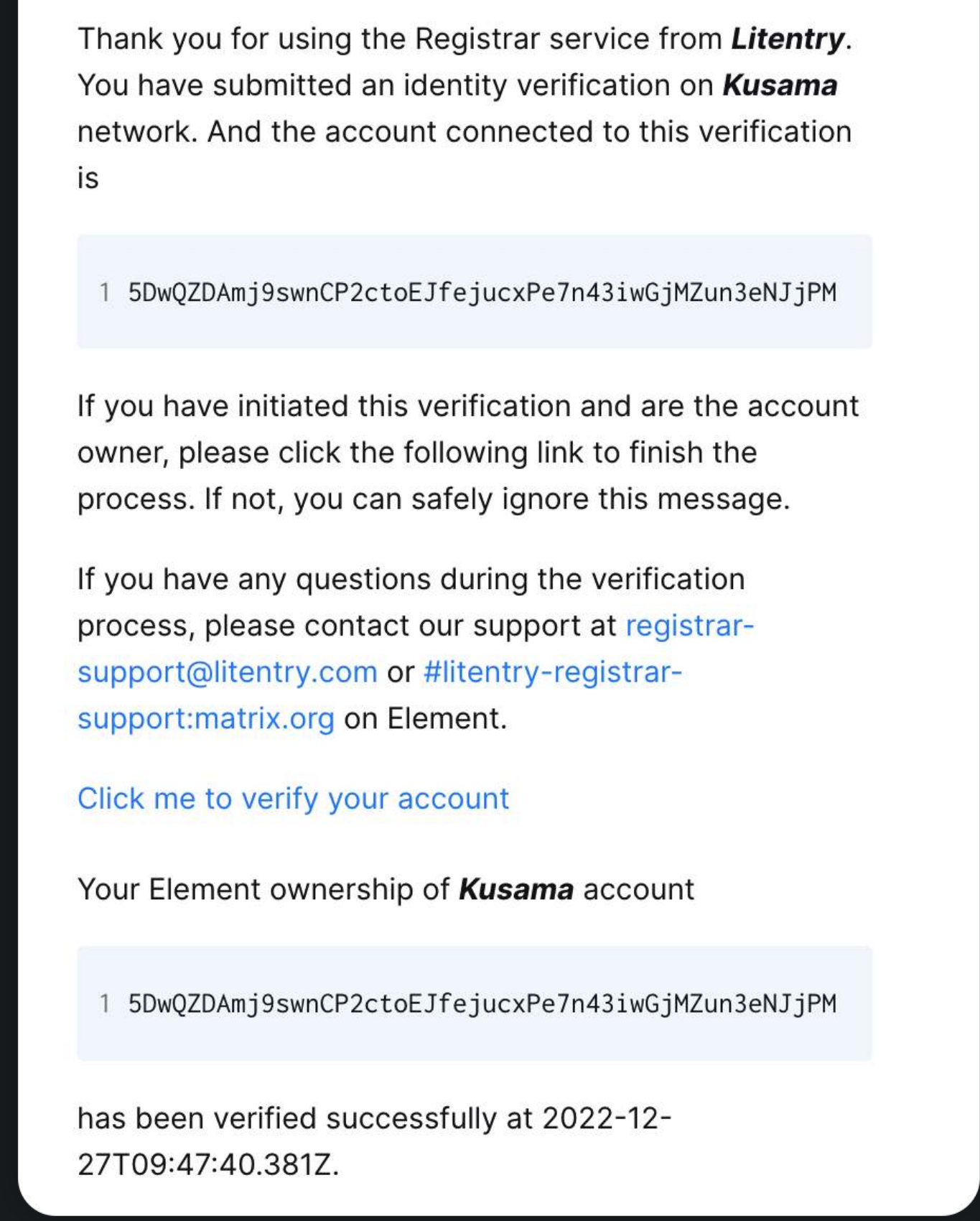


Figure 1.6 Element Verification Example

### Step 5: Twitter Verification (Optional)

- Follow @LitentryReg on Twitter
- Make sure your account is open to private messages in your privacy settings. Otherwise, the verification message will not go through.
- You'll receive a verification link on DM from @LitentryReg. Click on the link to complete the verification of your Twitter account. Once it is completed, you will receive a confirmation message (see figure 1.7).

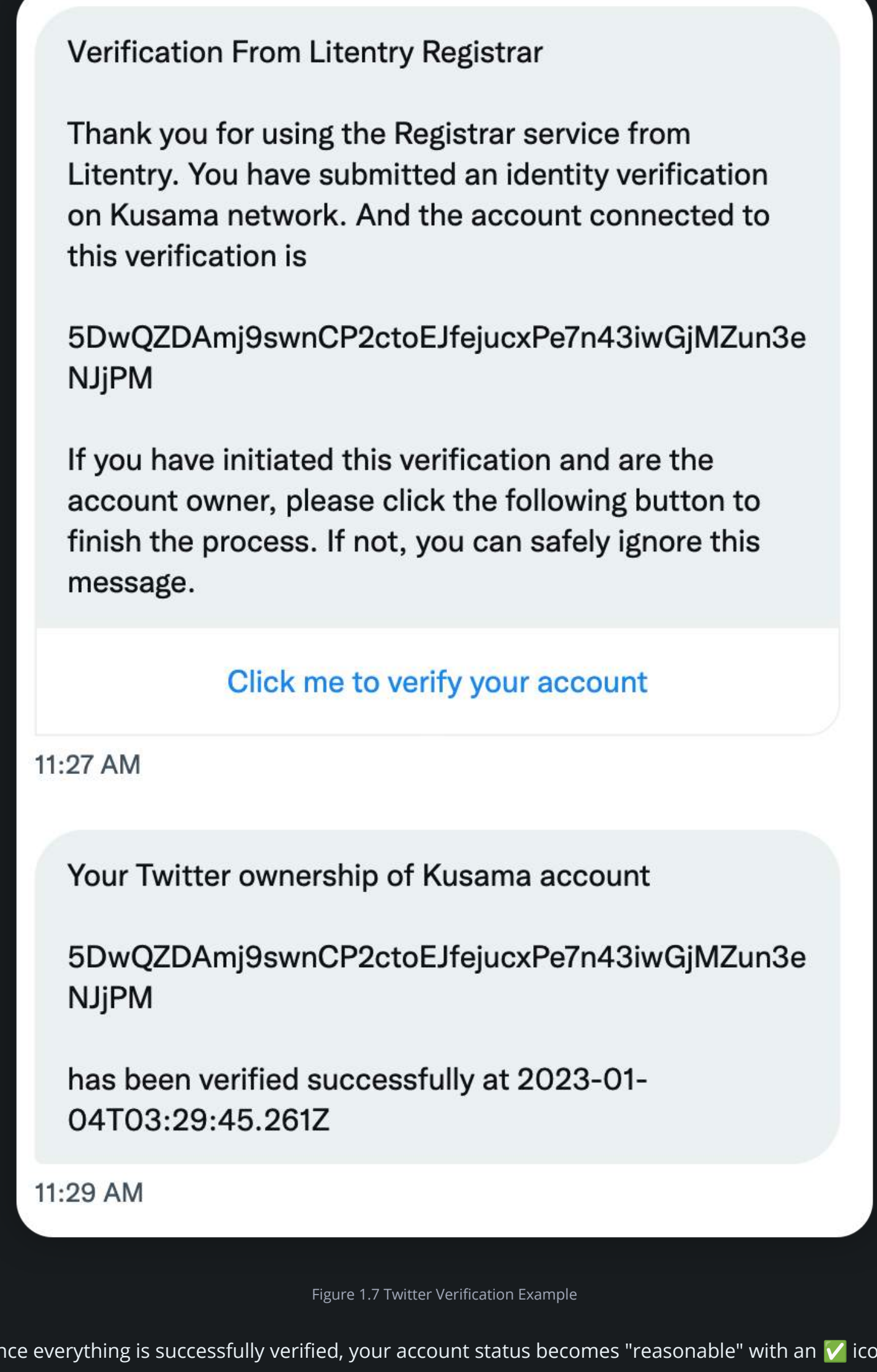


Figure 1.7 Twitter Verification Example

Once everything is successfully verified, your account status becomes "reasonable" with an icon. Congratulations - Your identity is now verified on PolkadotJS Apps!

### Registrar Fee

The registrar fee of the Litentry Registrar is 0.04KSM.

It's important to notice that no KSM is sent to our registrar at any time before the judgment is completed. You should NOT send funds to our account directly. When calling requestJudgement, the registrar fee will be locked and put aside. It will be transferred to the registrar only after the registrar finishes its job.

### Support

If you have any questions, contact us via Element Room or registrar-support@litentry.com

# Participate in Litentry Governance

Tutorial on how to Participate in Onchain and Offchain Litentry Governance

## Introduction

Governance is one of the core features of decentralization and an important mantle of the Litentry protocol. It allows our community members to discuss issues, ideas, and proposals that they care about publicly and transparently and give them the power to make important decisions over the development of the protocol using on-chain and off-chain governance.

**On-Chain Governance** is powerful because it can directly change the code of the Litentry Runtime and other storage. To ensure the quality of the governance proposal, proposers are required to deposit tokens to propose a referendum. Thus, it requires proposers to have a thorough understanding of their proposal, as well as to gain relatively broad support from other community members.

**Off-Chain Governance** has a low threshold so that it is more accessible and encompassing. Off-chain decision-making can not only help a proposal gain support before going on-chain, but it can also cover issues that shall be executed on a social level.

This guide will help you to understand how to participate in the two types of Litentry protocol governance.

## On-Chain Governance

With on-chain governance, you can propose making an Extrinsic Call that changes a specific part of the code of the Litentry Runtime. There are different sections of extrinsic on Litentry:

- `assetManager`
- `bridgeTransfer`
- `chainBridge`
- `collatorSelection`
- `democracy`
- `identityManagement`
- `tokens`
- And others, please see [polkadot.js - Litentry - Developer - Extrinsic](#)

## On-Chain Governance Process

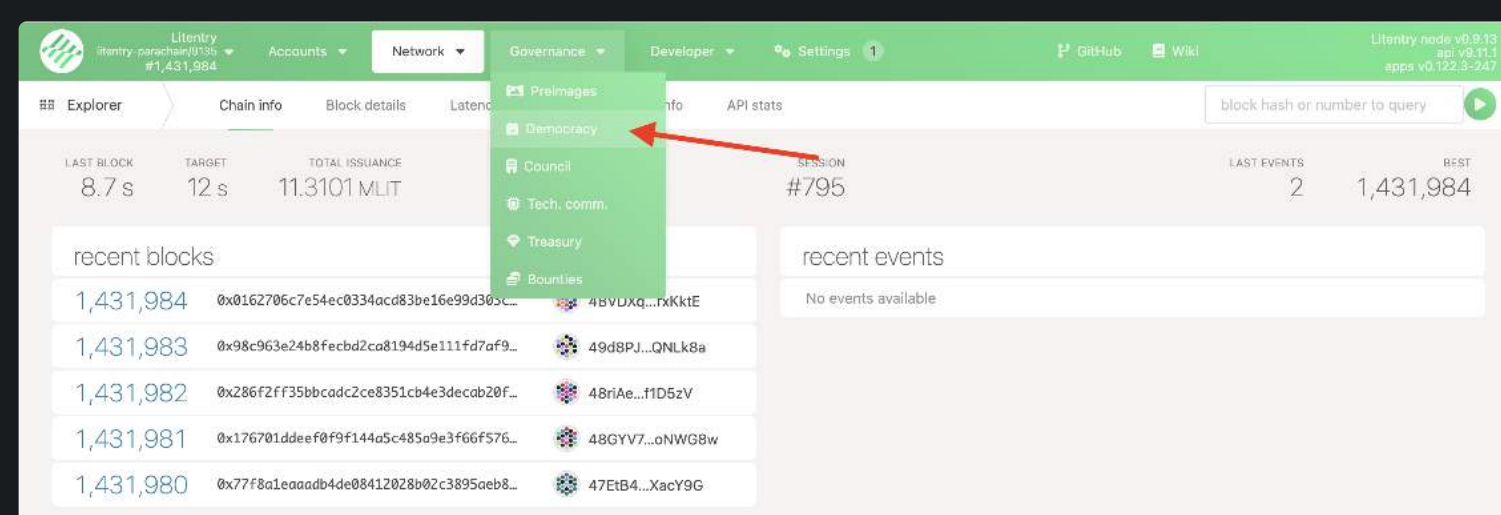
### Phase 1: Idea

Anyone is welcome to propose an initial idea and create a poll using **off-chain Governance**. If an idea receives support from other people, it can be proposed in a Referendum.

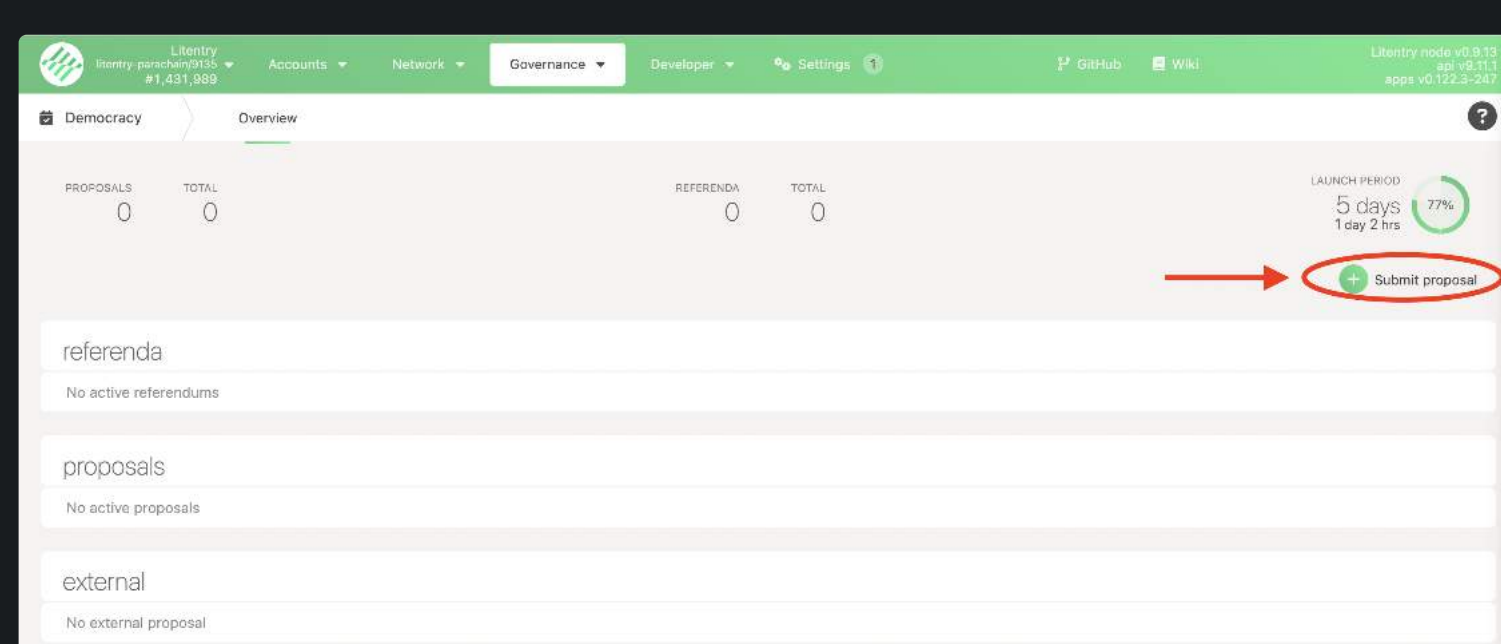
### Phase 2: Propose

What you'll need is a Litentry Wallet and some LIT

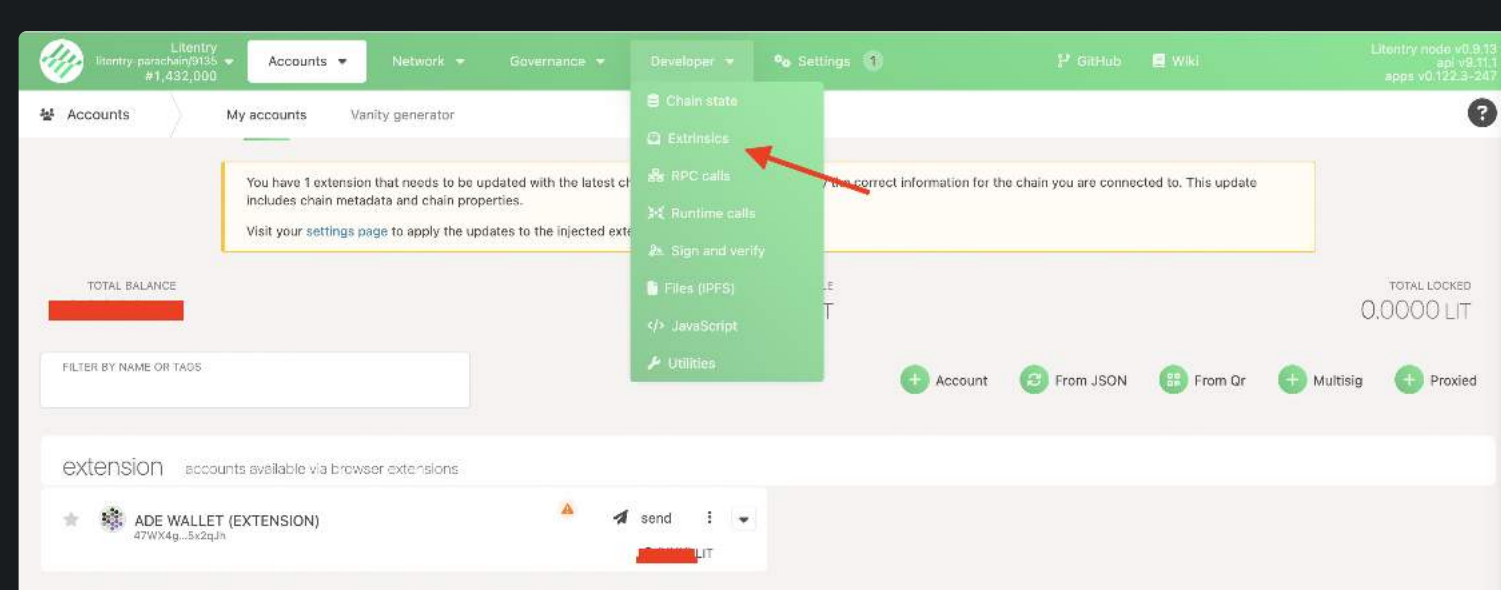
- Once you're ready, submit a proposal at **Democracy** on Polkadot.js by hovering your mouse on the Governance tab and clicking Democracy from the drop-down options.



Next, you click the Submit Proposal button as shown below:

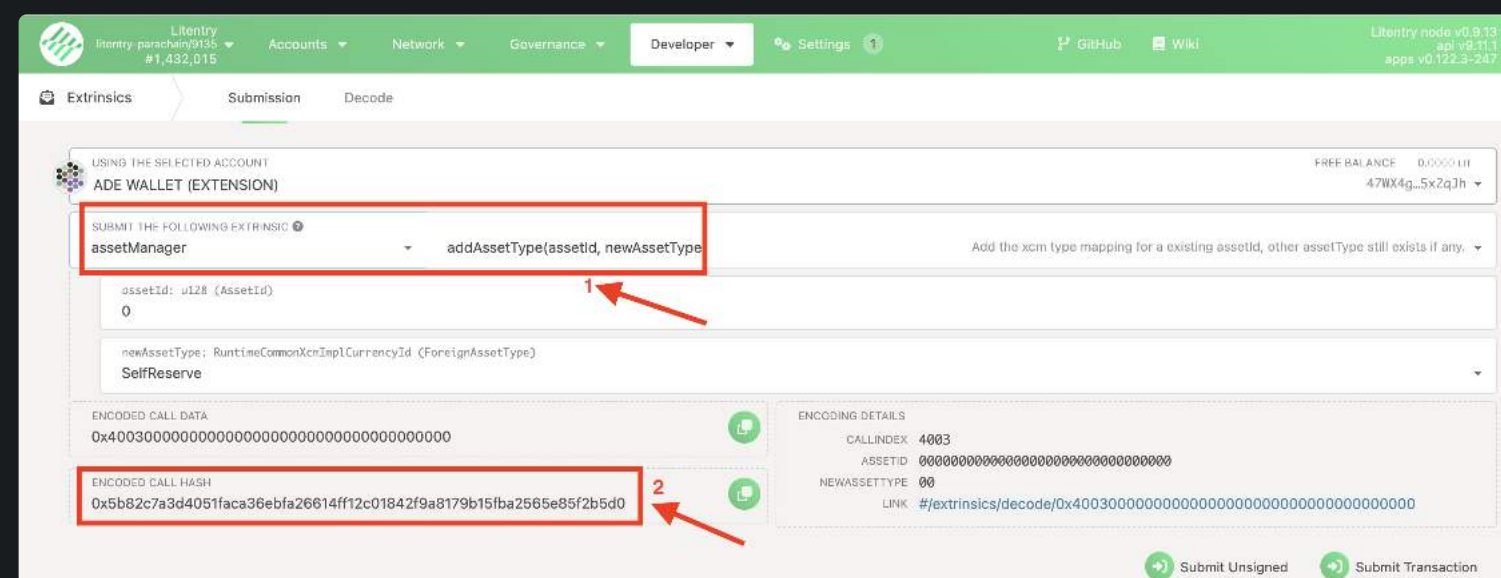


Upon clicking the submit proposal button, find the *preimage hash* that corresponds to the extrinsic call that you wish to make, depending on which section of code you are proposing changes to. You can find a preimage hash for a specific Extrinsic Call at **Extrinsic**.

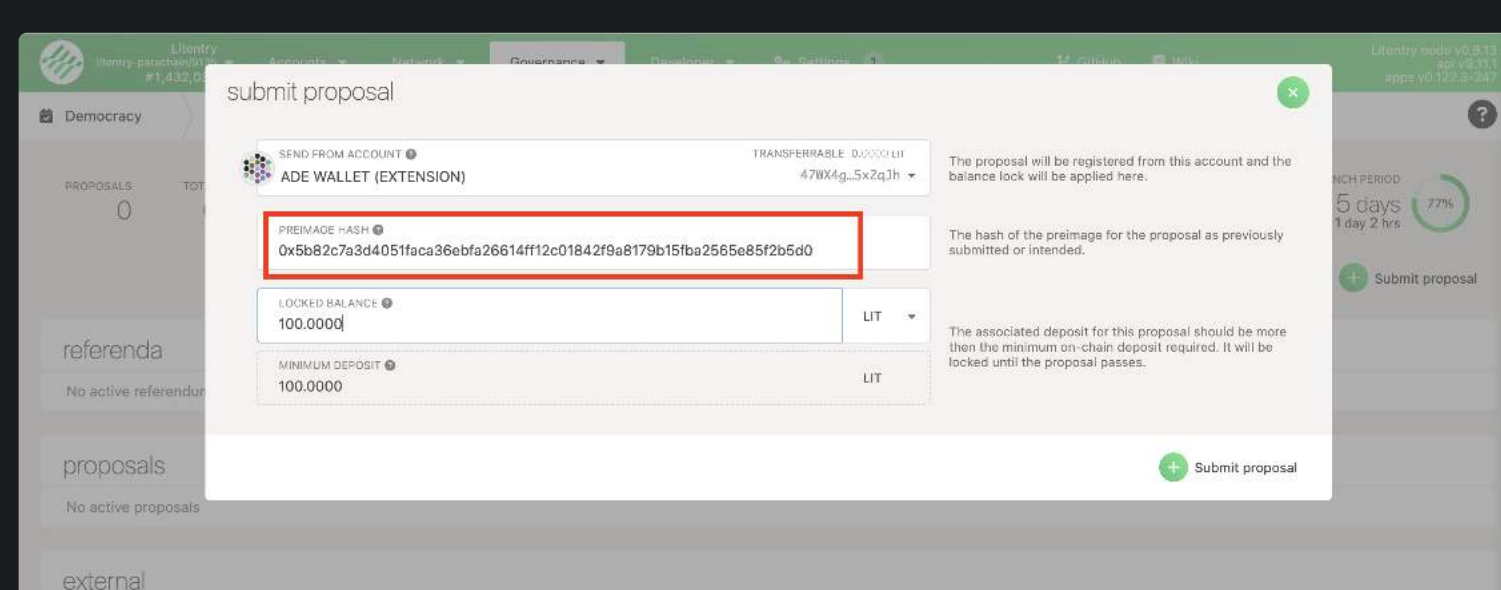


Extrinsics in the Developer menu

Once in Extrinsics, select your desired Extrinsic call by selecting an option from the drop-down in the part labeled 1 below. Then copy the Encoded Call Hash as shown in the part labeled 2.

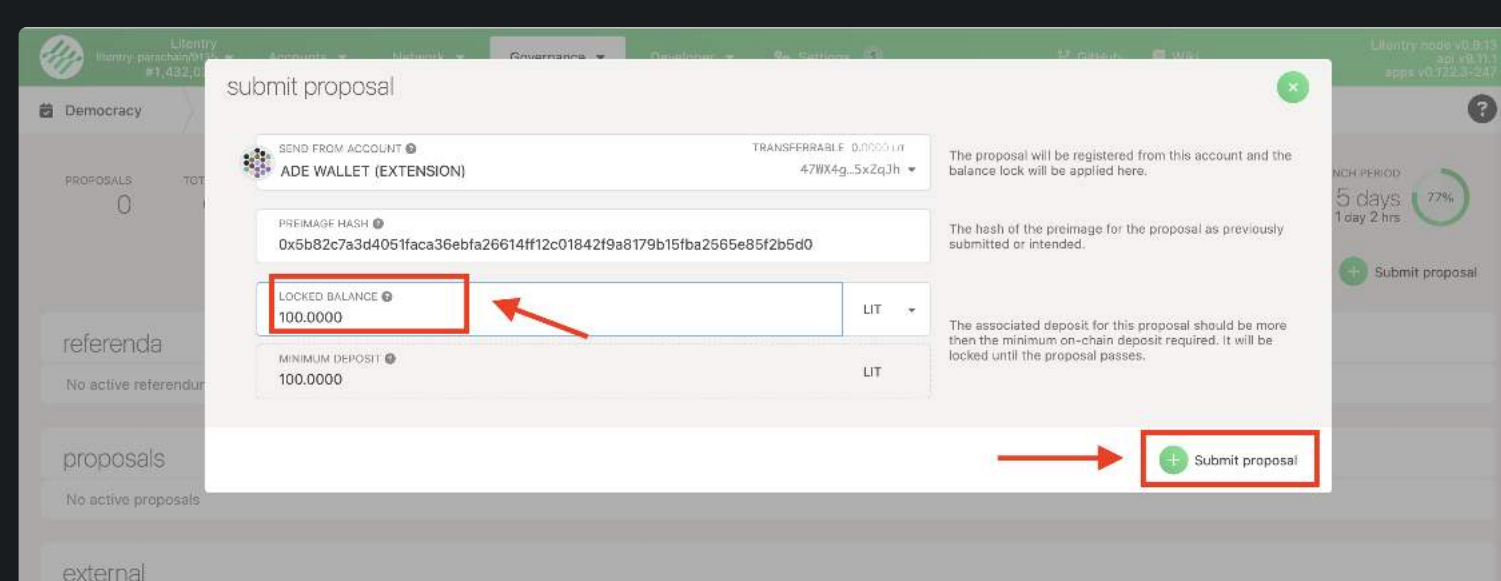


Next, go back to your submit proposal page and past your copied hash in the Preimage hash box



Finally, enter your required LIT balance and click the submit proposal button.

Kindly note that 100 LIT are required to submit your proposal. Also, check if you have sufficient balance to pay the gas fee (around 1 LIT per tx).



Once your proposal is submitted, change the text description of your proposal. It's important to write a strong rationale for others to endorse and support your proposal. If your proposal gets highly bonded support, it moves into a referendum.

### Phase 3: Vote

Once a proposal runs into a referendum, it will start an on-chain voting process where all LIT holders can stake to vote during a 5-day period. If your referendum passed, it will automatically fulfill and change the code in the Litentry Runtime.

## Off-chain Governance

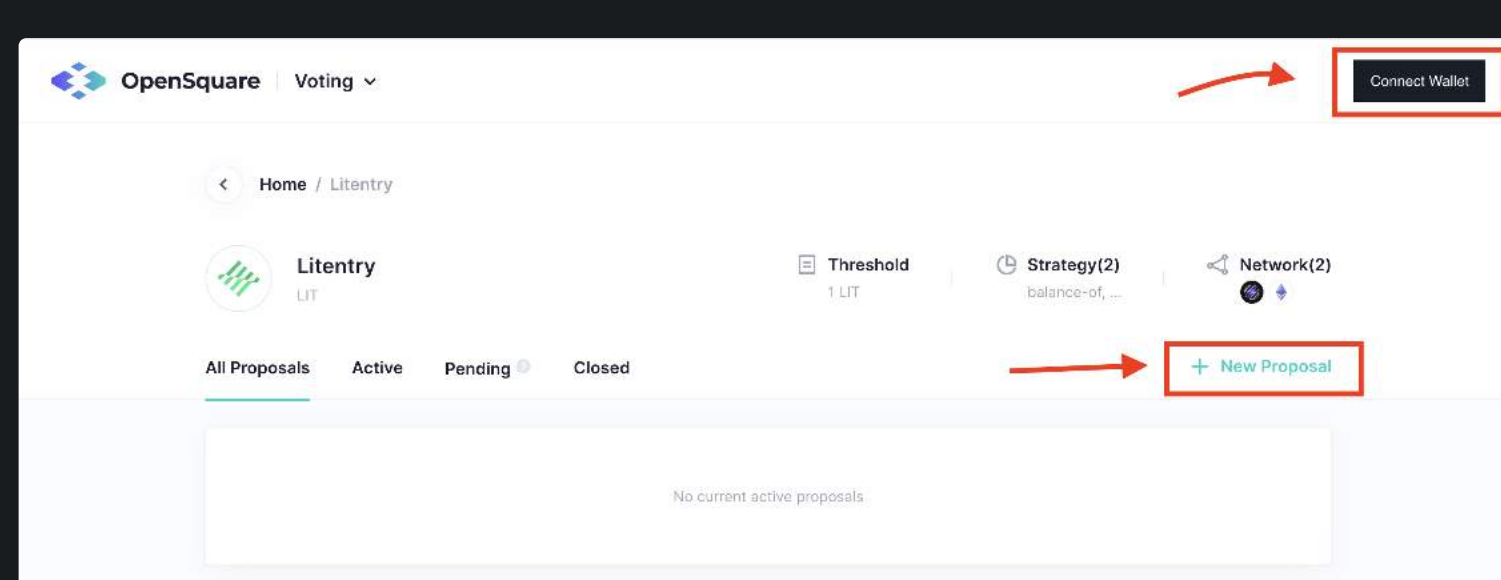
### Issues

At the social level, the LIT community can govern over issues in the following aspects:

- Tokenomics and slot auction campaigns
- Project partnerships with Litentry products
- External project incubation and onboarding to the Litentry parachain
- Community growth strategies, events organization, and technology avocation

### Off-Chain Governance Process

Litentry is partnering with **OpenSquare**, which provides us with a useful tool to raise off-chain governance proposals and start a voting process. Users can go to **Litentry Voting on OpenSquare** and create a new proposal. You can use an ETH wallet instead of a Litentry wallet to participate. Creating proposals require you to have at least 1 LIT token in your wallet, but you don't need to deposit or stake any token to submit a proposal.



Obtaining LIT tokens is the first step to getting involved in on-chain governance.

You can purchase LIT on the following platforms (skip this if you already have LIT):

- Uniswap
- Binance
- Gate.io
- KuCoin
- And more please refer to [this list by Coinmarketcap](#).

Next, transfer LIT from an ETH wallet to a Litmus Wallet. Go to [this tutorial](#) to see how.

If you do not have any LIT, you can still participate in soft governance using the **Discussion** and **Comment** feature on Litmus Governance Platform.

# Establishing XCM communication with Litentry

Horizontal Relay-routed Message Passing (HRMP) bi-directional communication between Litentry and other Parachains.

Cross-Consensus Messaging (XCM) Format was first mentioned in Polkadot's documentation. It evolved from Cross-Chain Messaging Format to become a robust architectural style of message transfer for more than chains but also pallets, smart contracts, bridges, and sharded enclaves. XCM is not a protocol. As such, it cannot parse messages across consensus by itself. Rather, it is a format for how message transfer should be performed and a language for ideas between consensus systems.

The architecture of Polkadot and Kusama allows the interoperability of parachains, enabling the cross-chain transfer of assets and data. To achieve this, an XCM format defines the language around how the message between two chains should be parsed. Since XCM is now available on Polkadot and Kusama, **Litentry/Litmus** being Parachains of the two relaychains have now opened XCM channel for any parachain in the ecosystem that wishes to partner with us.

The process involves gaining access to bi-directional communication between us and the parachains. This involves opening an outbound and accepting inbound Horizontal Relay-routed Message Passing (HRMP) channel by the two parachains.

The following is a quick summary for partners wishing to establish XCM Channel with Litentry/Litmus.

Parachain	Parald	Sovereign account on Relay	Sovereign account on Sibling Parachain
Litentry	2013	5Ec4AhPXyUNLcP3QNT7e6bKwKz9Xx2Svdgi7egBLzMwoXqcD	5Eg2fntLdCZELQmYvT4jDEfkaTpqf5W2hLMYudhqs5yNPMzV
Litmus	2106	5Ec4AhNxtC4V4TecEorgRt2LJdNhRifAeGQvXFVjXN9dWk	5Eg2fnsmxvFPC4NknoomyFmhGoyw67UpipHft3kPTYw17UP

Below is the Litentry/Litmus Native Token LIT Multilocation information:

Multilocation {parents:1, interior: [Parachain: *parachain\_id*, PalletInstance(10)] }

or

Multilocation {parents:0, interior: [PalletInstance(10)] }

And the current weight we charge for each individual XCM command:

```
UnitWeightCost: XcmV2Weight = 200_000_000u64;
```

(In Litentry/Litmus, 1 LIT in the real world= 1\_000\_000\_000\_000 Weight)

## How to Establish XCM Channel with Litentry/Litmus.

The [XCM Fungible Asset Implementation Guide](#) by Open Web3 Stack - the common-good collection of libraries to accelerate application development on Substrate has explained cross-consensus fungible asset design discussions and considerations. It also provided detailed information regarding orml-xtokens that Litentry/Litmus and many other parachains have adopted and currently testing.

Each parachain has its native token and parachains that want seamless communication between them and other parachains in terms of asset transfer, [orml-xtokens](#) can be used as a reference implementation in this regard. Follow the steps below to establish XCM channel between your parachain and Litentry/Litmus:

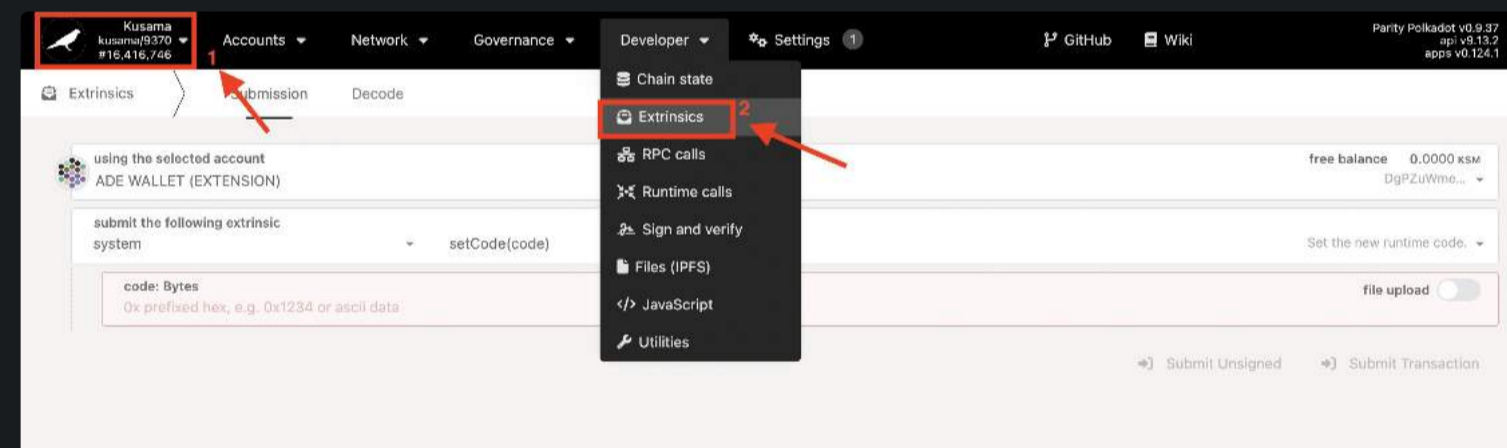
## Step 1: Communicate with Litentry Team and submit your info

We will need the exact same information as the summary provided above. Please connect with the Litentry team via [Telegram](#).

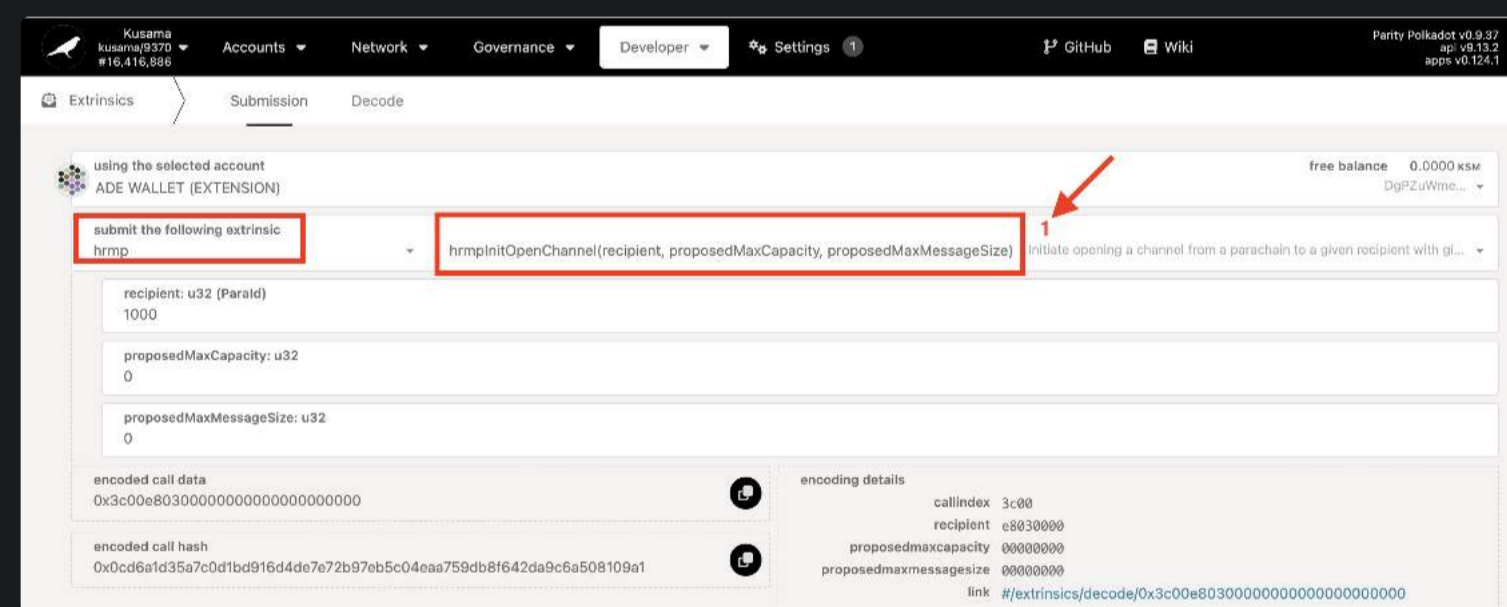
## Step 2: Sending/Accepting HRMP Open Channel Request

Next, both partner parachain and Litentry/Litmus parachain need to send two XCM requests (Open and Accept) to Relaychain. Once the transactions are mutually matched, the HRMP channel will be established.

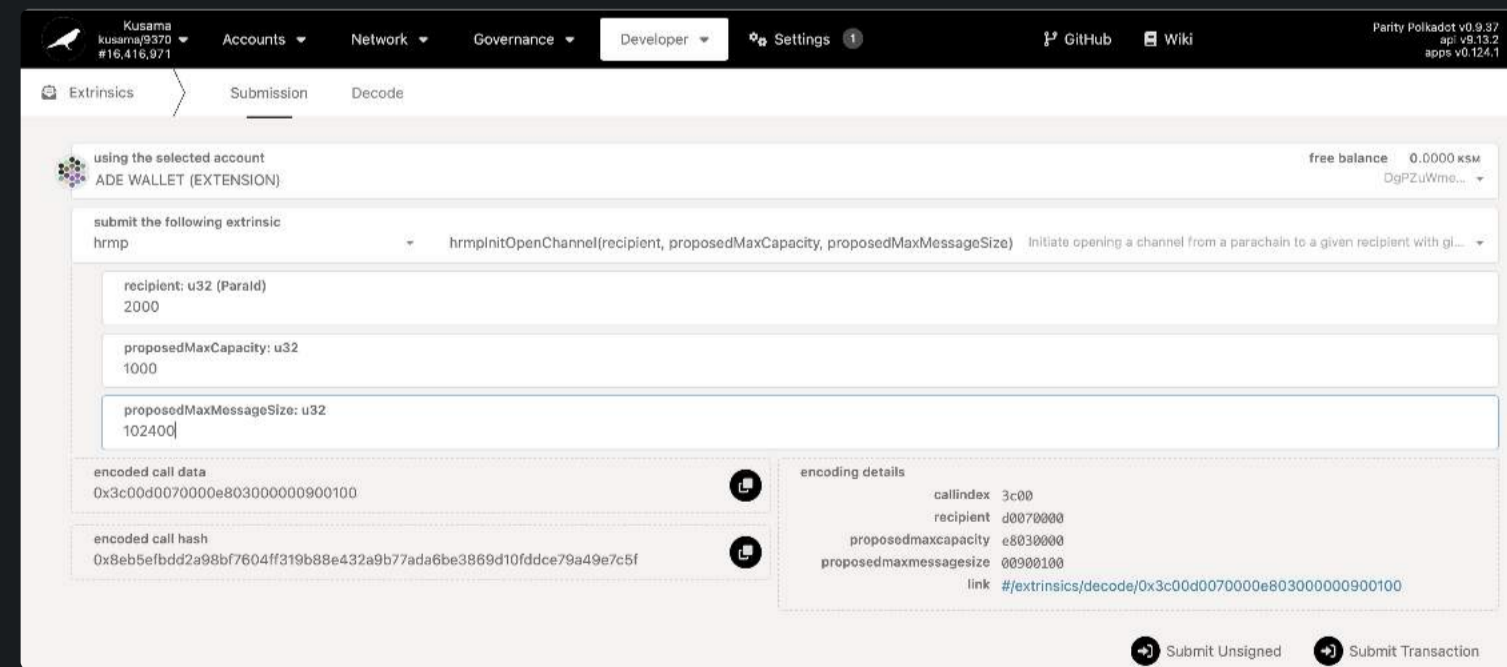
To do this, select the correct Relaychain (in this example, we use Kusama) as shown in 1 below and select Extrinsic from the drop-down options on the Developer tab.



Next, select HRMP from the "Submit the following Extrinsic" drop-down options and chose `hrmpInitOpenChannel(recipient, proposedMaxCapacity, proposedMaxMessageSize)` in the part labeled 1 below:

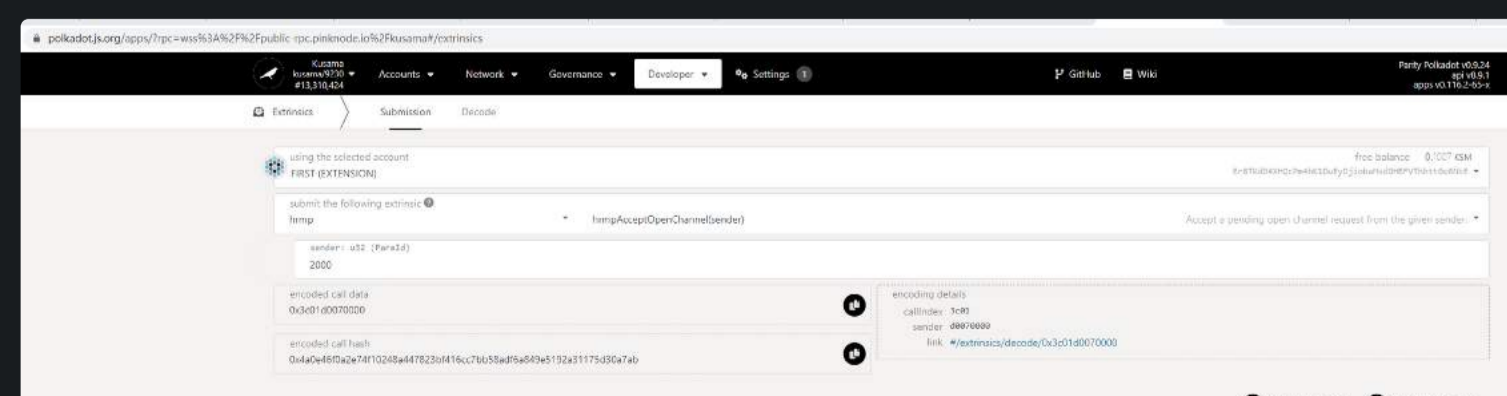


Next, enter the Parachain ID of your partner parachain in the `recipient: u32 (ParaId)` dialogue box as shown below and enter a safe estimate of your channel usage limit in the `proposedMaxCapacity: u32`, and `proposedMaxMessageSize: u32` dialogue boxes as shown below and hit the "Submit Transaction" button.



Open HRMP Channel Request Sample

Next, chose `hrmpAcceptOpenChannel(sender)` as shown below to accept the pending open channel request from the sender, enter the sender's parachain ID, then hit the "Submit Transaction" button.

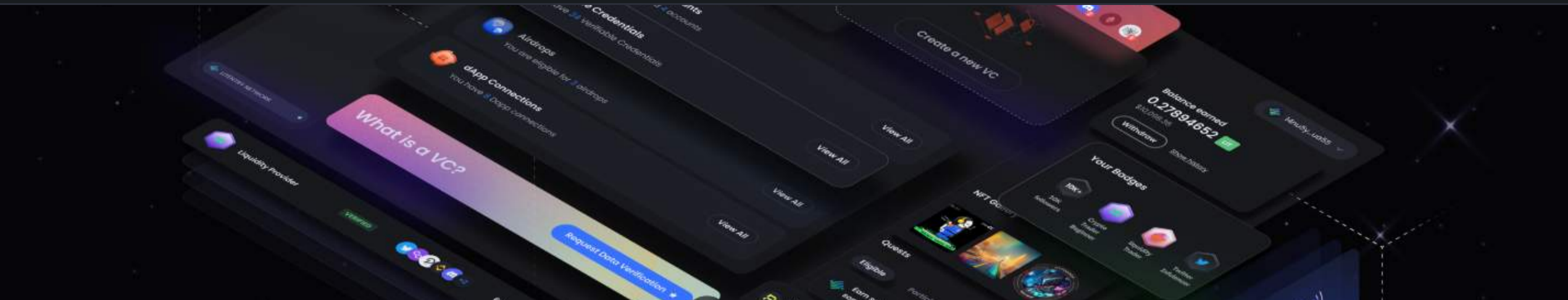


Accept HRMP Channel Request Sample

The transactions above should be sent through XCM to Relay by your sovereign account. Please refer to polkadot official documents <https://wiki.polkadot.network/docs/build-hrmp-channels> and <https://docs.substrate.io/reference/how-to-guides/parachains/add-hrmp-channels/> for more details.

## Step 3: Accepting/Waiting for Litentry Parachain's HRMP transaction

The Litentry Team will also send HRMP Channel transactions and once mutually matched, the HRMP channel will be established.



# IdentityHub

An introduction to the Litentry IdentityHub

## What is the IdentityHub (IDHub)?

The IdentityHub is the the interface to the protocol and front-end product of Litentry. It functions as a **decentralized Web3 personal data management tool**. Designed for users to aggregate and manage their personal data among blockchains and decentralized storage systems and provide data access to 3rd party dApps to maximize personal identity value without compromising privacy & anonymity.

The IdentityHub the platform where the user and a project can discover the value of identity data together. Projects can define their data requirements and attract their perfect user or audience in return for a benefit in the form of product personalisation, early access, social impact or other identity based incentives.

The IdentityHub is also an experimental playground for new social and economic innovations based on privatized identity data. It functions as a showcase of the Litentry Protocol, SDK and it's privacy preserving identity technology.

## 3 functions of the IdentityHub

- **As an interface** to the underlying Litentry Protocol, it offers features such as privacy-preserving identity linking, proving account ownership, and issuing privacy-preserving verifiable credentials based on the underlying digital data of the linked accounts.
- **As a gateway**, it offers web3 products and services several ways and methods to guide its users in leveraging granular identity details to unlock a better product experience.
- **As a platform**, it provides an environment where the web3 entrepreneur and digital identity owner meet to define the value of identity and can exchange data and benefits with one another.

## The problem the IdentityHub (IDHub) solves

IdentityHub is a solution to the violation of user data rights and privacy breaches. It allows users to control who can access their data without revealing their personal identifiable information or root accounts. When a dApp uses IDHub to access user data, it is following the correct process for protecting user data rights. On the other hand, if users create identities on IDHub, they can be assured of their personal data rights. The IDHub protects personal data rights and restores an anonymous yet richly identifiable pseudonymous web browsing experience.

← [Establishing XCM communication wi...](#) Previous

Next [Product Features](#) →



# Product Features

This page provides an overview of the Identity Hub product features.

## The Identity Hub Client

The ID-Hub client is the interface for the user to generate, manage and interact with their aggregated identities. The client will also sync with the Litentry blockchain to submit the latest state of the user's ID graph and issue verifiable credentials.

When a 3rd party dApp wants to access the user's identity data, it must make a request to the ID-Hub client and get the user's authorization before it gets the data. The Litentry blockchain will only allow returning identity data to the identity owner, it is up to the user to decide whether to give the data to a 3rd party.

The ID-Hub client and interface allows the user to interact with the following product features.

### Identity Dashboard

The profile Dashboard offers the Identity Owner an overview of their current decentralised identity. The information displayed on the dashboard can range from identity insights, trends relevant to their identities and new opportunities to leverage and use their decentralised identity across the web3 ecosystem. As our feature roadmap evolves over time so will our identity dashboard.

### Identity Graph

An identity graph records the sensitive relationships between the different accounts of a user. It is used for mapping the user's aggregated identity through any of the associated crypto addresses. IDHub aims to provide the tool to generate a trustless identity graph that can be used for generating verifiable identity data for Web3 products and services. Read more about how an identity graph is safely stored inside the [TEE](#) or how we [secure privacy](#).

### Verifiable Credentials

By analysis of the on-chain history, IDHub is able to add credentials to the aggregated identity of the user. e.g. *The credential 'long term holder' is added or created when on-chain data proofs that the identity owner has held a digital asset over an extensive period of time.* These credentials can unlock custom product experiences, exclusive access or other benefits. Credentials allows the Identity Hub to offer a tailored experience to the identity owner and show dApp offerings that fit the user's preferences.

### Identity Scores

The IDhub enables scores that reflect your web3 experience, crypto engagement, and trustworthiness as a human. A score is an identity analysis of a user's web3 & web2 behaviour. Scores provide a more granular assessment, filtering or segmentation of the user, but are also less exclusive. Not everyone has the same experience but could still get access due to their effort in different fields. Scores are more nuanced and complex than credentials. IDHub uses weighted verifiable credentials to calculate scores, and uses these scores to match projects and their identity benefits with their desired audience.

### Exploring Identity Benefits

In the near futures projects will be able to offer 'Identity Benefits' to user who share granular access to some of their identity data. Projects can get to know more by asking less. Instead of only having information about 1 wallet address the projects now can request all relevant information available across the aggregated identity of the user while respecting their privacy.



Front-end Products - Previous  
**IdentityHub**

Next

**Direct Invocation**



# Direct Invocation

Direct Invocation empowers clients to directly request the Enclave Sidechain while ensuring data integrity by posting the results on the Parachain.

The IdentityHub has implemented the first version of Direct Invocation. This update enables users to send requests to the TEE enclave directly and reduces waiting time when accessing the IDHub services. This client->blockchain request forwarding path change redirects service requests from IDHub now to the blockchain.

Direct Invocation has streamlined the process for features like setting up a shielding key, linking identities, and generating VCs. Users can now directly send service requests to the TEE sidechain, eliminating the need to go through the Litentry Parachain. Despite this change, the results of these actions will continue to be synchronized and logged on the Litentry Parachain, ensuring data integrity and transparency.

Changes in the request workflows:

- (Formerly) In Indirect Invocation: IDhub <-> parachain <-> TEE sidechain <-> parachain.
- (Currently) In Direct Invocation: IDhub <-> TEE sidechain <-> parachain.

See below for the detailed request workflows.

## Benefits and Tradeoffs

Benefits of this change include:

- Increased speed in tasks such as setting up a shielding key, linking identities, and VC generation.
- Removal of dependency on gas fees, enabling free services, and unblocking users from obtaining LIT tokens on a substrate wallet.
- Maintaining the same level of security: Identity verification, user shielding key synchronization, and VC generation remain secure within the TEE sidechain, while VC issuance transparency remains intact through the Litentry Parachain.

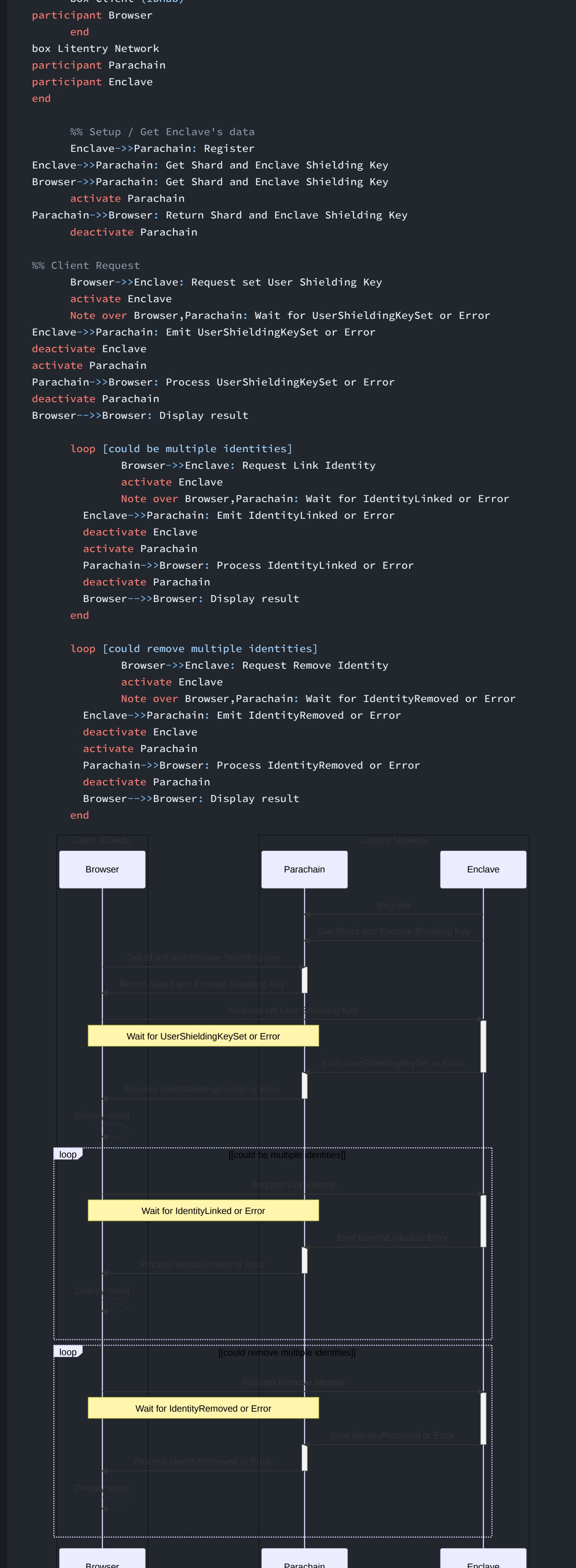
Are there any trade-offs?

- Temporary removal of LIT charges mechanism from VC generation.
- Certain transaction logs may become invisible.

## Request Workflows

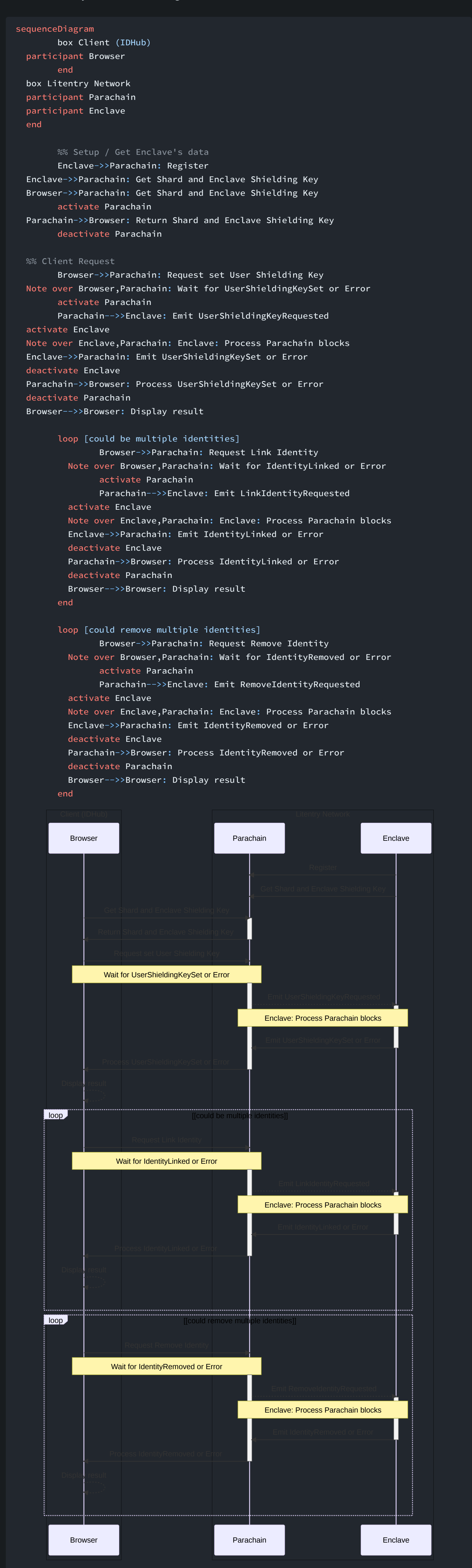
### Direct Invocation Request Workflows

Here is how Direct Invocation changed the request workflow. It enables users to directly send requests such as "set shielding key," "link identity," and "request remove identity" to the TEE Enclave.



### Indirect Invocation Request Workflow

In the previous workflow of Indirect Invocation, users were required to send a request to the Parachain initially and wait for block generation before it could reach the TEE Enclave.



## EVM Sign-In

This feature allow users to interact with the IDHub with their Ethereum Virtual Machine wallet address

To expand the assertion of digital identity securely and privately, Litentry has extended its service provision to include Ethereum users with our Ethereum Virtual Machine (EVM) sign-in feature. This novel approach leverages the **Direct Invocation** attribute of the Identity Hub that enables sending direct requests without the need to send extrinsics. With this, the IDHub can accept queries from EVM-based addresses, and users who do not own a parachain/substrate account can create IDGraphs and request VCs with their EVM wallet addresses.



### Ethereum Virtual Machine

The Ethereum Virtual Machine (EVM) is a key component of the Ethereum blockchain. It is a runtime environment that executes smart contracts, which are self-executing contracts with the terms of the agreement directly written into code. It is often described as a sandboxed, isolated environment that ensures the execution of code is consistent across all nodes in the Ethereum network.

The EVM operates on a stack-based architecture, where instructions manipulate data on a stack. It is a Turing-complete machine, meaning it can perform any computation that can be expressed algorithmically. This allows for the execution of complex operations and the implementation of decentralized applications (dApps) on the Ethereum network. Overall, the EVM enables the creation of a wide range of blockchain-based services and functionalities.

The EVM sign-in feature on the IDHub empowers users to authenticate their identity using their Ethereum addresses. In essence, it's akin to employing your Ethereum wallet to sign into a myriad of platforms, mirroring the convenience of using Google or Facebook for diverse website logins, with the big difference being that you control your data.

### Primary Identity of IDGraph

Upon the launch of IDHub, an IDGraph is created when the shielding key for a Litentry-parachain address is set. Users can link, verify, or remove identities and request VCs by sending extrinsics from that address. Now, the same can be done with an EVM address. This means that:

- Each IDGraph has one primary identity, which can be a Litentry-parachain address or an EVM address.
- Only the primary identity can modify the IDGraph.
- The shielding key is bound to the primary identity.
- IDGraphs are not merged automatically. IDGraphs with distinct primary identities are considered different. For example, if substrateA links substrateB, and the user later logs in with substrateB, the IDGraph is empty and they need to link substrateA again if they want to include it in the VC assertion building.

**i** When users log in using an EVM address for the first time, a new account with an empty IDGraph is created. To set up this new IDGraph, users must follow steps including creating a shielding key, requesting a VC, and linking identities. Existing users wanting to log in with an EVM address cannot transfer their IDGraph from their previous substrate address.

It is important to note that the primary identity is the identity that can manage (read and write) the IDGraph. Users can designate at most one web3 address as the primary address with zero restriction on the network type (Litentry-parachain or EVM address).

The address that the user uses to log in to the IDHub for the first time becomes the default primary address and it can only be changed by sending a `set_primary` request from the old primary address.

### Implementation of EVM Sign-In

The core scopes of the EVM sign-in feature are:

**Ethereum signature verification** - With EVM implementation, users will be able to create identity graphs and request VCs on IdentityHub with their EVM public/private keys, instead of only substrate keys. Users can sign in with their EVM-compatible addresses such as Ethereum and BSC to seamlessly access the IdentityHub services. This integration allows users to link their identity and establish an IDGraph using their EVM-compatible address as the primary account.

**EVM IDGraphs** - An identity graph is a data structure that represents the relationships between different identities that belong to the same individual. It represents the relationship between a user's different accounts and can be used to map out a user's aggregated identity through his EVM address.

**Identity Linking** - User Identity can be linked to only 1 IDGraph. This is because the user **shielding key** is bound to the IDGraph but can only be managed by the primary identity. As a result of this, it is the only entity that can view or update the shielding key which is required to encrypt the user data (e.g. IDgraph, or VC payload).

### Advantages of EVM Sign-In

- **Security:** Leverage the built-in security protocols of your Ethereum wallet to interact with the IDHub.
- **Simplicity:** Bypass the hassle of juggling multiple usernames or passwords. Your Ethereum wallet address is the foundation of your web3 identity.
- **Interoperability:** A bridge between Litentry parachain and Ethereum is been maintained to facilitate token flow and usage in both networks. This allows you to engage effortlessly with an extensive array of applications within the Ethereum ecosystem (now including the IdentityHub).



# Litentry

## Securing Privacy

This page discusses how Litentry will assure confidential identity data aggregation and computation.

The litentry parachain carries the confidential identity data aggregation and generates trusted identity data. The following technologies are put in place to assure the users data privacy.

### Trusted execution environment

The storage of ID graphs and the entire identity data aggregation process will be implemented by the TEE Sidechain of the Litentry network. A [Trusted Execution Environment \(TEE\)](#) is an environment for executing code, it guarantees code and data loaded inside to be protected with respect to confidentiality and integrity. [See also TEE FAQ](#)

### Confidential storage of ID graphs

Clients will submit ID graphs to the blockchain and the blockchain will verify the ID graphs. Validated ID graphs will be stored in an encrypted on-chain TEE storage.

### Request desensitiser

The request desensitiser is executed inside the TEE, it is designed for splitting accounts into separate queries and batching parallel requests belonging to different users aside from adding random addresses when sending it to external data providers. The request desensitiser makes it impossible for data providers to guess a user's ID graph based on the data requests.

### Decentralized data aggregation

After sending data requests to data providers, the Litentry network will listen to the results from data providers and aggregate the results. The data results will only include the relevant values and is aggregated according to the request ID. The user's address does not appear in the process or is not disclosed in the credential

### Selective Disclosure

A user will have maximal control over the amount of information they want to 'disclose' to a dApp. Litentry's verifiable credentials allow the user to only disclose the minimum needed amount of information required for the specific use case.



Previous  
EVM Sign-In

Next

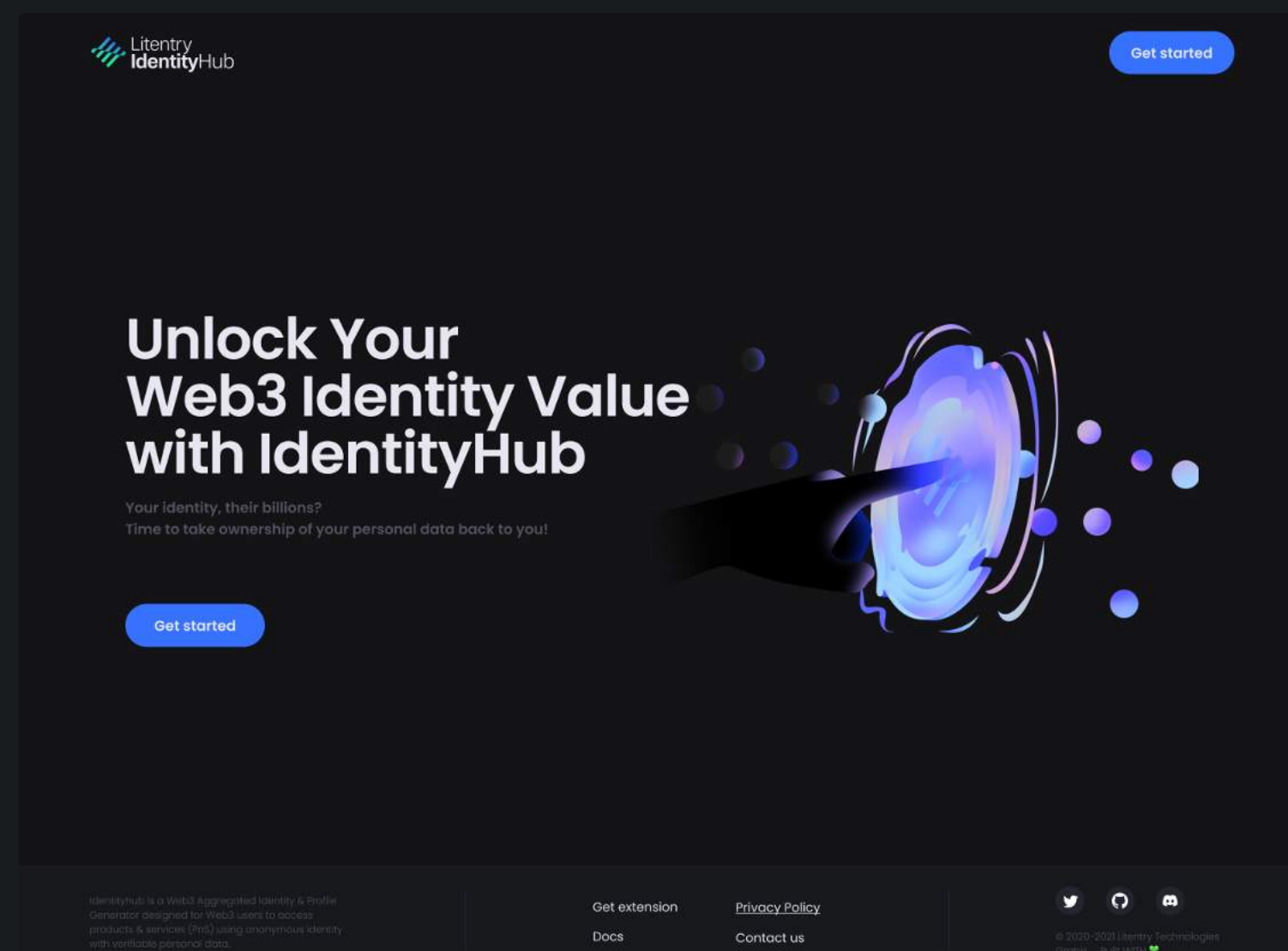
User Guide



# Getting Started with the IDHub

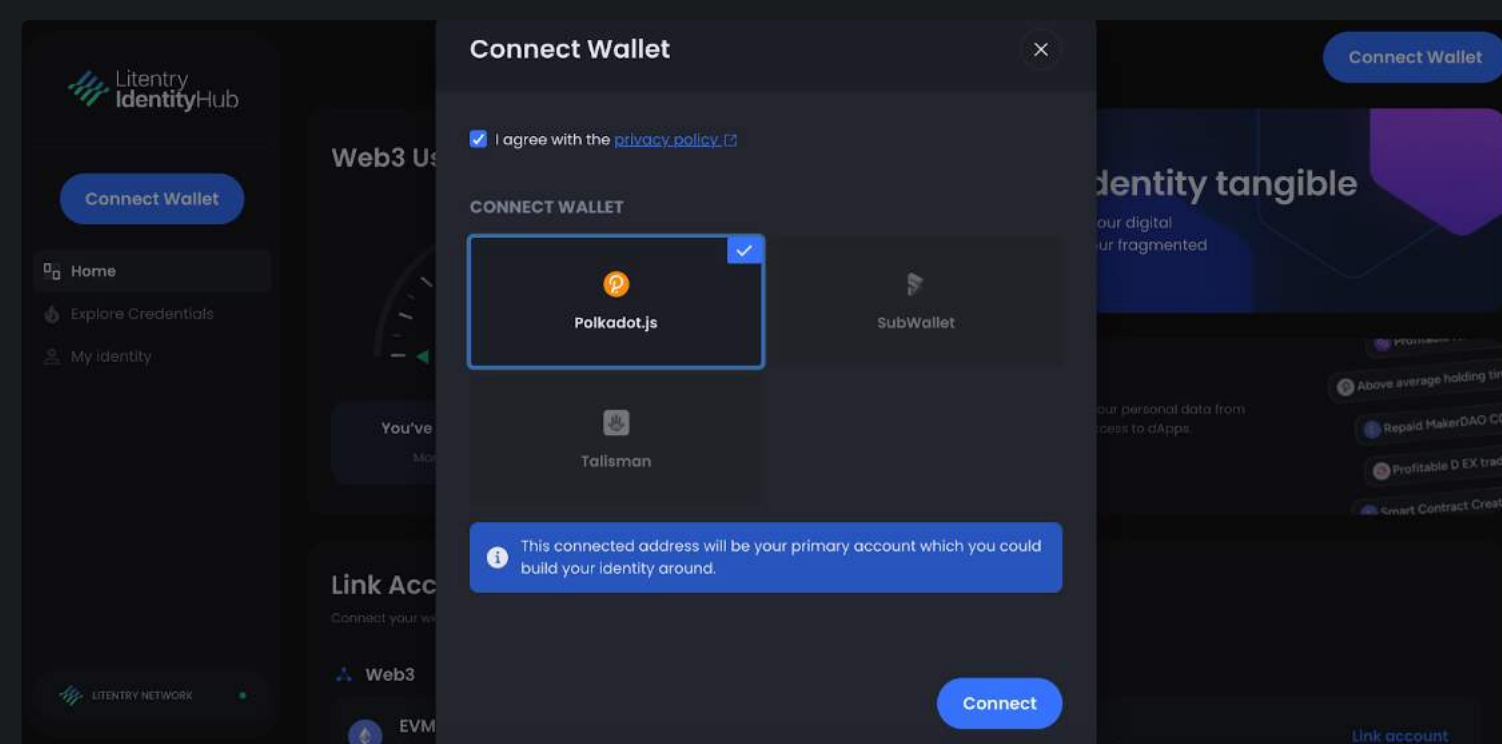
To unlock your Web3 Identity Value with IDHub, you can follow these steps and get started;

**Step 1:** Visit the IDHub via <https://idhub.litentry.io/> and click 'Get Started'.

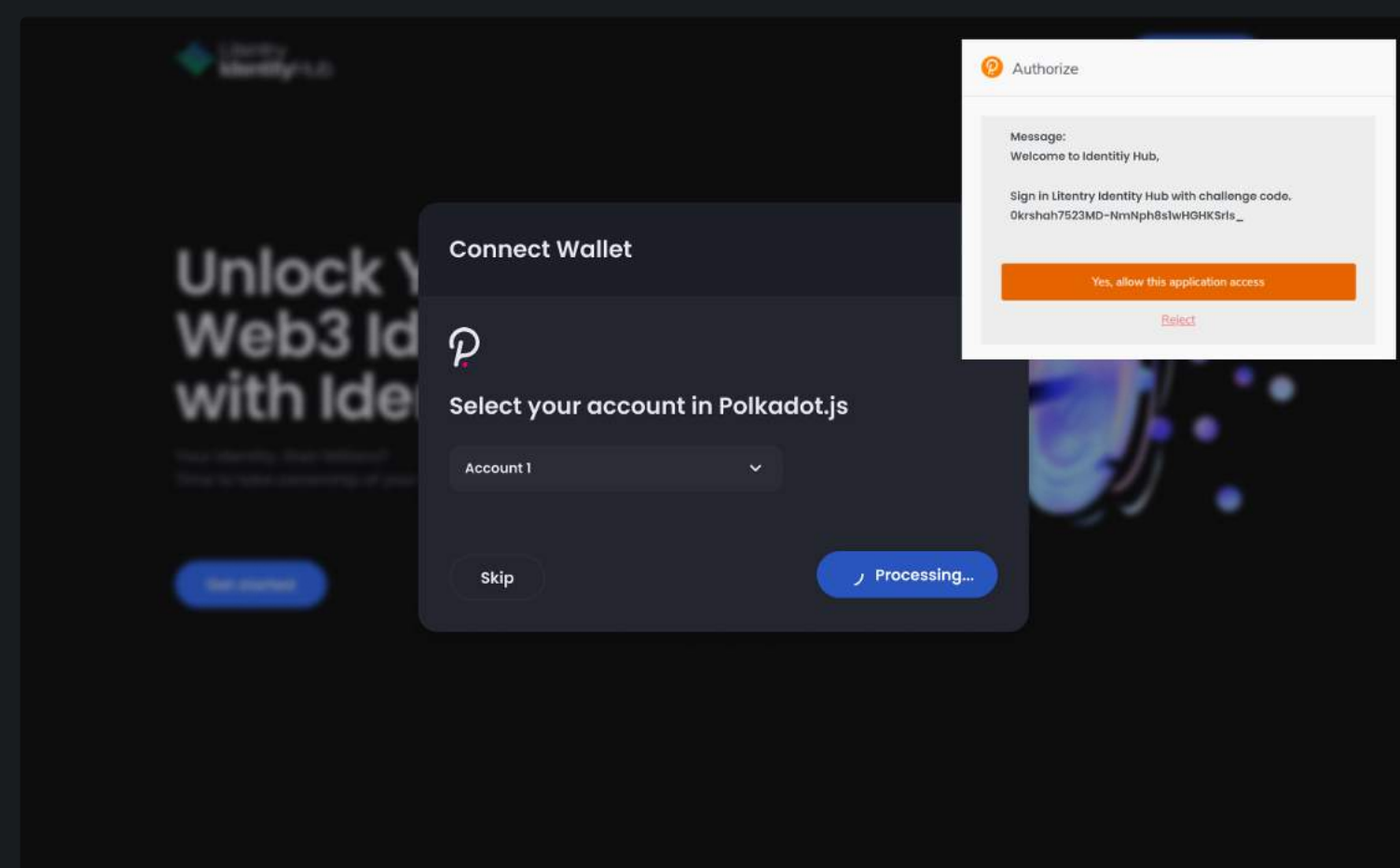


**Step 2:** Upon clicking the button, a Connect Wallet prompt will appear on your screen. Tick the privacy policy box (once you've reviewed the privacy policy) and select your desired wallet and click connect as shown below:

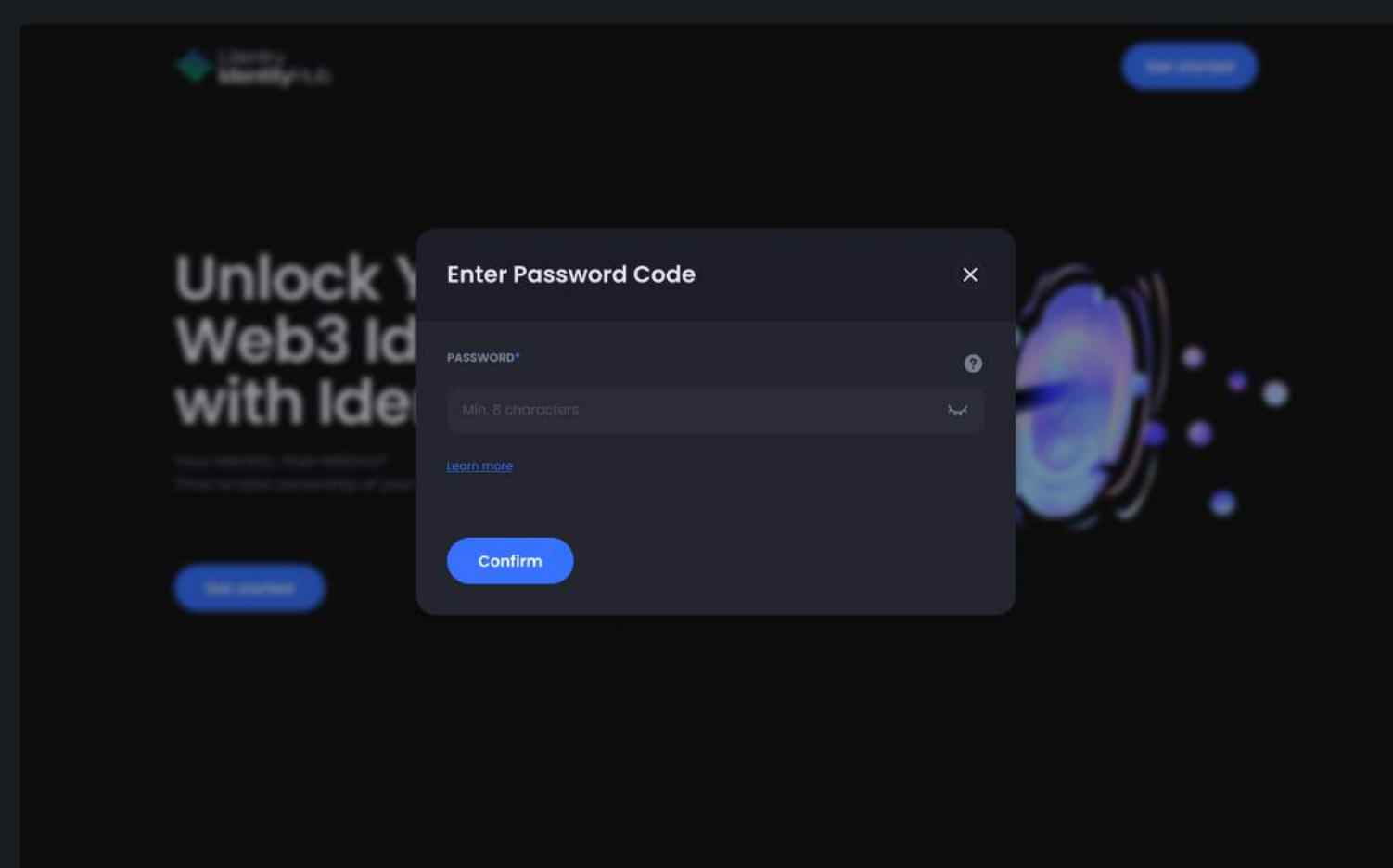
*Note: For the purpose of this tutorial we will be demonstrating using a Polkadot.js wallet.*



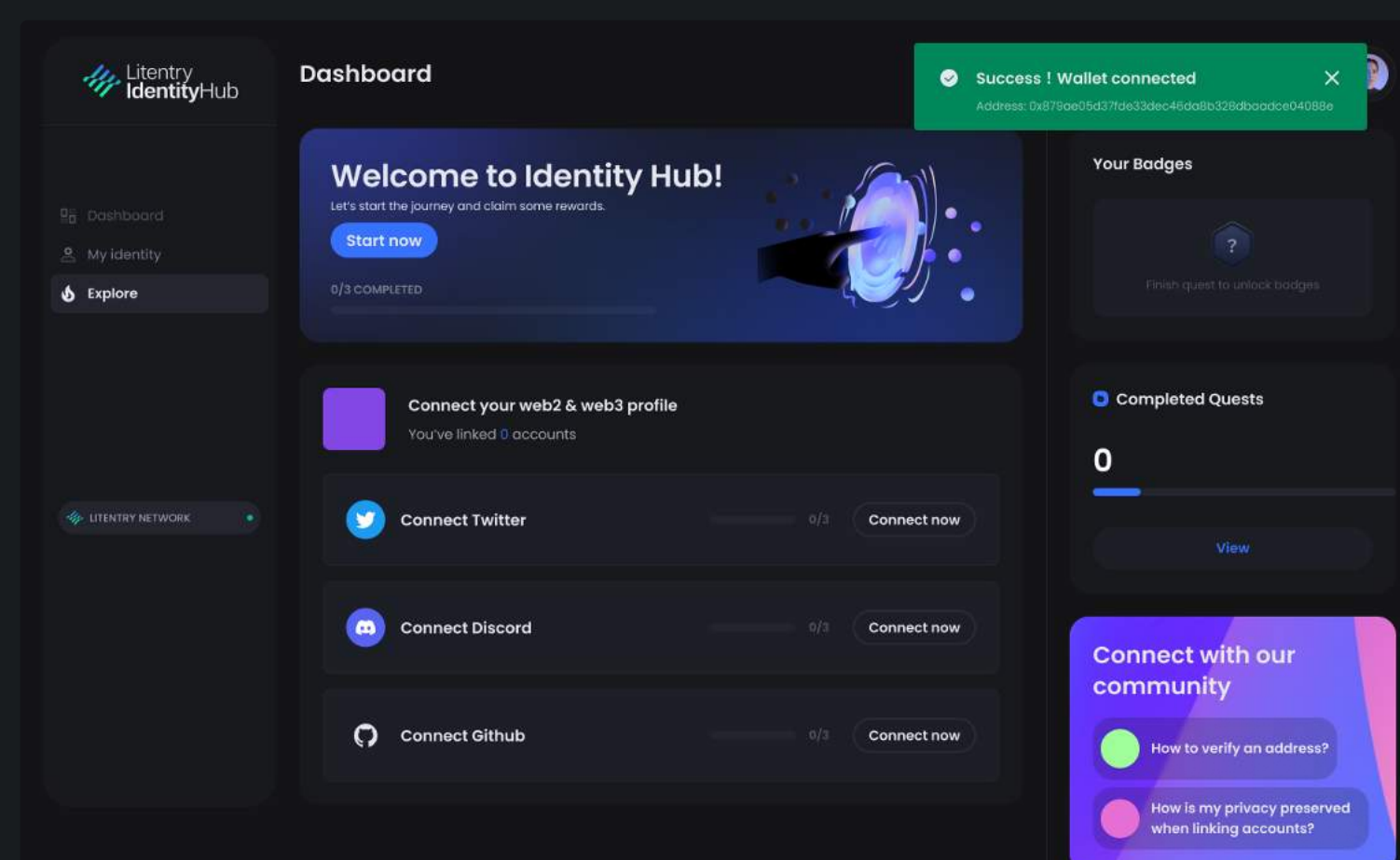
**Step 3:** Once you click the connect button, you will see a pop-up message on your screen requesting you to authorize IDHub. This includes a secure challenge code to sign in to the Identity Hub. Click the "Yes, allow this application access" button.



**Step 4:** Next, you'll be requested to enter your password.



**Step 5:** Once entered, click the confirm button and you'll be redirected to your dashboard with the pop-up message 'Success! Wallet connected' as shown below:

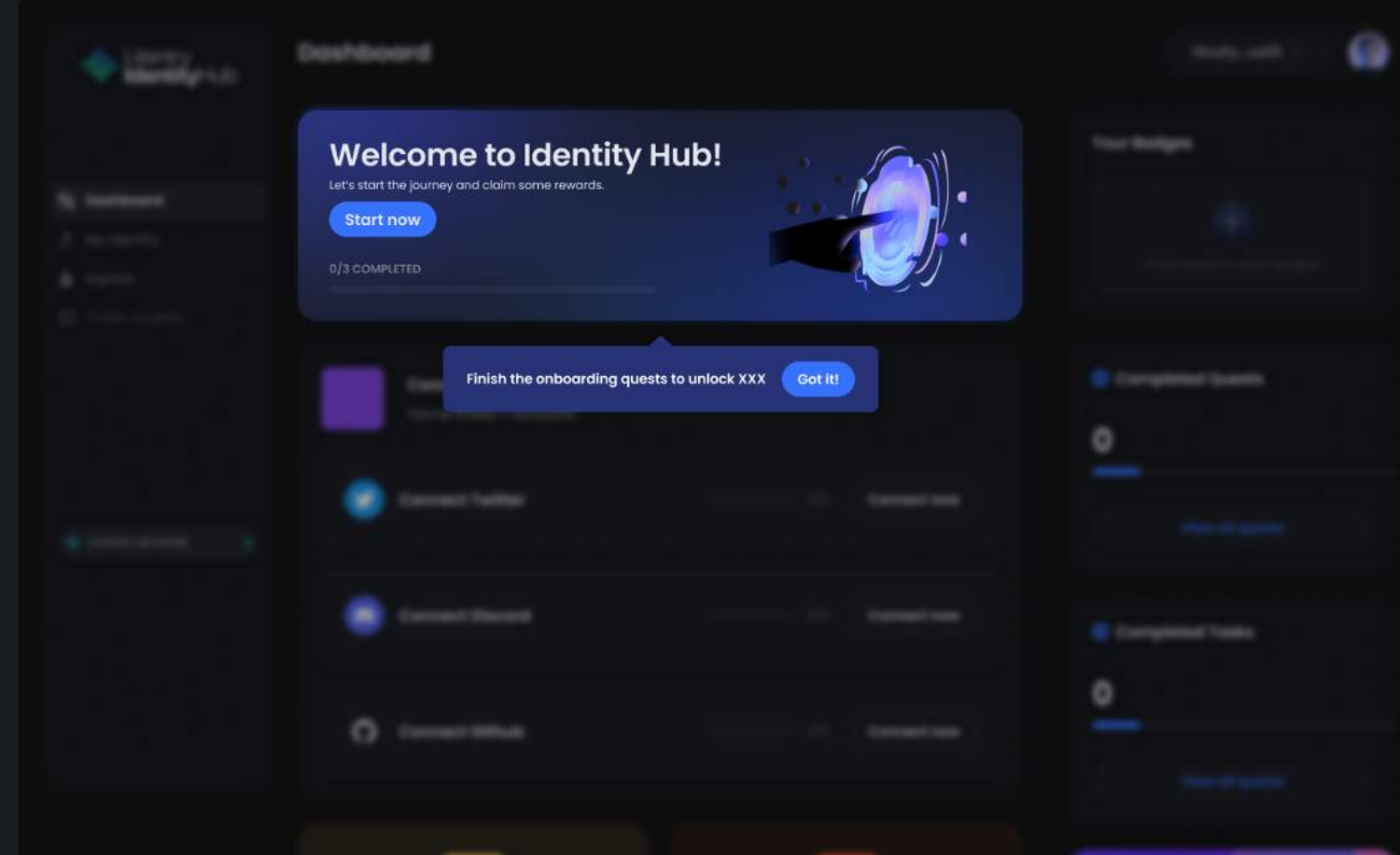


# Connecting your Web3 Account

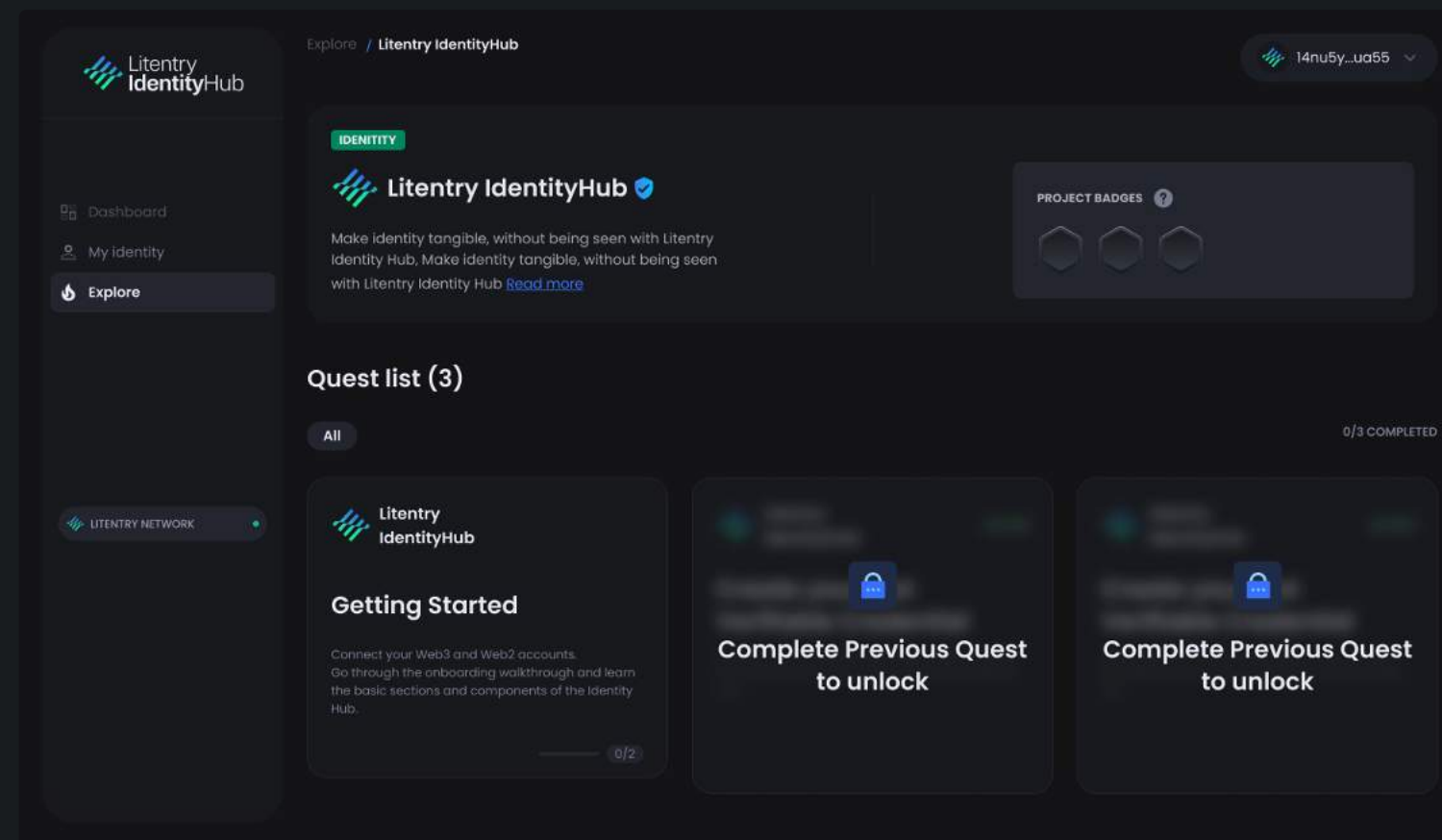
To be able to access the IdentityHub further, you need to complete some onboarding quests. These quests involve going through the onboarding tutorial and connecting your Web2 and Web3 accounts as well as others. Completing each quest, you'll be awarded project reward badges that you can share with your others and brag about with your friends.

Note: To be able to link any account, users need to create a minimum 8-character password to generate a shielding key. Please save this password locally and securely. The shielding key is used to encrypt all user on-chain data in the Litentry Network. Click [here](#) to read more about the shielding key.

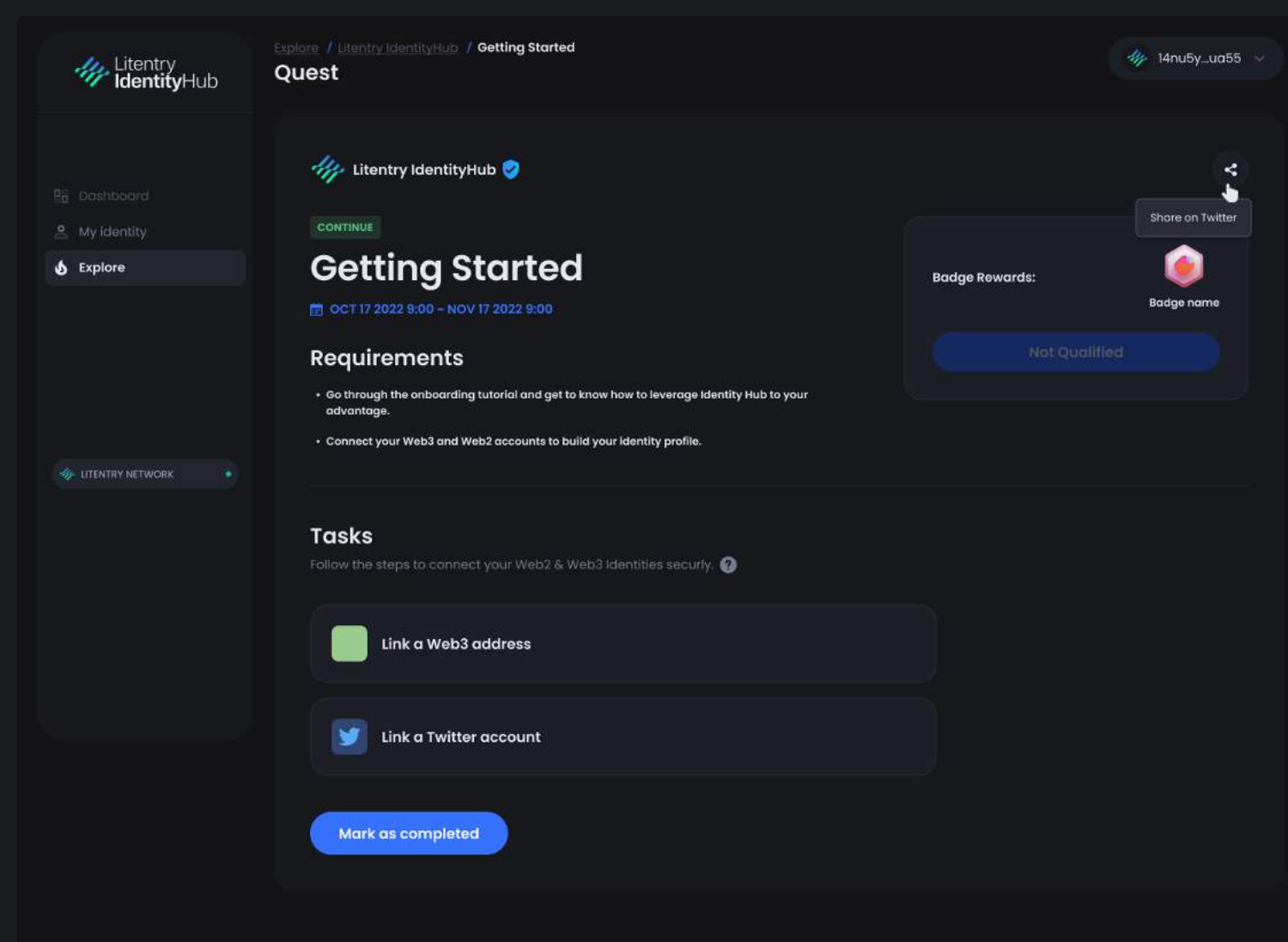
**Step 1:** To get started, click 'Start Now'.



**Step 2:** Once clicked, you'll be redirected to the interface shown below. Click the 'Getting Started' quest card.

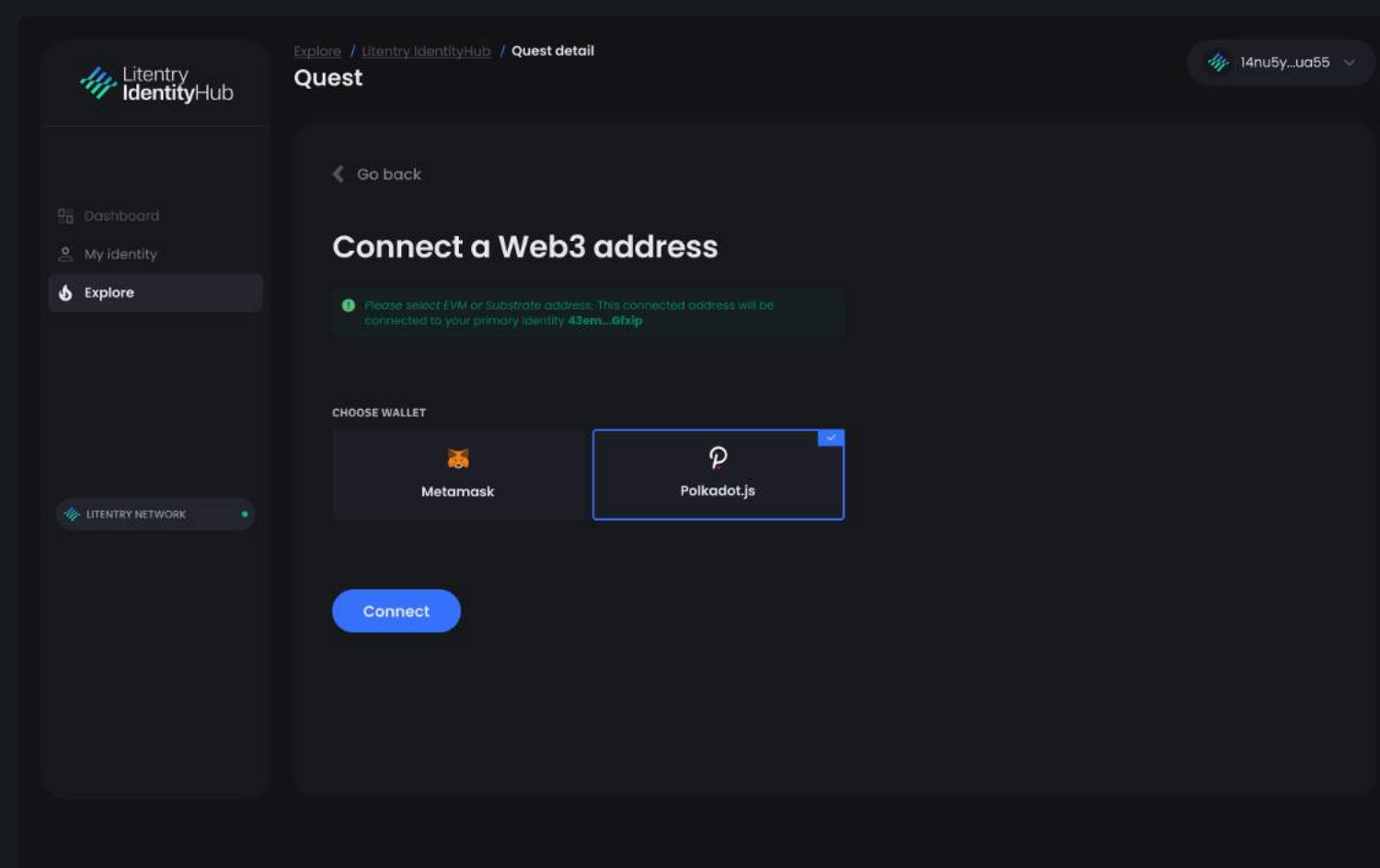


**Step 3:** The requirement for completing this quest is to go through the onboarding tutorial and connect your Web2 and Web3 accounts to build your Identity Profile.

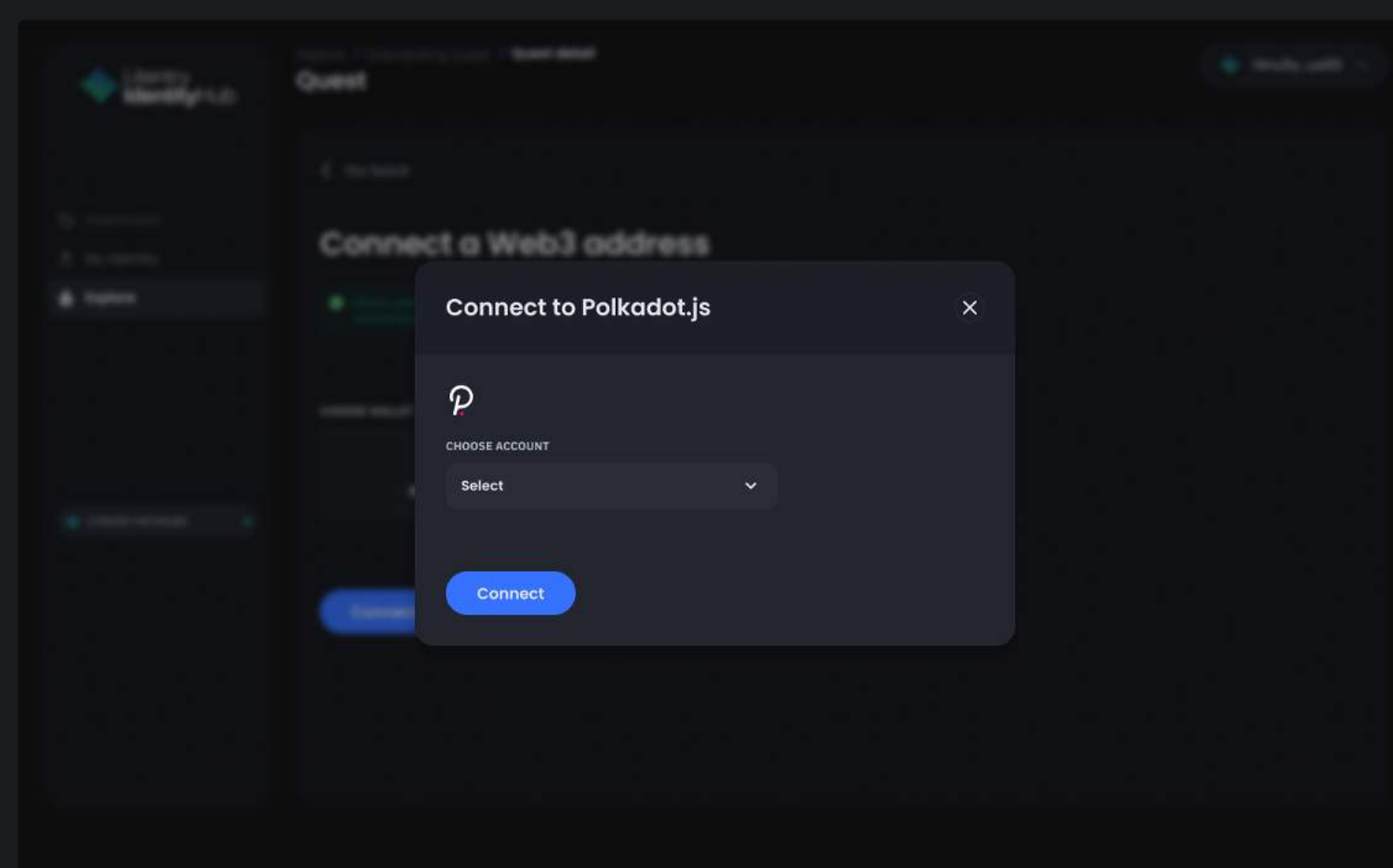


**Step 4:** Next, click 'Link a Web3 Address' to connect a Web3 address. On this page, you can connect either an EVM or Substrate Wallet address so choose to connect either Metamask or Polkadot.js wallet. Your connected Web3 address will be added to your primary account.

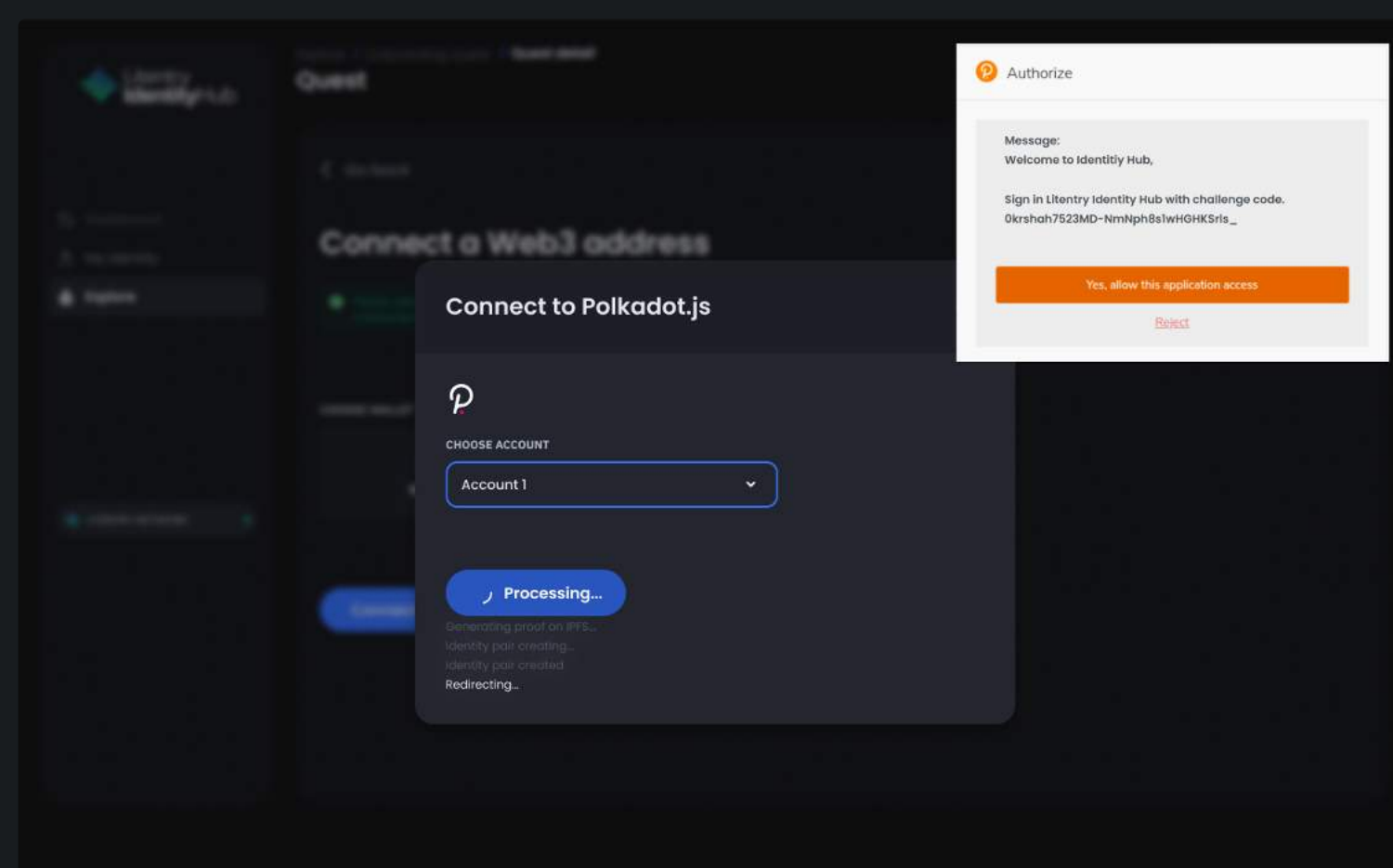
Note: For demonstration purposes, we'll use a Polkadot.js wallet.



**Step 5:** Once you select Polkadot.js, click 'Connect' and select your desired Polkadot wallet from the drop-down options.

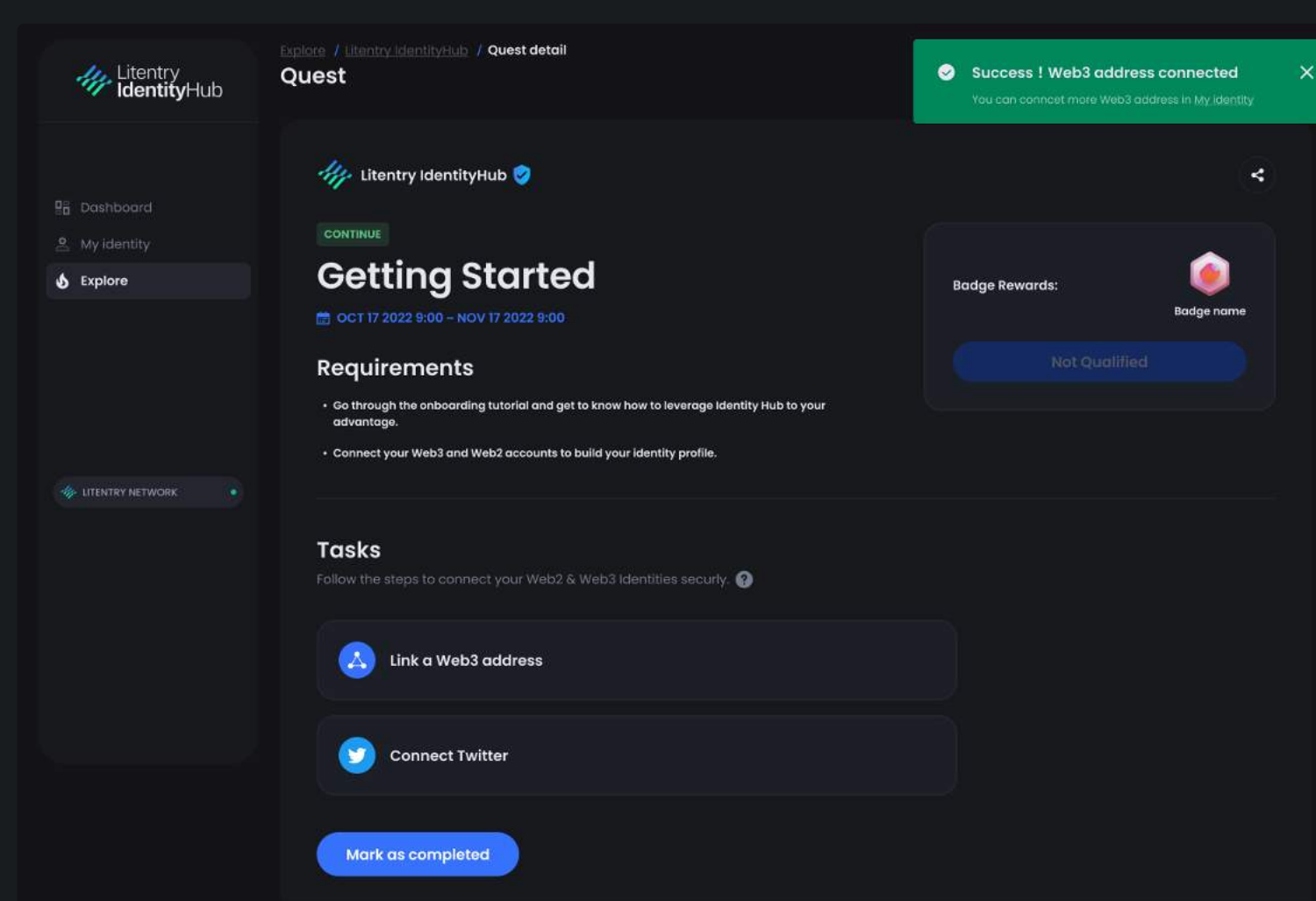


**Step 6:** A pop-up message will appear on your screen requesting you to authorize the IdentityHub. This includes a secure challenge code to sign in IdentityHub. Click 'Yes, allow this application access'.



**Step 7:** Another pop-up message will appear on your screen with the text: 'Success! Web3 address connected'. You will also see your reward badge on this page.

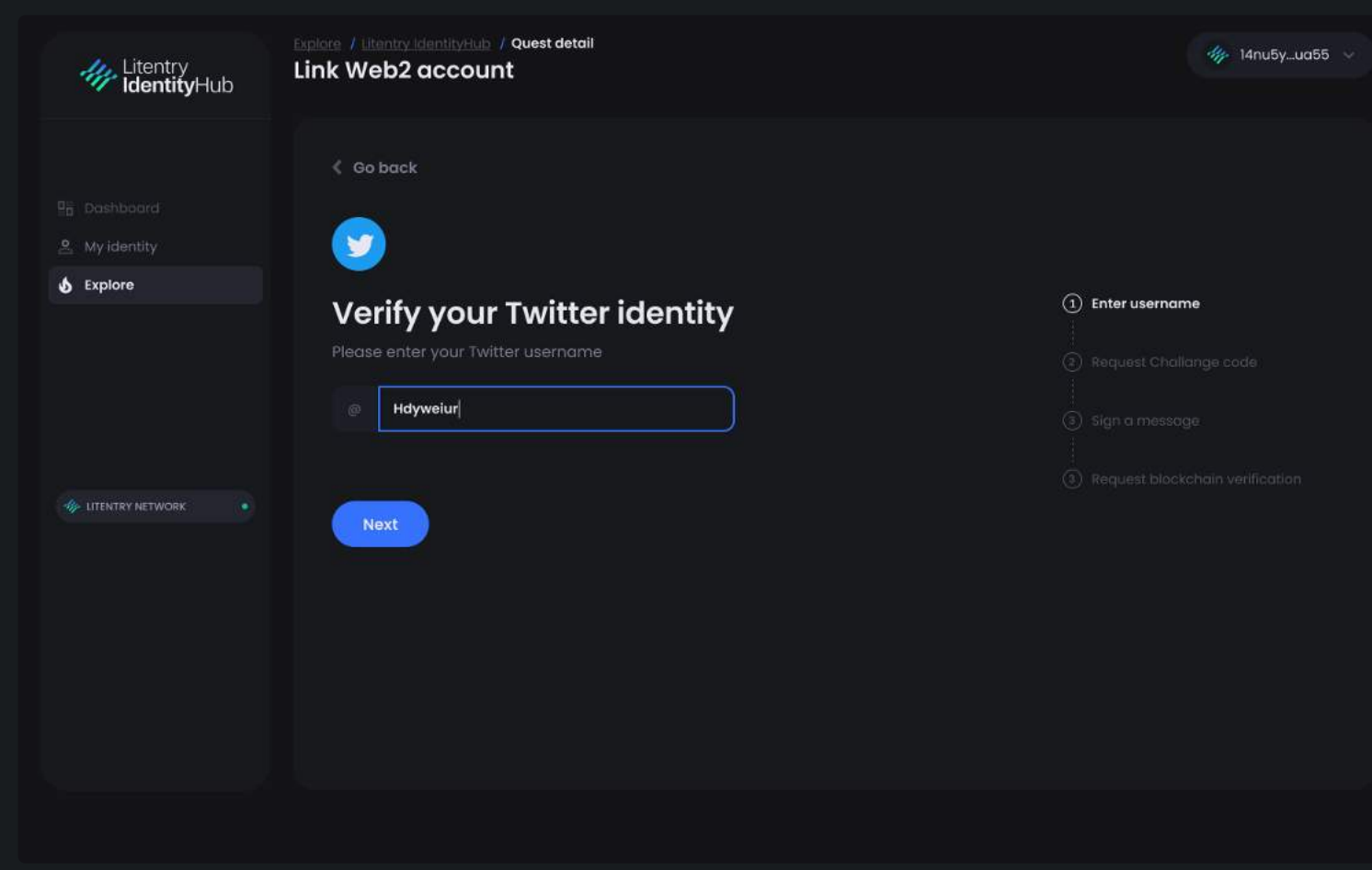
With this, you have successfully connected your Web3 address and you can connect more addresses in the 'My Identity section' of the IDHub.



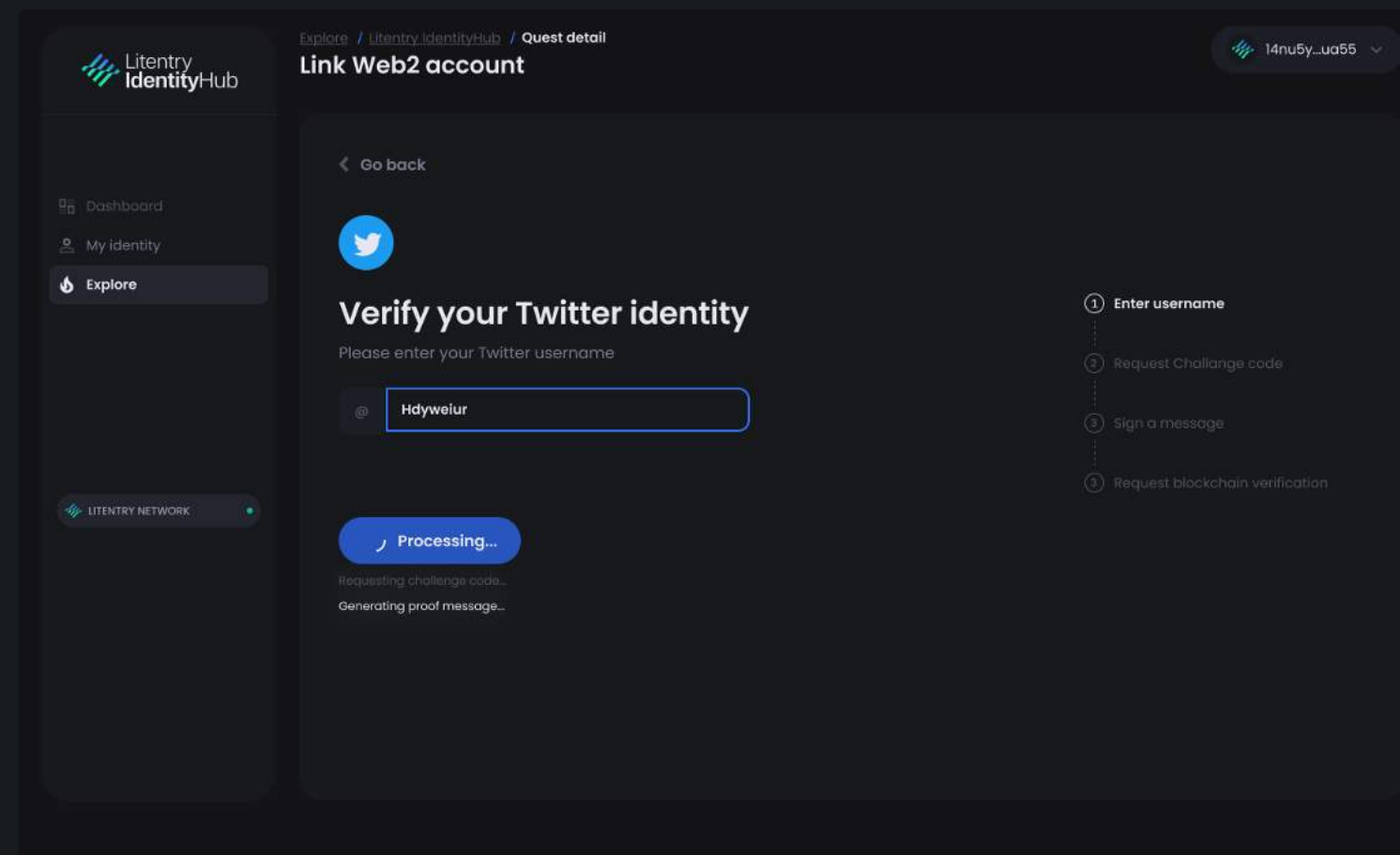
# Connecting your Web2 Account

To complete the quest, you need to link your Web2 account as well. To do that, follow the steps below:

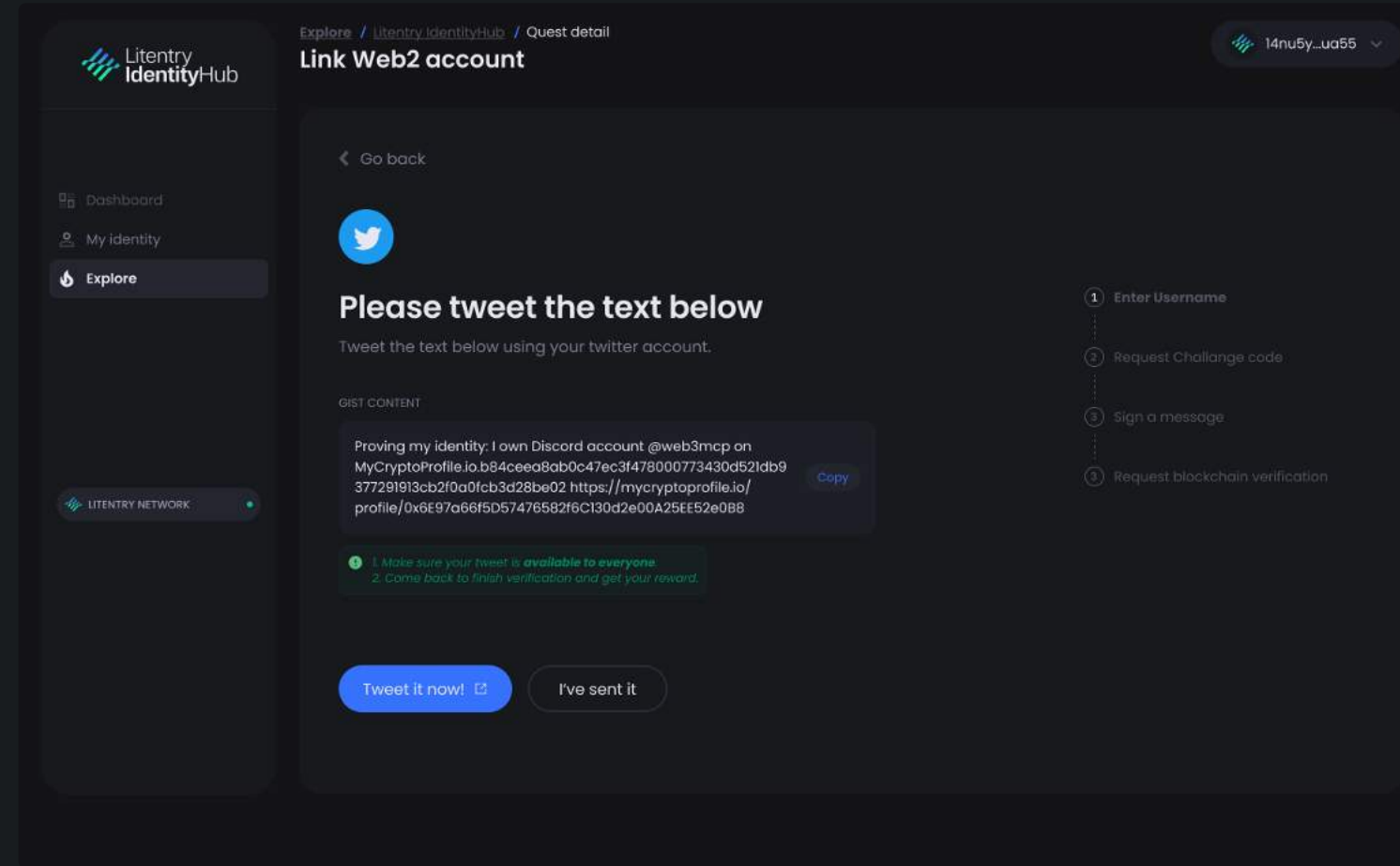
**Step 1:** Click 'Connect Twitter' and enter your Twitter username. Click 'Next' to proceed.



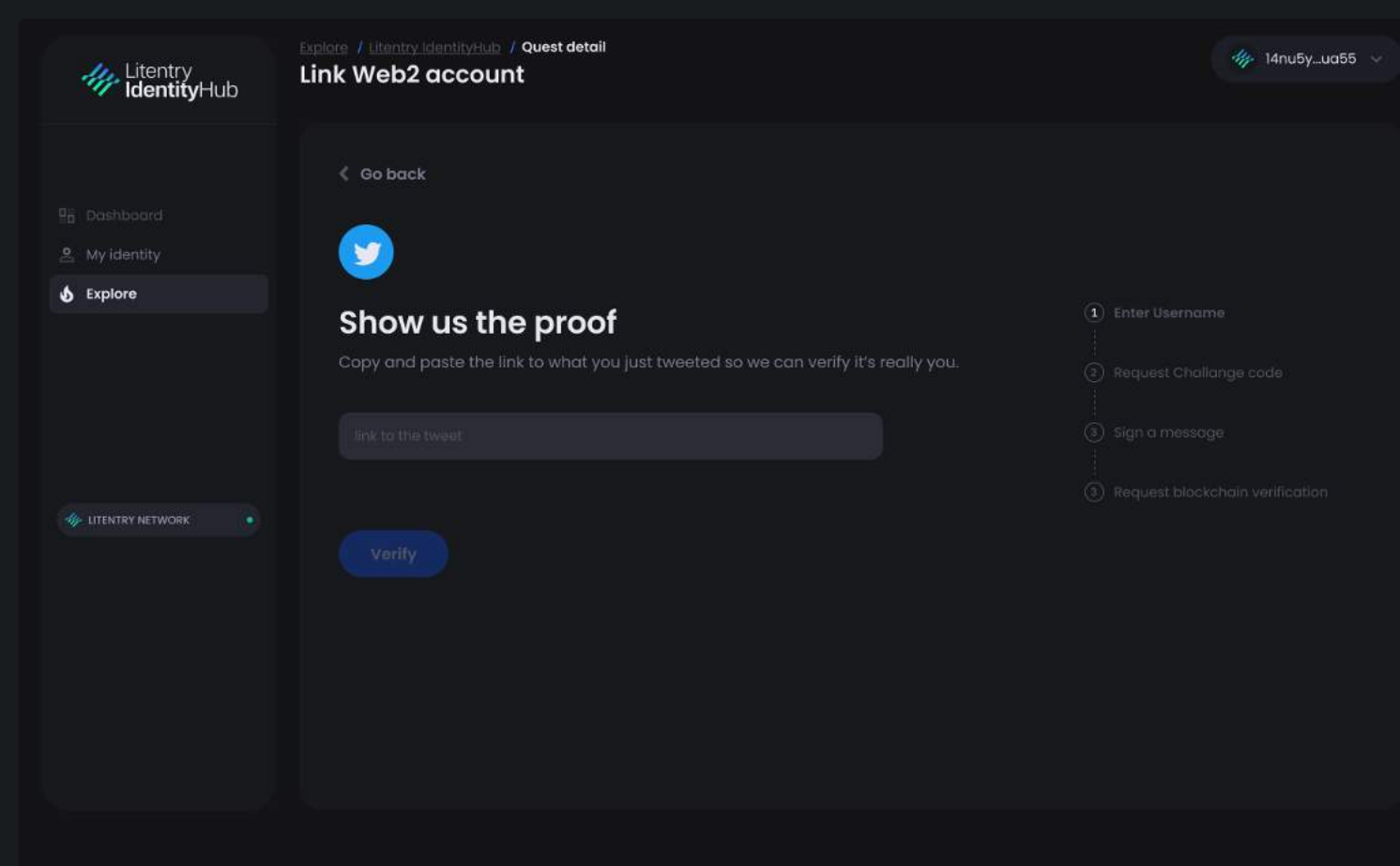
**Step 2:** The IdentityHub will automatically request a challenge code and generate a proof message for you.



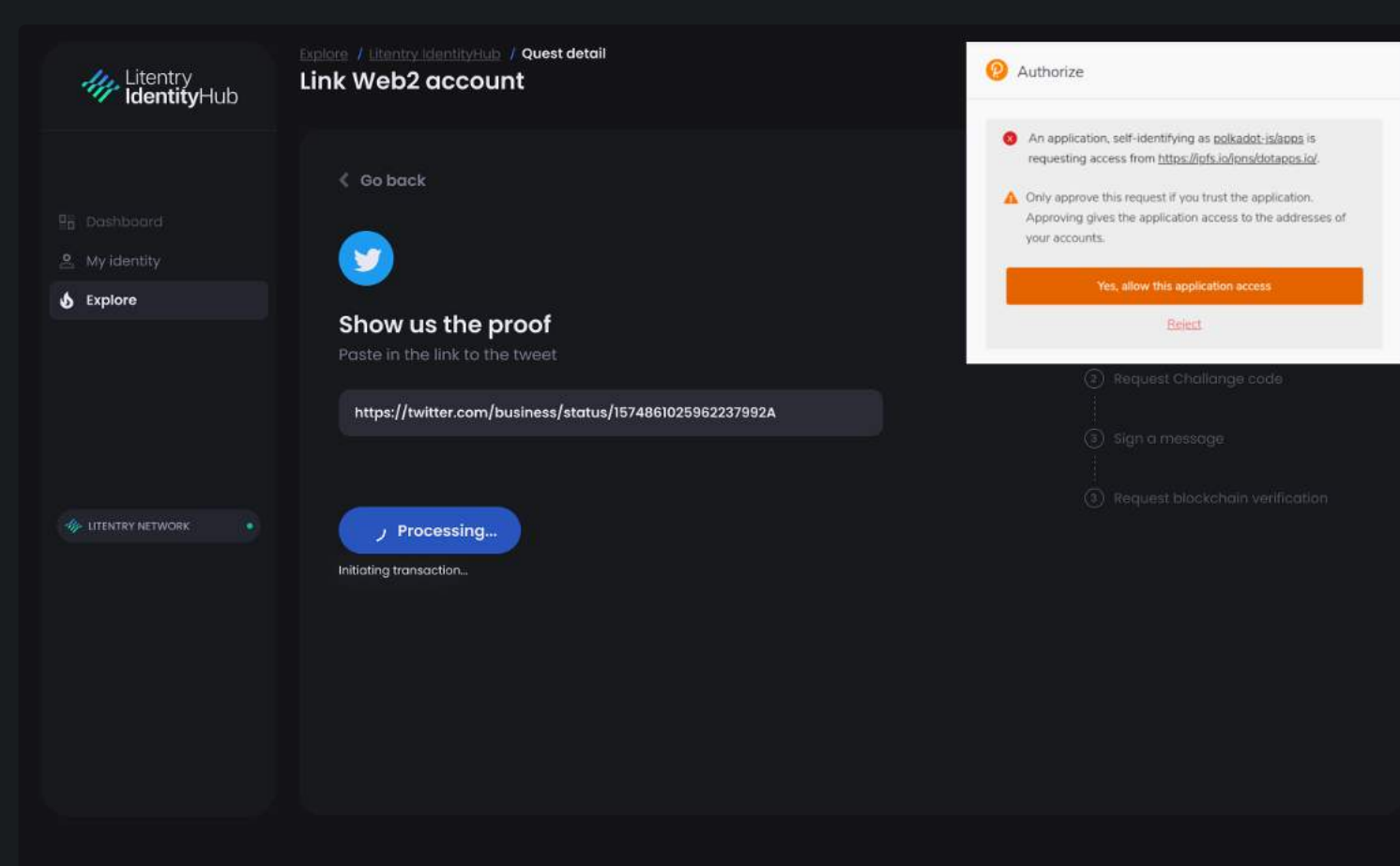
**Step 3:** You will be requested to Tweet the generated proof message on your Twitter account as shown below. Click the 'Tweet it now' to send the tweet, then return to the IDHub to proceed by clicking 'I've sent it'.



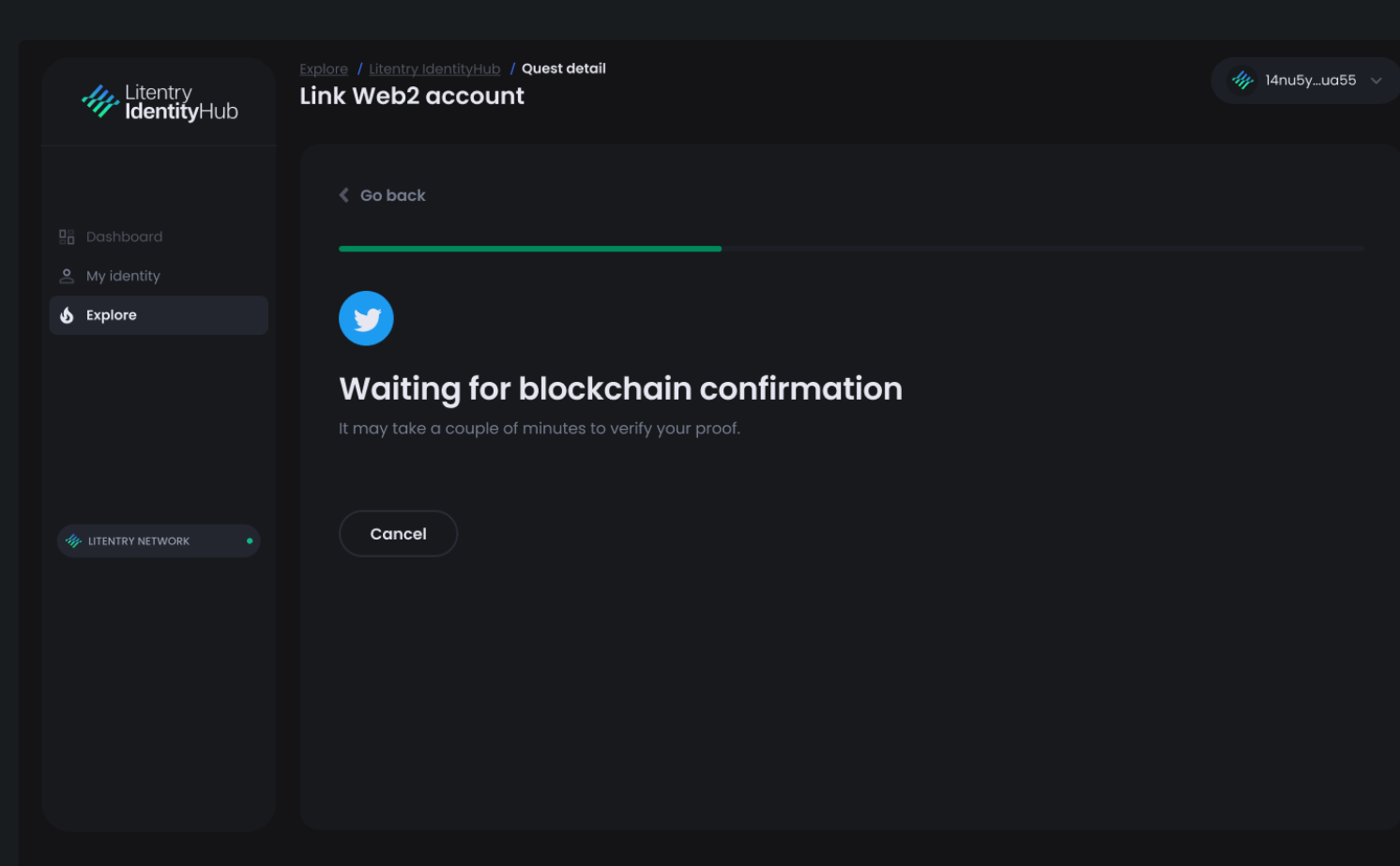
**Step 4:** Show proof of the Tweet by copying and pasting the link to it in the IDHub as shown below. Next, click 'Verify'.



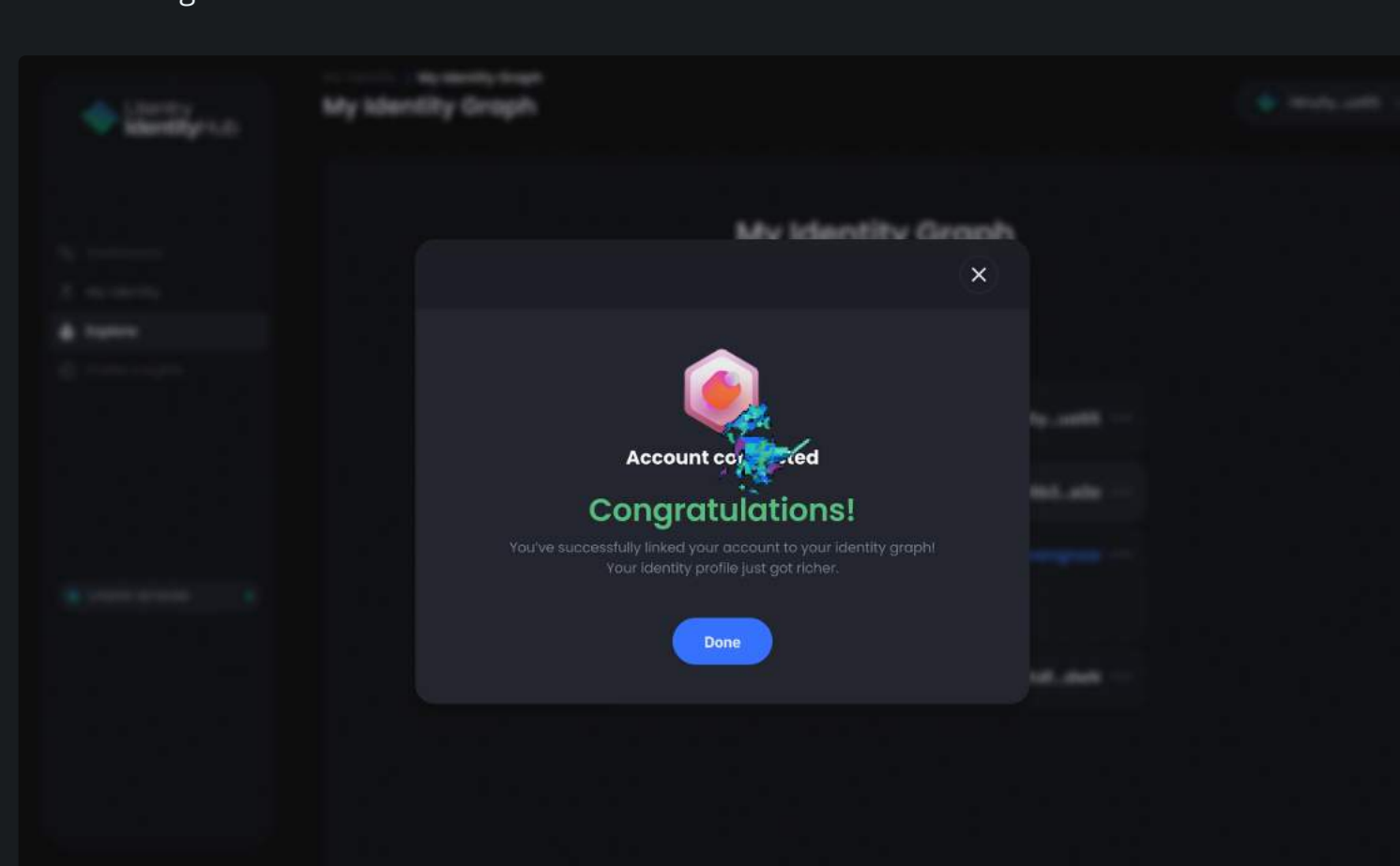
**Step 5:** You'll receive a pop-up authorization request to proceed. Click 'Yes, allow this application access'.



**Step 6:** Upon authorization, your identity will be verified on the blockchain. Wait a few moments for the blockchain to verify your proof.

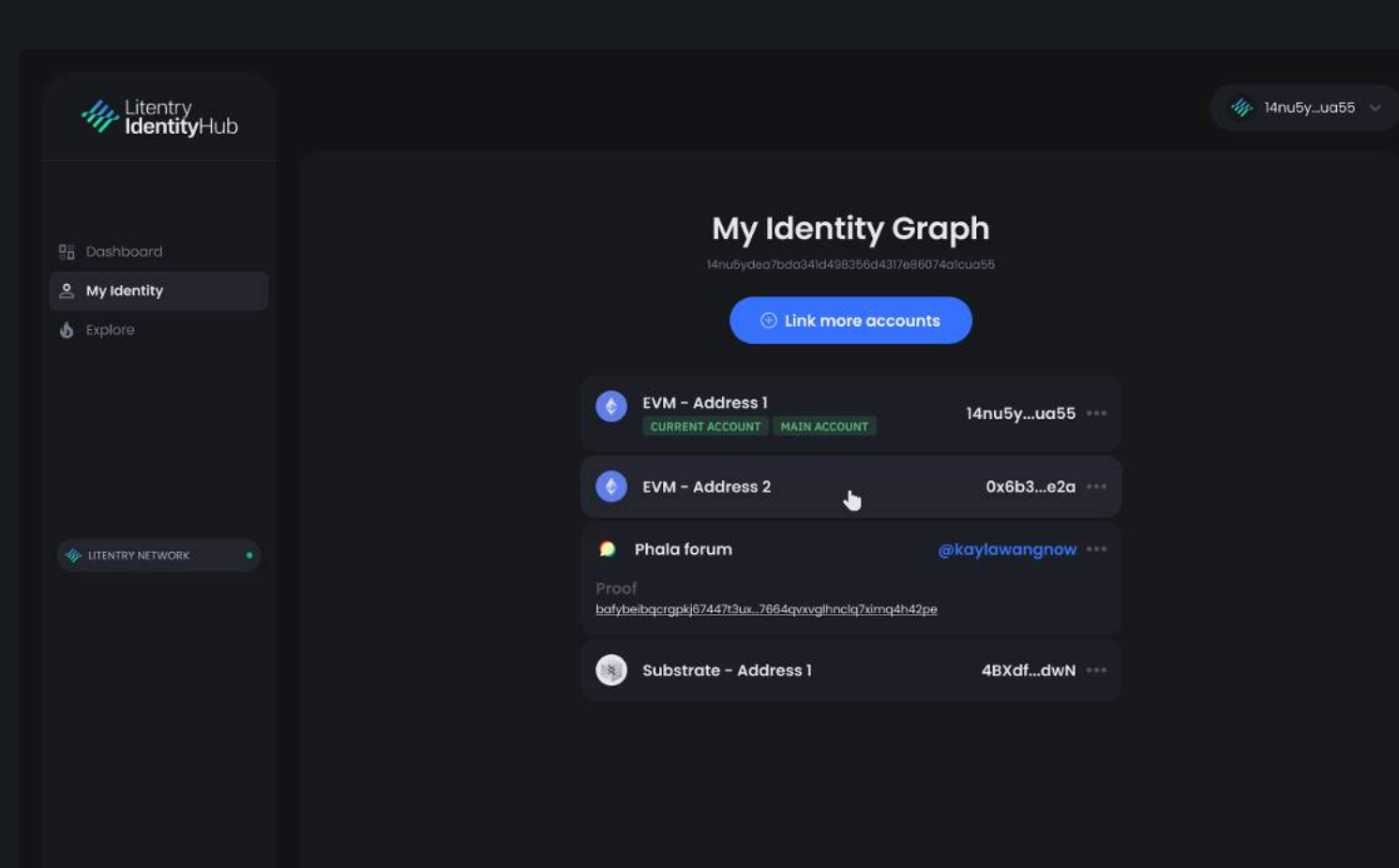


**Step 7:** Once confirmed, you'll receive a pop-up message with a congratulatory message and your reward badge.

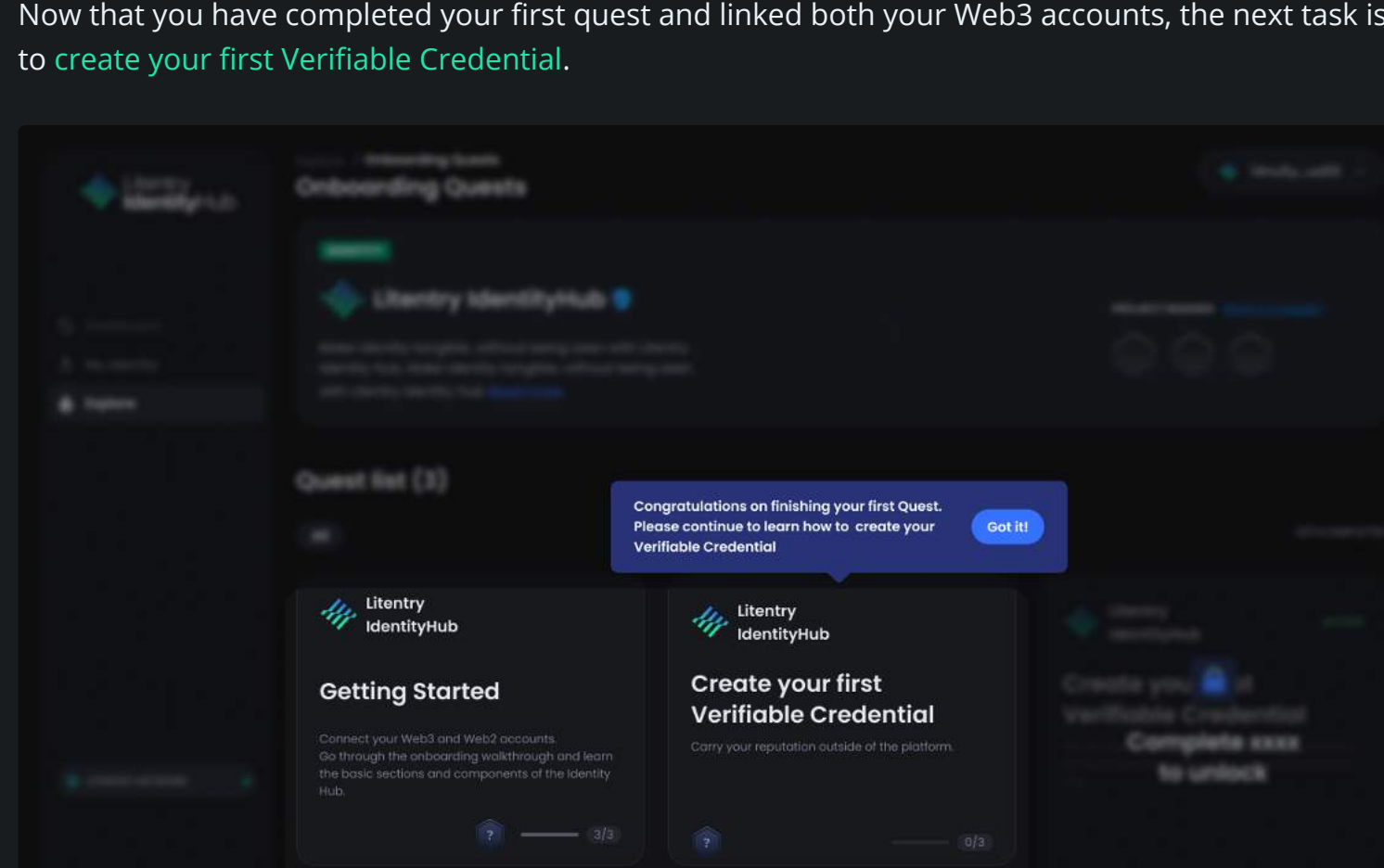


With this, you will have successfully linked your Web2 and Web3 accounts to the IDHub. You can click 'My Identity' just below the dashboard to link more accounts.

To demonstrate this, the screenshot below shows an account with two EVM addresses, a substrate address, and a Twitter account.



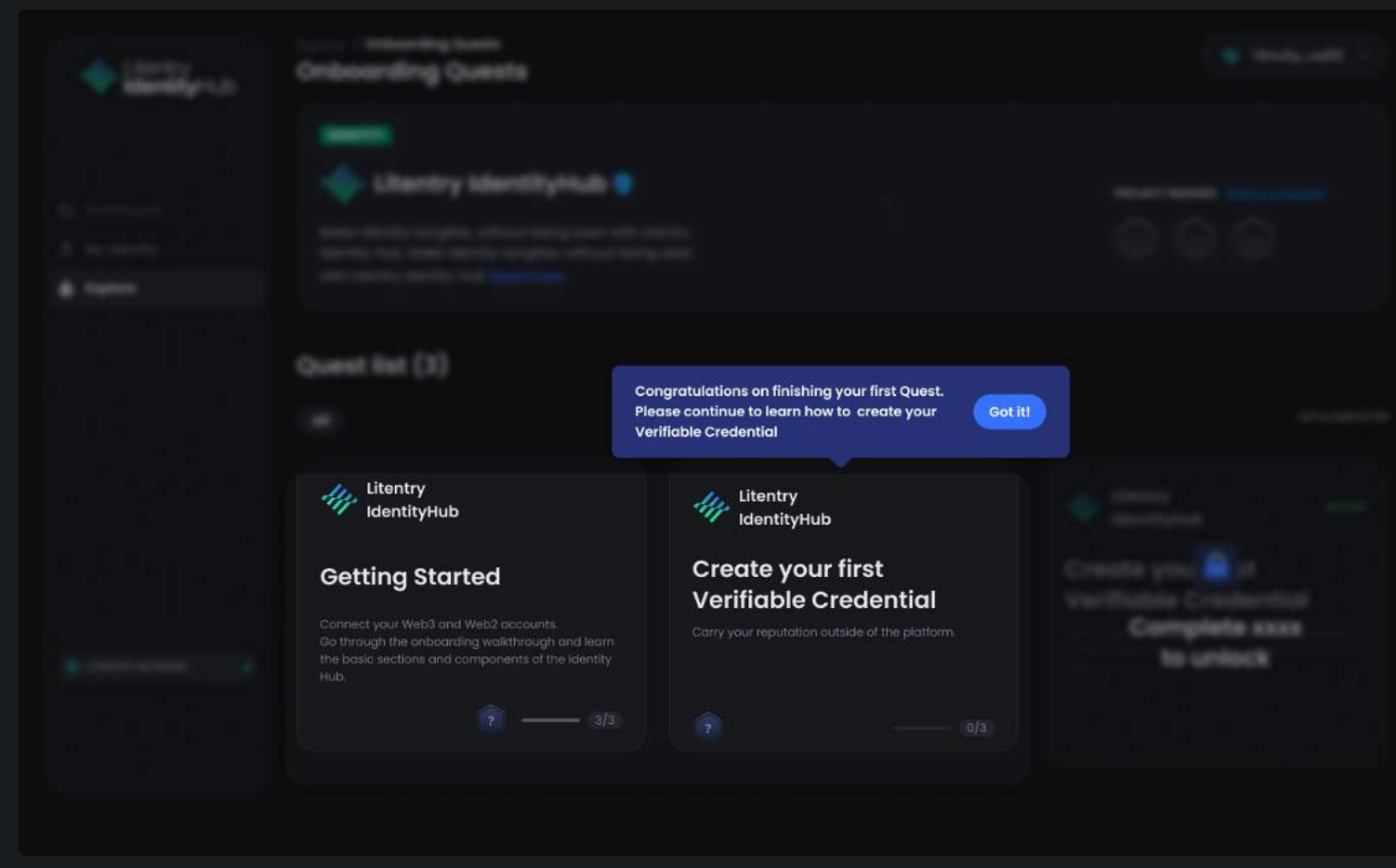
Now that you have completed your first quest and linked both your Web3 accounts, the next task is to **create your first Verifiable Credential**.



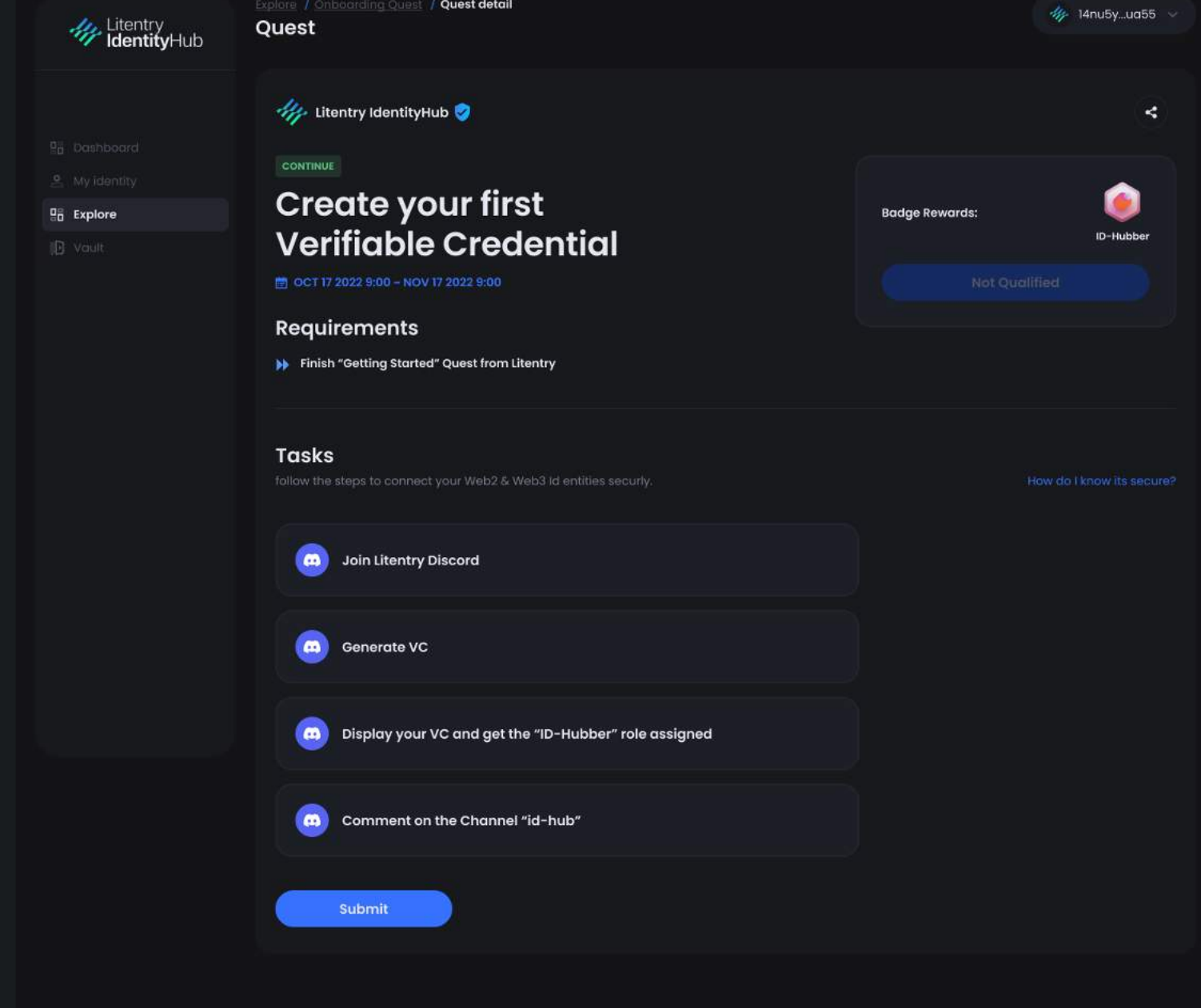
# Creating your First Verifiable Credential VC

Verifiable credential (VC) enables individuals to hold and share their digital identities with ease, privacy, and security. It is based on the idea that identity attributes can be cryptographically signed and verified through trusted parties. To generate your first VC, follow the steps below:

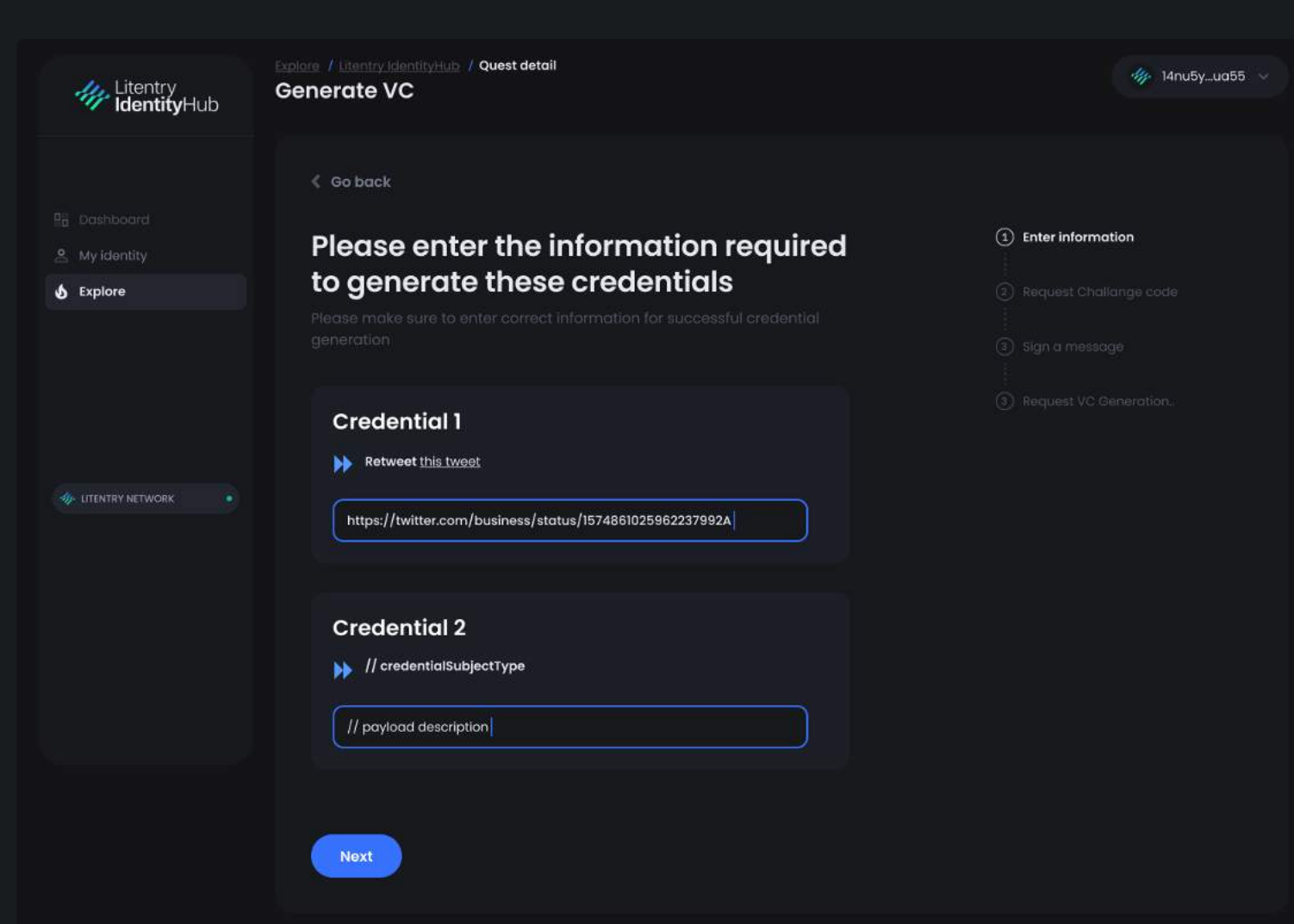
**Step 1:** Click the 'Create your first Verifiable Credential Quest' card and get started.



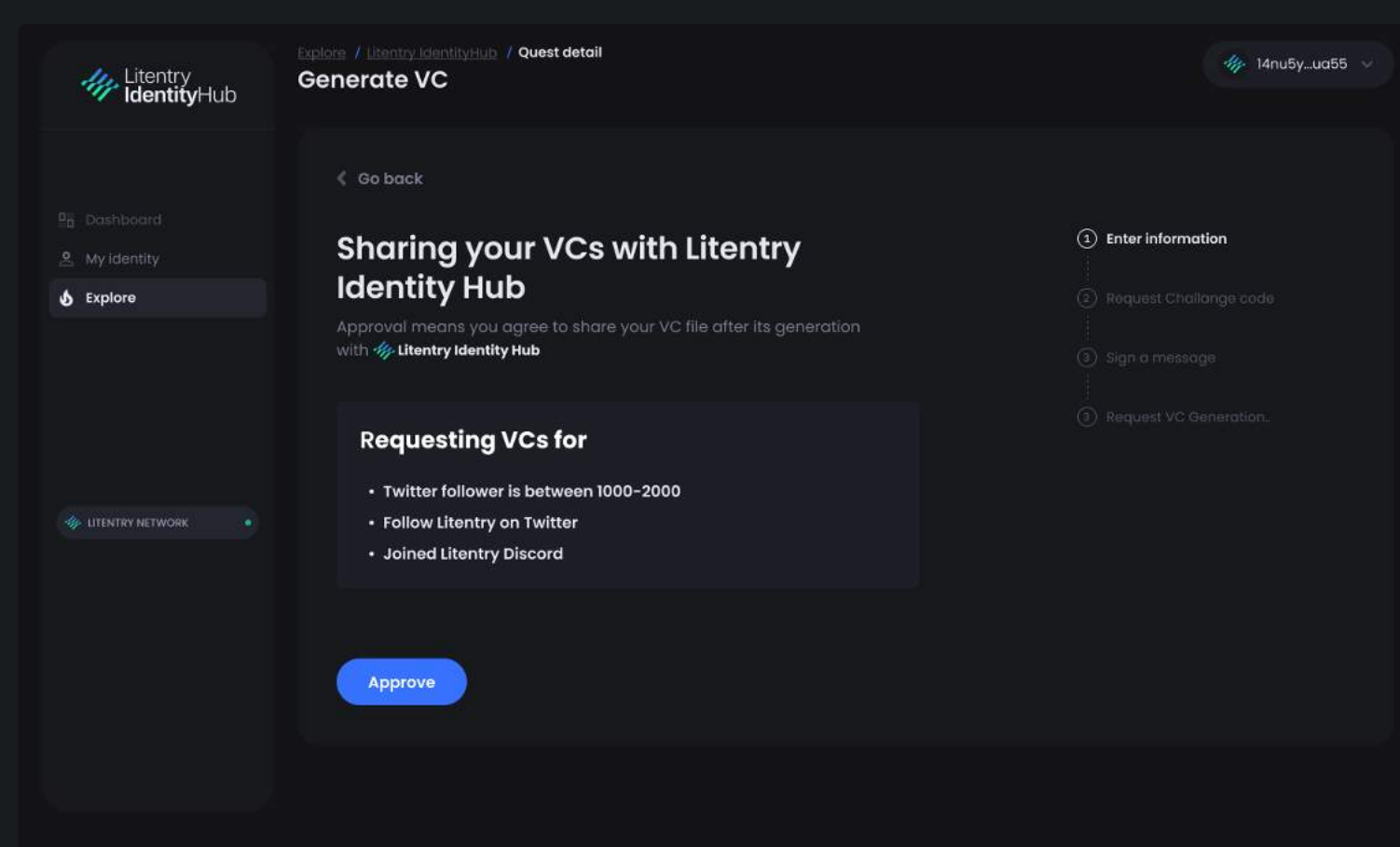
**Step 2:** You need to join the Litentry Discord before you generate a VC. To join the Litentry Discord, click the corresponding button and enter your Discord username. Then, follow the steps as shown.



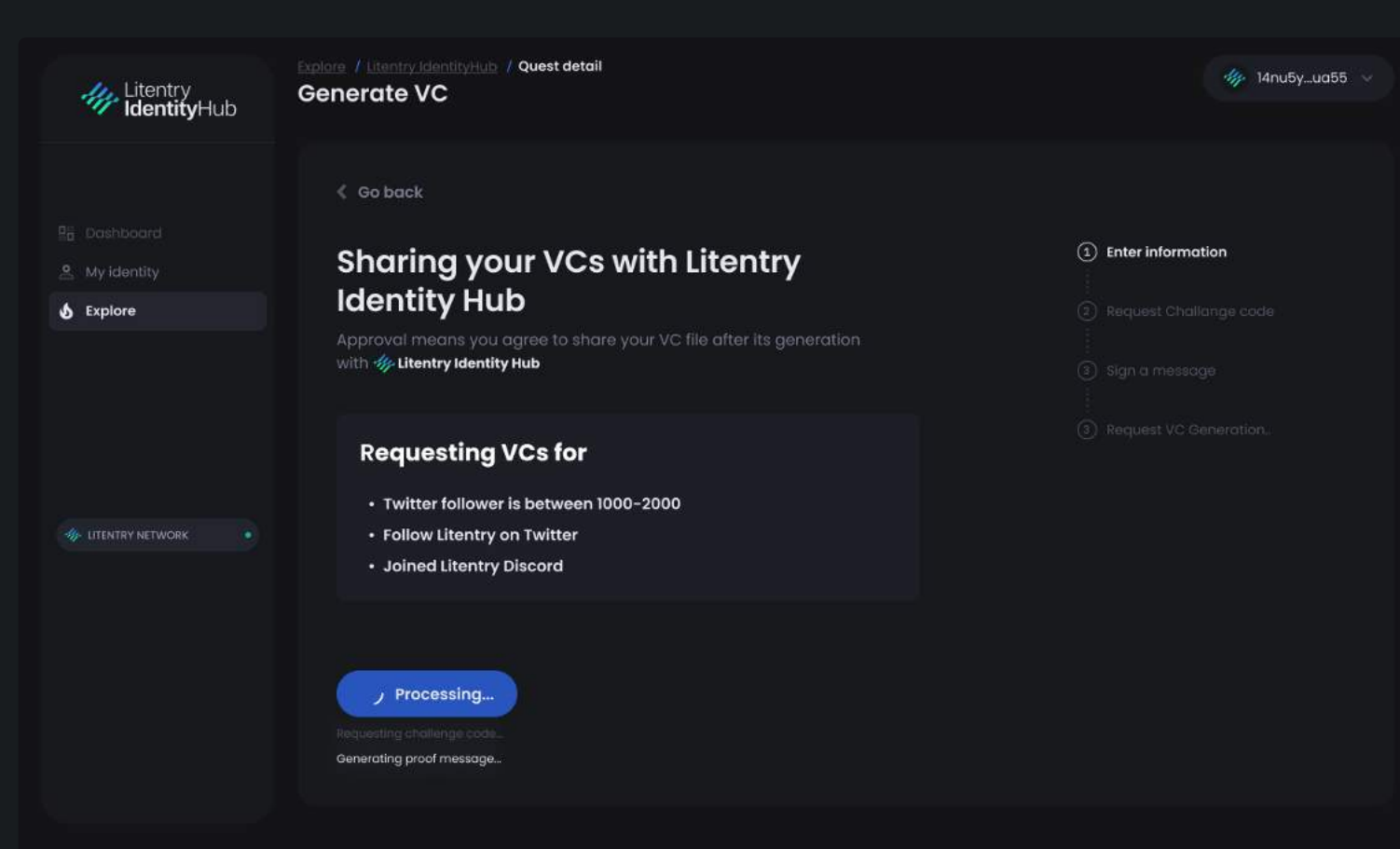
**Step 3:** Upon joining the Litentry Discord, return to IDHub and click the Generate VC button. Once clicked, the interface below will be displayed. Enter the link to the Tweet that you're requested to retweet in Credential 1 and enter the credential subject type in Credential 2 then click the next button.



**Step 4:** Next, you'll be required to approve the sharing of your VCs with the Litentry Identity Hub. The details of the VCs you're requesting will also be displayed. Click the approve button.



**Step 5:** The IdentityHub will automatically request a challenge code and generate a proof message for you.

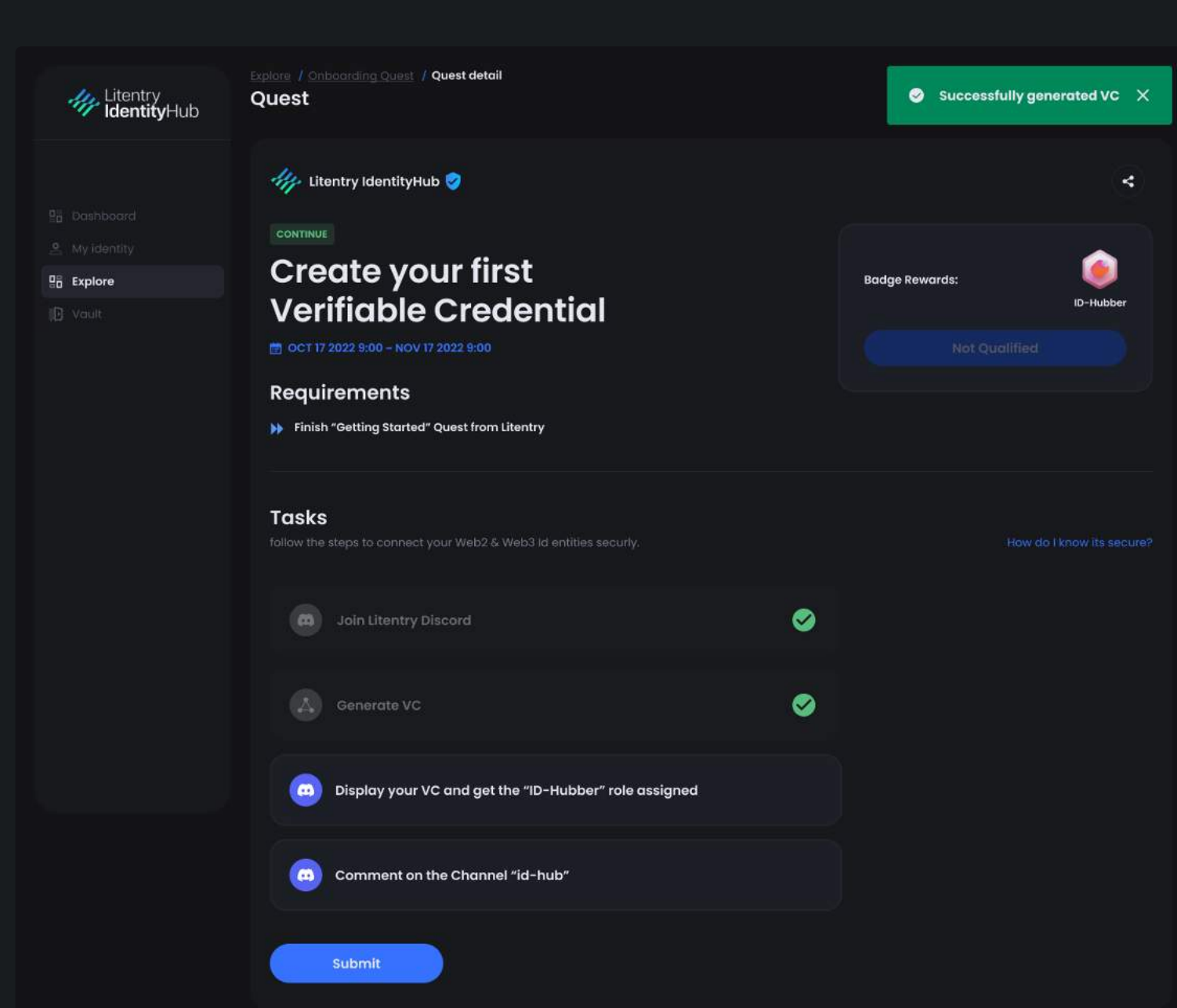


**Step 6:** Once processed, your VC generation request will take effect. This process may take 1-2 minutes as the Litentry TEE worker is retrieving your data from the open web.

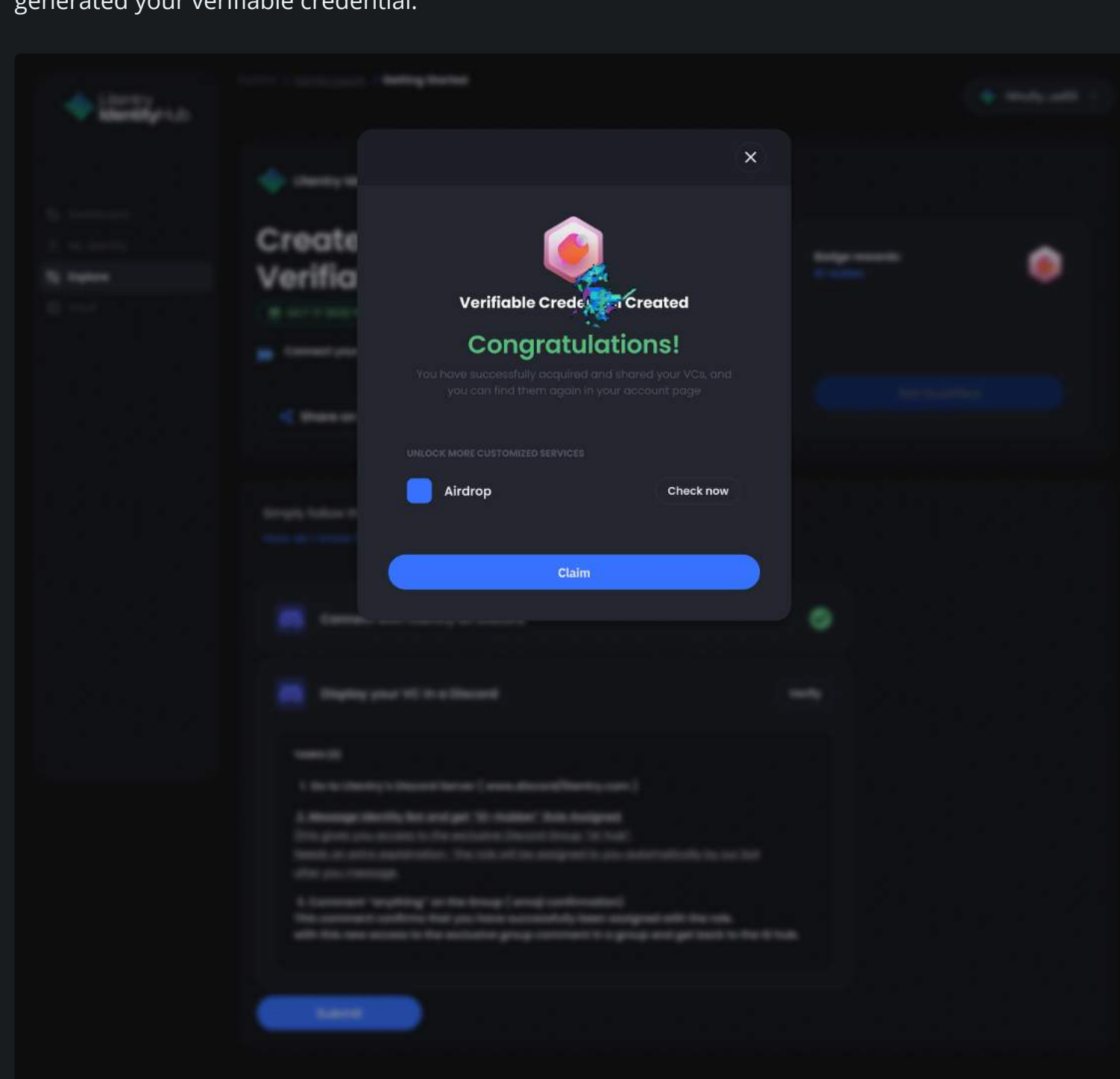


**Step 7:** Once your VC has been generated, proceed with the remaining tasks in the quest:

- Display your VC and get the "ID-Hubber" role assigned
- Comment in the Channel "ID-Hub"



**Step 8:** Once you have completed the tasks, you will have successfully completed the quest and generated your verifiable credential.





# Set up a Shielding Key

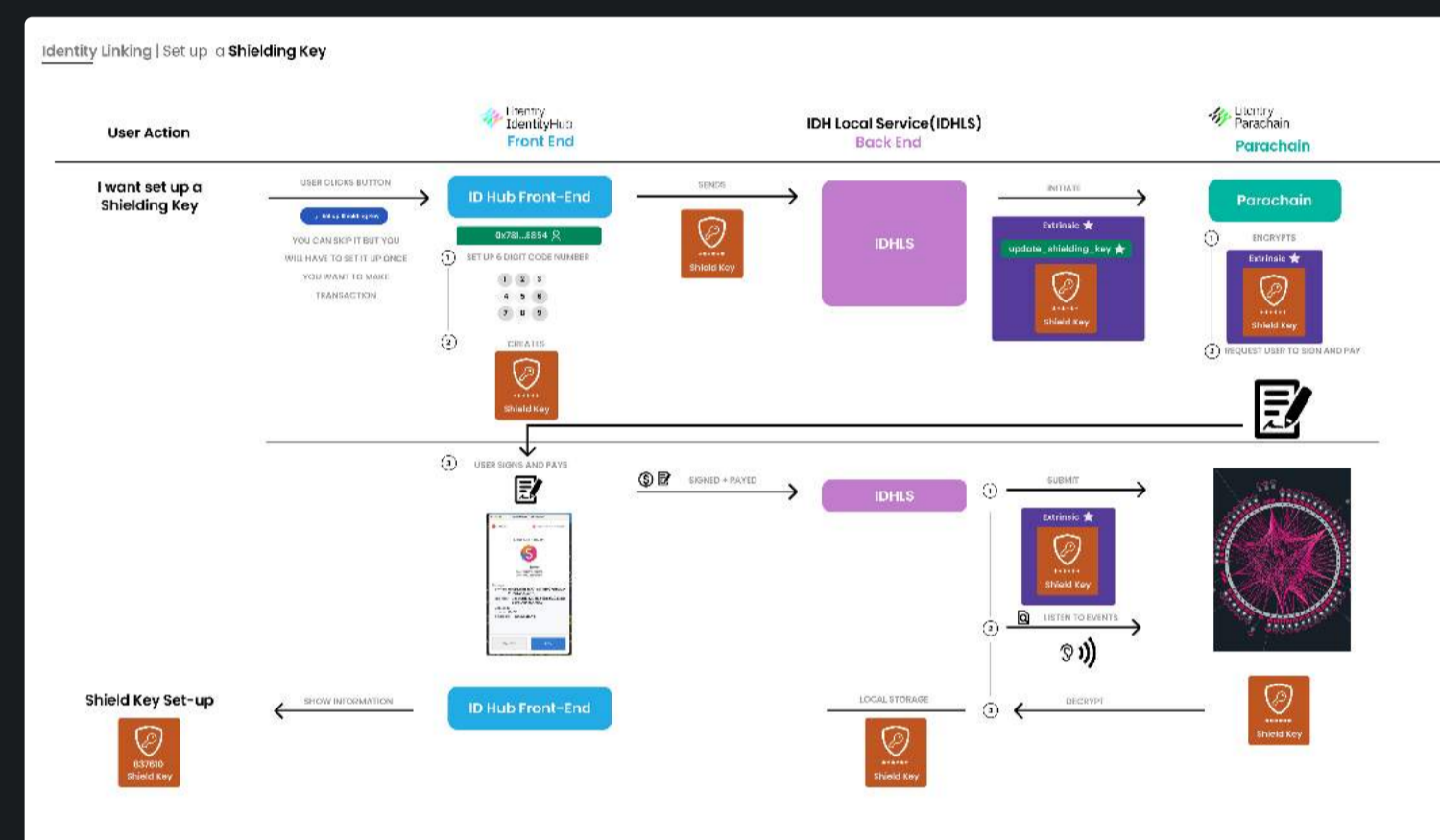
A shielding key is a 256-bit AES-GCM cryptographic key pair that is generated randomly and protected by the user password. Its public key is used by the Litentry TEE worker to encrypt user-sensitive information when passing back the data. It is used to encrypt all the data in the communication between the user and the Litentry Parachain. The shielding key is generated in the user's local environment and used by the IDHLS to isolate sensitive user data from the IDH server and all other third parties.

Overall, it is an extra layer of security on top of your Substrate private key (Account) used exclusively to encrypt your data for transmission and ensure only you or the Enclaves can decrypt it.

It is important to note that there are two types of shielding keys;

User shielding key - This key is applied to the on-chain data returned by TEE. TEE shielding key- is used in the other direction (user -> TEE) and is publicly visible.

## Set up a Shielding Key



1. User set a 8-character password as a root of their Shielding Key, this password will also be used to unlock IDHLS
2. IDHLS will update the Shielding Key to Parachain:
  1. IDHLS will use the 8-character password to generate the Shielding Key with algo 32byte AES-GCM
  2. IDHLS will initiate the `set_user_shielding_key` extrinsic to pass the user's shielding key and encrypt the extrinsic with the Parachain TEE's Shielding Key
  3. The user will sign and pay the extrinsic, the signing address must be the same as the user's sign-in address in the JWT, otherwise, IDHLS should forbid the user to continue.
  4. IDHLS will submit the extrinsic and listen to the parachain events
  5. IDHLS will find its parachain event by the user's main address and decrypt the event to get the result of `set_user_shielding_key` extrinsic
    - Success - finish set up, go to page *My Account*
    - Failed - show the error response and ask the user to try again
  6. IDHLS will then store the Shielding Key locally as the current only valid Shielding Key of the user
  7. The Substrate address that the user used to sign the `set_user_shielding_key` extrinsic will be stored as the `mainAccount` in the Local ID Graph\*\*. \*\* Also, local storage will create separate storage spaces for each main account, and different storage spaces will only be able to unlock with the shielding key of the corresponding `mainAccount`.



← Creating your First Verifiable Crede...

Previous

Next

FAQ →





# Litentry

## FAQ ⋮

This page discusses frequently asked questions. Please do not hesitate to get in touch via discord when you're running into trouble.

> **Can Litentry see the sensitive relationships between the accounts in my profile?**

> **What are the next chains that you would like to be more present on?**

> **How does the shielding key protect my data during processing and computation?**

> **How do I send my verifiable credential to a dApp or community?**

> **Will the IDhub's credentials be used for KYC?**

>



Previous  
[Set up a Shielding Key](#)

Next

[Litentry Foundation](#)



# Identity is fragmented.

## Tokenomics

### Introduction

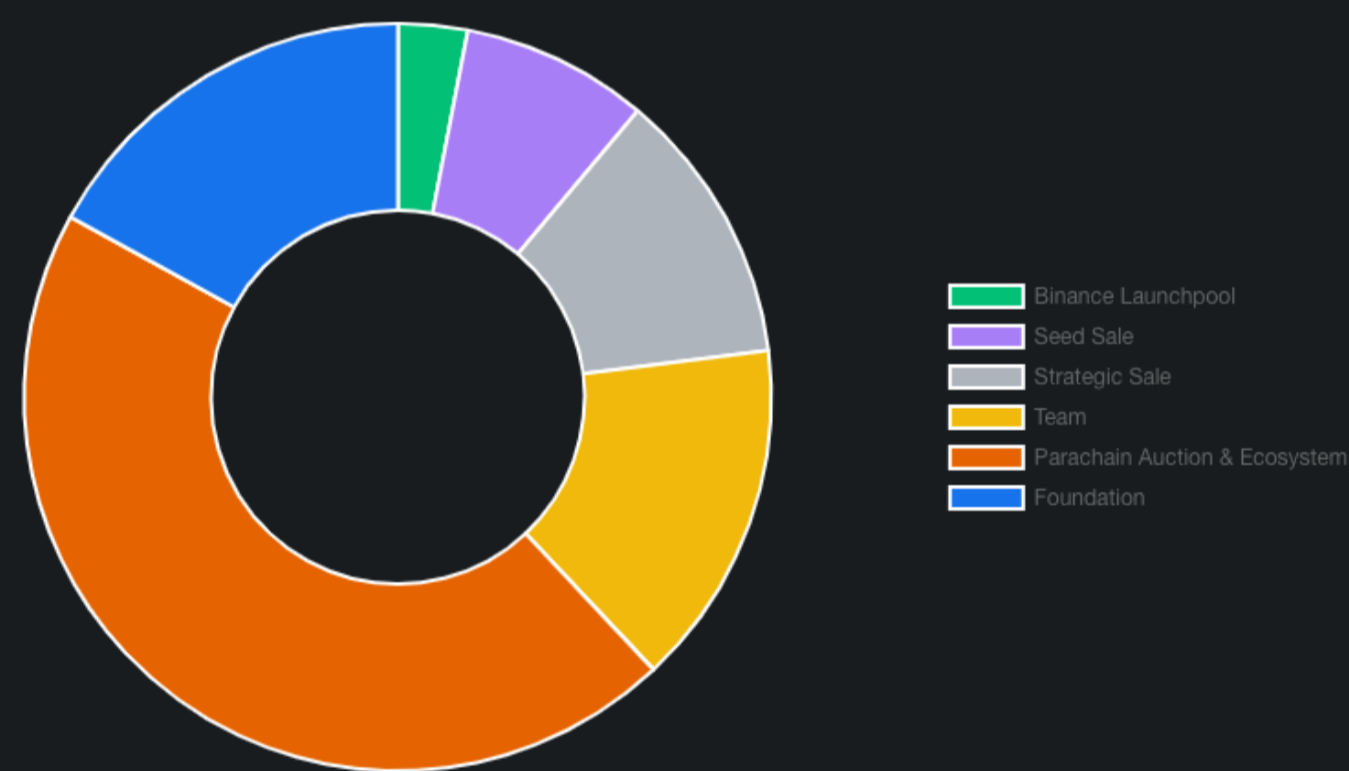
Litentry is a Decentralized Identity Aggregation protocol across multiple networks, it features a DID indexing mechanism and a Substrate-based credit computation network. The protocol provides a decentralized, interoperable identity aggregation service that mitigates the difficulty of resolving agnostic DID mechanisms.

**LIT token** is the native cryptocurrency of the Litentry, including Litmus parachain on Kusama and Litentry parachain on Polkadot with the following utilities:

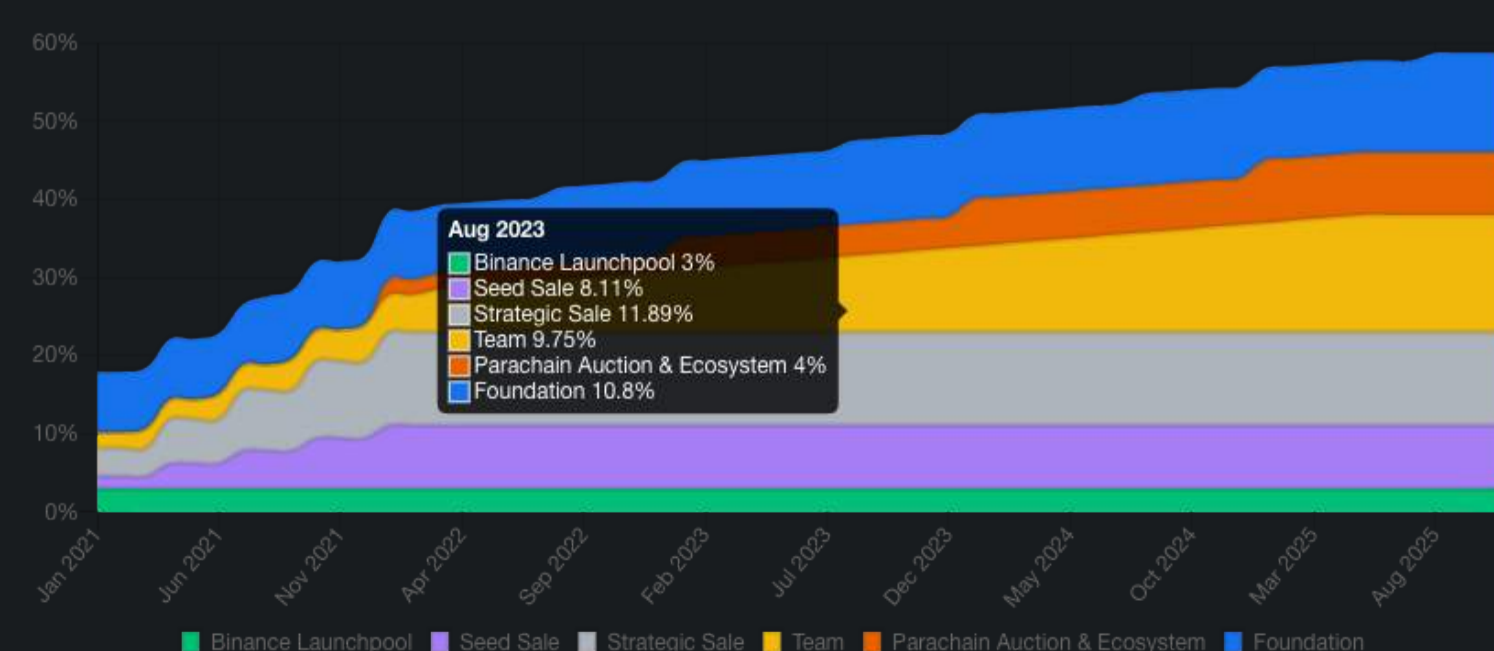
- Pay transaction fees in the network
- Incentivize collators and fund promotion campaigns to support Litentry parachain
- Empower governance mechanisms in the chain, including proposing referendums, the election of council members, voting in a referendum, etc.
- Serve as the utility token for identity product or service
- Collator staking

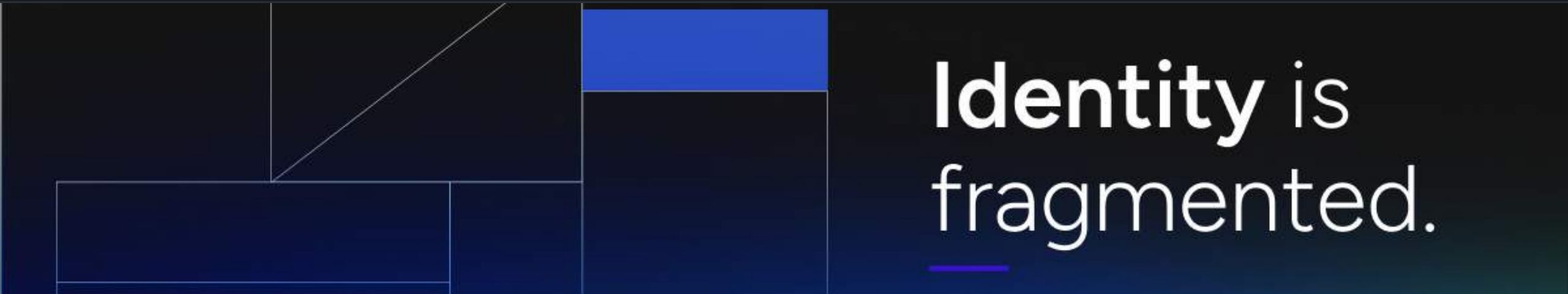
### LIT Token Allocation

Token Name	LIT
Binance Launchpool	3.00% of the total token supply
Seed Sale	8.11% of the total token supply
Strategic Sale	11.89% of the total token supply
Team	15.00% of the total token supply
Parachain Auction & Ecosystem	45.00% of the total token supply
Foundation	17.00% of the total token supply



### LIT token release schedule





# Litentry parachain inflation and collator staking

## Inflation Model

This inflation model of the LIT token aims to guarantee the security of the Litentry network in the long run.

The newly issued LIT can incentivize collators to provide block production services and delegators to stake to support the Litentry network. Litentry collators are run and maintained by the Litentry team at the early stage. We hope the community can participate in running and maintaining the collator nodes to make the identity protocol more decentralized, stable, and robust.

Besides, LIT holders can participate in building network security by staking their LIT and helping power the collator selection process.

Generally speaking, newly issued tokens from inflation are used to pay for a parachain slot in either Kusama or Polkadot. However, it will not be the case for Litentry. Because Litentry has reserved 45% of the LIT for slot auctions and building the ecosystem, the Litentry team has enough tokens to acquire slots on Polkadot and Kusama in the upcoming decade. Inflated LIT tokens will not be used for team or ecology building.

According to the community's opinion and based on the overall market situation, if Litentry parachain enters the inflation model, it will harm the interests of the community and LIT holders, so we will continue to burn the equivalent amount of new issuance LITs from the inflation model quarterly for the next two years after collator staking is deployed.

The new issuance token for the next two years will be :

```
newToken = 100,000,000 * [(1+1.5%*1/2+2.5%*1/2)*(1+2.5%)^-1] = 4,550,000 LIT
```

The amount of LIT that should be burnt each quarter is :

```
burnLIT = 4,550,000/8 = 568,750 LIT
```

The following are the key parameters of the inflation model:

- Target a 1.5% annual inflation rate in the first six months (~=1,296,000 blocks),
  - 0.5% goes towards incentivizing collators, 1% is for users that stake their LIT tokens
  - After six months, a 2.5% annual inflation rate will be applied, 0.5% goes towards incentivizing collators, 2% is for users that stake their LIT tokens
  - 568,750 LIT will be burnt from the ecosystem wallet each quarter and continues for eight quarters (two years).
- Note:** The proposal to burn the planned 568,750 LIT was tabled as referendum #0 and has been voted and passed. You can check the details here: <https://litentry.subsquare.io/identity/referendum/0>

## The staking pallet is enabled at block height 1,075,800

Figure 1.1 shows the pre-inflation LIT token release schedule. A small portion of these released tokens is from the team, but most of them are from crowdloan rewards of the slot auctions. According to the existing token release plans, about 55,000,000 LITs will be in circulation by the end of 2024.

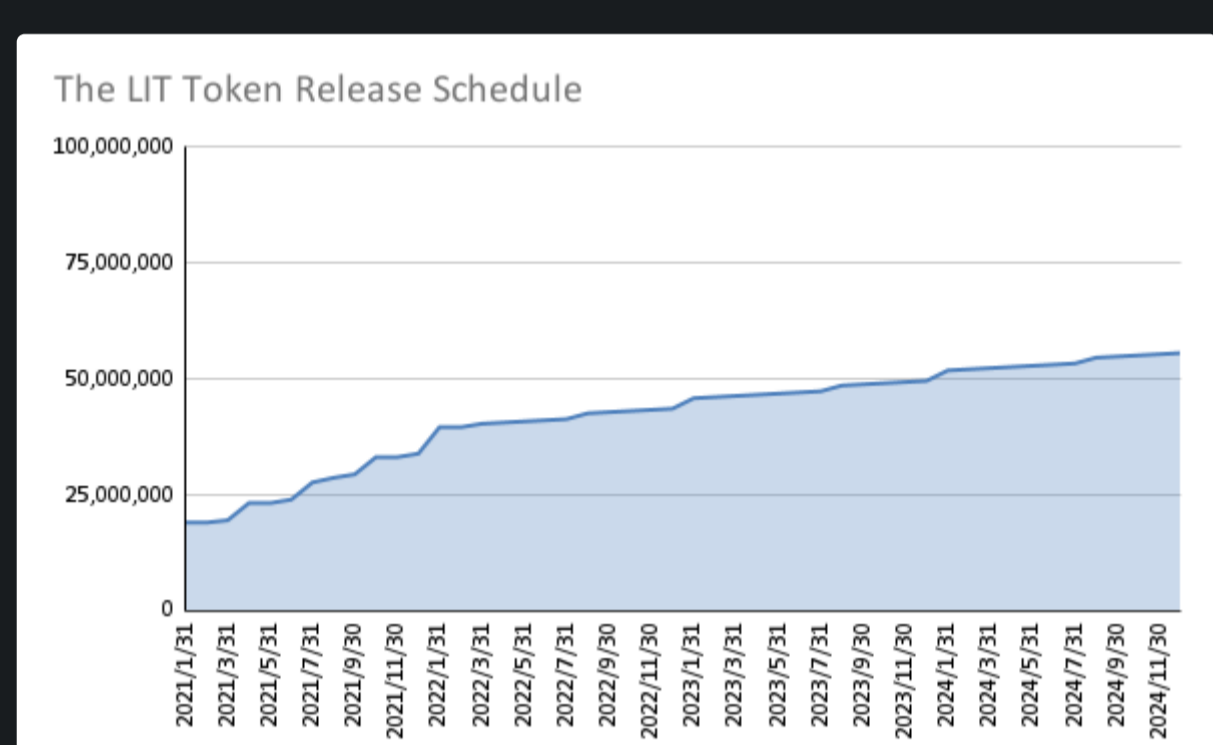


Figure 1.2 presents the inflation rate after collator staking is deployed. The inflation rate will be 1.5% for the first 6 months and 2.5% after 6 months.

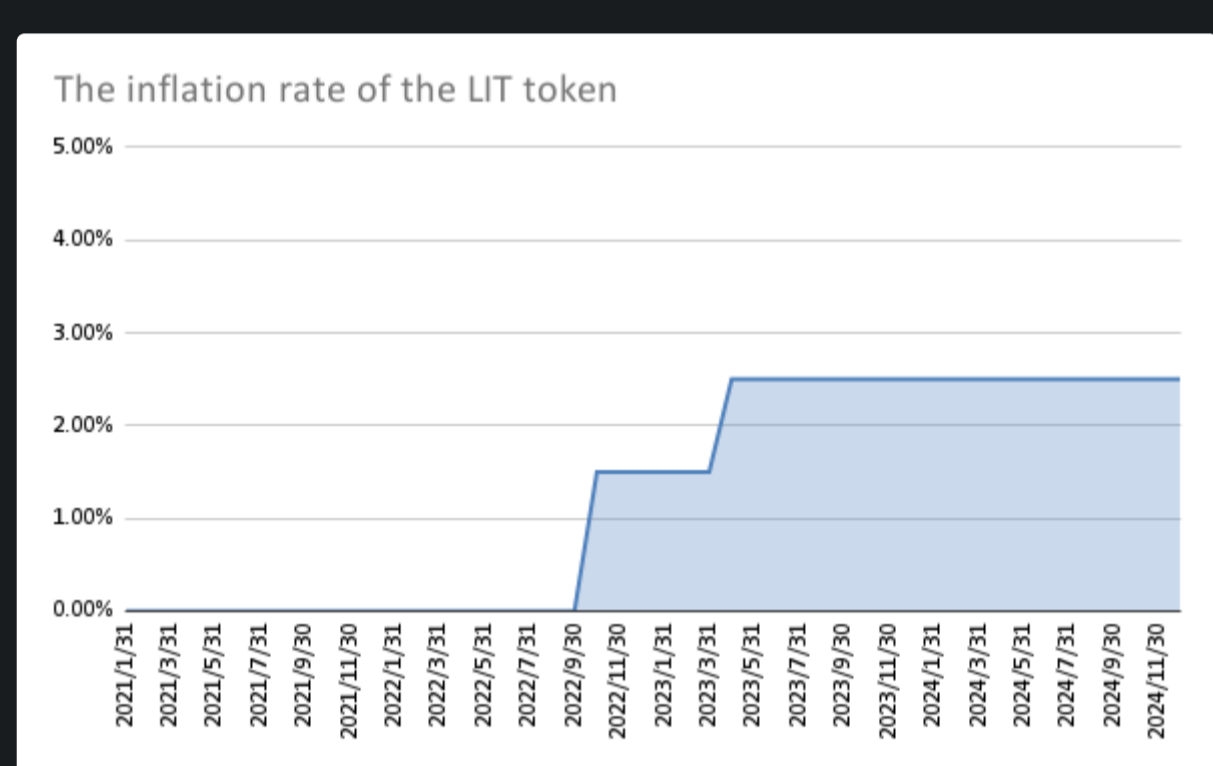


Figure 1.2 The Inflation rate of LIT token

To offset the reduced number of LITs in circulation, we introduced the inflation model, as shown in Figure 1.2. When the collator staking functionality comes online, the LITs in circulation will be lower. We can not accurately predict the number of LITs that will be removed from circulation, but this does not prevent us from making a hypothesis.

Assuming that the LITs participating in collator staking receive an annual reward of about 10-20% and that 2% of the inflation token (i.e., 2,000,000 LITs) is reserved for collator staking users. About 10%-20% of the total LITs (i.e., 10,000,000 -20,000,000LITs) will be taken out of circulation. The estimated circulation of LIT is shown in Figure 1.3. There will be about 40,000,000 LITs in circulation by the end of 2024 after collator staking functionality is deployed.

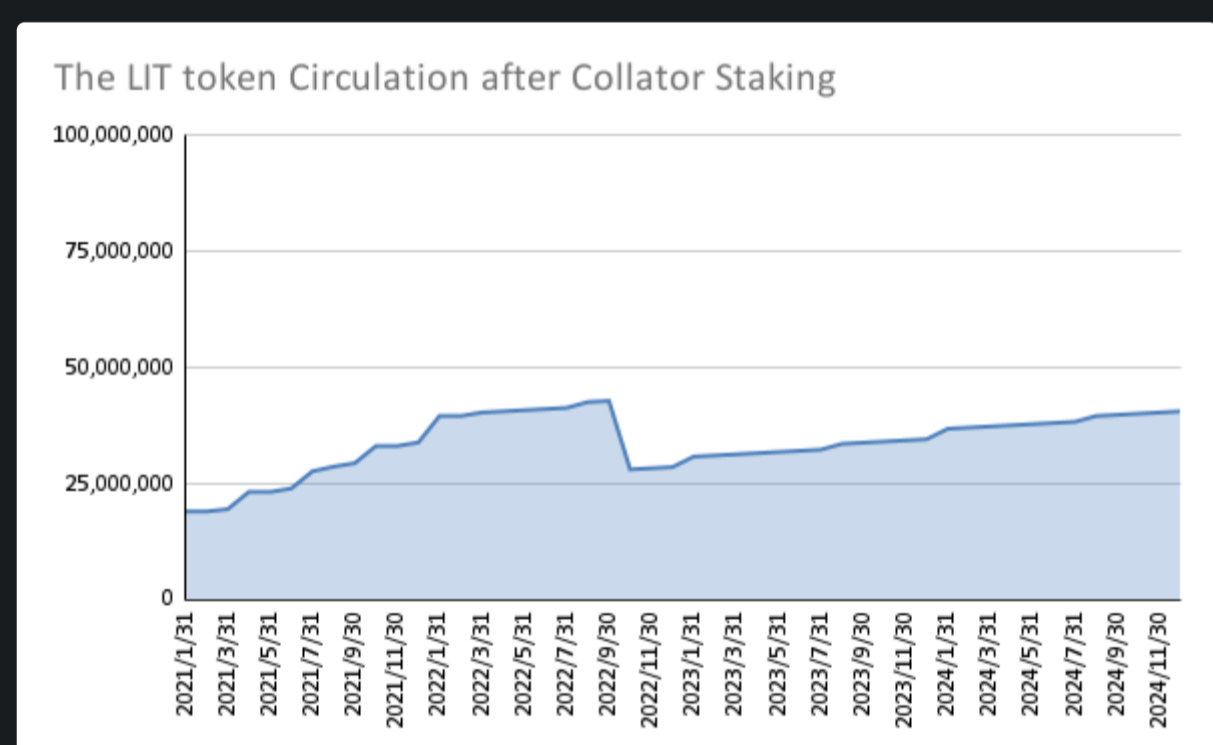


Figure 1.3 The LIT token circulation after collator staking

## Collators

Collators are full nodes for the parachain and the relay chain. They maintain parachains by collecting parachain transactions from users and producing state transition proofs for Relay Chain validators. Even though collator nodes do not contribute to the safety of the network, they are an integral part of the parachain. The Litentry Network will become more stable and robust with a good number of high-quality collators.

Opening collators to community participation is an essential step towards community governance in Litentry parachain. In the early stages of Litentry parachain, collators are run and maintained entirely by Litentry. After opening up to community participation, Litentry will continue to run and maintain some of the collators and will gradually open up the number of collators to community participation. **All revenue from the collators run and maintained by Litentry will go to the treasury, and the community will decide how to use it.**

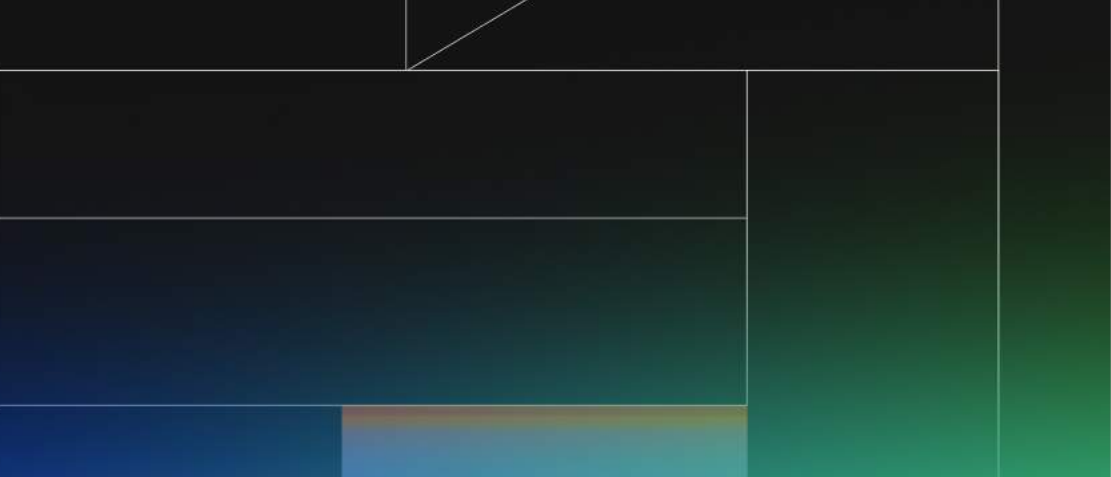
Requirements for an organizer candidate include one's machine, bound, account, and community. Maintaining a collator requires a certain investment of time, technology, and hardware by the candidate, so in our inflation model, about 500,000 LIT new issuance tokens will be used to incentivize the community to participate in the operation of the collator actively.

## Staking

LIT holders are encouraged to participate as a nominator. LIT holders can select a list of collators that they trust and stake the amount of LITs to support them. If some of these candidates are elected as block producers, they share the block rewards or the sanctions on a per-staked-LIT basis.

- From the event of unstaking, the unstaked amount remains locked for about seven days.
- Rewards are received as unlocked LIT tokens and are not automatically added to the stake.

Parameters	Value	Description
Session duration	1800 blocks (6 hours)	a specific number of blocks around which staking actions are enforced.
Minimum staking per candidate	50 LIT	the minimum amount of tokens to delegate candidates once a user is in the set of delegators
Maximum delegators per candidate	1000	the maximum number of delegators, by staked amount, that a candidate can have which are eligible to receive staking rewards
Maximum delegations	100	the maximum number of candidates a delegator can delegate
Reward payout delay	2 sessions (12 hours)	How long until you get the staking rewards
Add or increase delegation	takes effect in the next round (funds are withdrawn immediately)	How long until your funds will take effect
Decrease delegation delay	28 sessions (7 days)	How long until your funds will be transferrable after unbonding
Revoke delegations delay	28 sessions (7 days)	How long until your funds will be transferrable after unbonding
Leave delegators delay	28 sessions (7 days)	How long until your funds will be transferrable after unbonding



# Ecosystem Wallet ⋮

This page is mainly used to track and record LIT transfers from the ecosystem Wallet. The ecosystem wallet address is :

`0x9cdF4E1347328416daF17335CaF6A314201CC1Dd`

and the fund will be mainly used in the following use cases:

- Support both Polkadot and Kusama crowdloan rewards
- Support identity Hub products, e.g. identity staking
- Litentry parachain ecosystem building
- Support early stage collator staking rewards

Tx Hash	Comments	Transferred Balance
<code>0xc8c7b0d784badeada4e23990b687b846acad9878c9bb11595980101189d6bb3c</code>	Move Litmus crowdloan reward to token bridge	487,219 LIT
<code>0xef8cc6f9035cfcb97e03b78ccaca53a3216cecd69eab45f3dfa2c2a010f86b96</code>	Move Litentry crowdloan reward to token bridge	2,000,000 LIT
<code>0x66fd7fde697e67f0906dd1c02a6c716e71b4dff6e65a05f365472e5a400c1f41</code>	Move Litentry crowdloan reward to token bridge	1000 LIT
<code>0xe95a3e59c34f389a67bd94fdff65861d2121b4aecc517787227e08ea8773cb6</code>	Move Litentry crowdloan reward to token bridge	8,443,219 LIT

← [Litentry parachain inflation and colla...](#) Previous Next [Team Wallet](#) →

# Team Wallet



The Team wallet address is

0x65E4a77536d47Bf42Db6817373939A63C00A0904

and the fund's purpose is mainly used to support the development team's salary compensation.

Tx Hash	Comments	Transferred Balance
0x733971ef9581f6e885a57e d59656c9bcfe5ca49b38d734 5e4af8cee454e03de5	Monthly release	250K
0xbe5ca0c80083b6660ce0a2 3f4a28c5f878cfbdfad709e91 0dcd59755ca3d0109	Monthly release	280K
0x638e04cdb0b23d143a5f7d f17808ed745984beed31acb0 eba610ca93f4757565	Monthly release	250K
0x5cf1677f071827bcc2e65d0 d89a9393d6abe388f0d632f0 f45e083752680c87a	Monthly release	250K
0x61d9dcf744219524538c6f 5c5ca1024497d3eeb896caf4 1cb6fe21b7c4fd7eb3	Monthly release	250K
0xec948ccf6c5e9455a8e6d4 aa9e5ab999be2e55a780204 e6994f0651b0df0a4a4	Monthly release	250K
0x3244c1e83a8101f31f1b9e 34551c0eedd7a5465861298 dbc864cd32aa4ec1092	Monthly release	250K
0x5336c55f6afe6c98a0b25b 9191f0ae493343f8d6e68fcd7 2c8cc165710556d43	Monthly release	250k



Previous  
Ecosystem Wallet

Next

Foundation Wallet





# Litentry

## Foundation Wallet

The Foundation wallet address is

```
0xD481E3499c8Ef073913513073C97d4d89bb9797e
```

and will be used as following aspects:

- Support exchange Listing
- Market Maker Fund support
- Community event support

Tx Hash	Comments	Transferred Balance
0x7258418f91a70d5aa6f1e962acd092ca7196553a213ac8a1cf7458e3d3270b5e	Used as minimum staking of the Litentry hold collators	20000 LIT

← [Previous Team Wallet](#) [Next - Misc Glossary of Terms](#) →

# Make your identity tangible without being seen

## Glossary of Terms

Term	Definition
Auction (Parachain)	Parachain auctions are how non-common-good parathreads win a slot to become a parachain
Aggregated Identity	An identity that consists of multiple data streams from web3 & web2 accounts and platforms.
Bounty	A mechanism which works in some sense as the reverse of a Treasury Proposal, allowing the Polkadot Council to indicate that there is a need to do some task for the Polkadot network and allowing users to receive DOT in return for working on that task.
Council (Polkadot)	An on-chain entity that consists of several on-chain accounts. The Council can act as a representative for "passive" (non-voting) stakeholders.
Decentralised Identity	An identity that consists of multiple decentralised data points across web3 platforms and blockchains. A D-ID is managed and accessed by decentralised applications.
Governance	The process of determining what changes to the network are permissible, such as modifications to code or movement of funds. The governance system in Polkadot is on-chain and revolves around stakeholder voting.
Identity Owner	The user who has provided access to his various accounts and in this way creates his aggregated identity.
Identity Subject	A label added to the the users aggregated identity determined by their on-chain credentials. e.g. 'Longterm Holder'
NFT	A non-fungible token is a unique and non-interchangeable unit of data stored on a blockchain.
Referendum	A vote on whether or not a proposal should be accepted by the network. Referenda may be initiated by the Governance Council, by a member of the public, or as the result of a previous proposal.
Registrar	After a user injects their information on-chain, they can request judgement from a registrar. Registrars can set a fee for their services and limit their attestation to certain fields.
(to) Second (Polkadot)	Agreeing to a proposal by putting up tokens equal to the original bond
TaskFI	Completing tasks to earn tokens
Treasury (Polkadot)	The Treasury is an account that accumulates funds by inflation as well as by taking a portion of transaction fees and slashes.



Previous  
[Foundation Wallet](#)

Next - Misc

[Related Links](#)







# Litentry

## Related Links

Would you like to learn more about Litentry? Great! Take a look at some of the links below.

Title	Link
Litentry Website	<a href="https://www.litentry.com/">https://www.litentry.com/</a>
Twitter	<a href="https://twitter.com/litentry">https://twitter.com/litentry</a>
Telegram	<a href="https://t.me/litentry">https://t.me/litentry</a>
Discord Server	<a href="https://discord.gg/WR4RPTHjAz">https://discord.gg/WR4RPTHjAz</a>
Medium Blog	<a href="https://litentry.medium.com/">https://litentry.medium.com/</a>

← Misc - Previous  
**Glossary of Terms**

Next - Misc  
**Media Assets** →



# Litentry

## Media Assets

Hi-res Litentry logos for media

Logo Litentry Icon Symbol Litmus

PNG



SVG



Misc - Previous  
Related Links