

# Introduction

Litentry Protocol is helping support change that in favour of a user-centric internet with the blockchain.

## Concept of User-Centric Internet

As we entrust safe-storage of our passwords and online activity data to third parties on the app-centric internet of today, we are often asked to comply with ambiguous and unfair privacy policies handing over control our own data. As a result our data is often harvested and used in ways we do not have a say in.

Litentry is helping support change that in favor of a user-centric internet with the blockchain. That means the internet should dominant by the users. We decide what services or apps to users, the decision should not limited by the migration cost or specific terms. The profit generated by the user data should go back to user, instead of mainly flows to service provider / companies.

Litentry includes an identity-based network and related tools, as a whole it consists of a decentralized identity authentication and user activity data management infrastructure.

The protocol and network designed at start is not only fits to the internet, but also fits to all the digital services in the real world.

## Highlights

### **Blockchain Powered:**

Litentry is build on Substrate, which inherits great features and best technologies in the Blockchain industry. Litentry aim to be the Parachain of Polkadot Network and benefit from the thriving cross-chain ecosystem and share security.

### **Identity Management:**

Get back the control of user access history and privacy data generated in the apps or services, user's identities are anonymous and independent from each other.

## **Decentralized Storage:**

A user is able to store these encrypted identity related data on a decentralized storage like IPFS or Arweave, or on-chain Database. The data is protected by the access control of decentralized storage.

## **Identity Staking:**

With a transparent protocol, an identity owner could stake his identity into identities pool and get regularly reward with data protected. Litentry enable anonymous data matching and permissioned data query. With data protected, Litentry monetize the identity data and benefit identity owner and DApp.

## **Decentralized Ecosystem Contributor:**

Share the identity anonymously between different platforms, a user do not need to create multiple accounts in order to use different services, and prevent these services for building user profile. No password, no registration, no migration barriers. Litentry and decentralized ecosystem benefit each other.

# **Concept of Decentralization**

The decentralization of Litentry includes following aspects:

## **Decentralization of identity storage:**

User data, including identity credential, should be storage in the a decentralized storage, instead of the central data server of service provider, currently we support IPFS, in the future we will have other storage support like Arweave or Litentry native on-chain key-value store.

## **Decentralization of identity authentication:**

The identity validator connect to the decentralized network periodically, and it could validate the authentication request independently.

## **Decentralization of identity relationships:**

The relationship of data, and identity could be validated with cryptographic calculation, and it is also record in the decentralized network instead of regular centralized service like Certificate Authority used in HTTPS protocol. Different identities belong to owner are not discoverable, and protected by Schnorr25519 Algorithm.

### **Decentralization of Identity Data Allocation:**

The user data generated when using third party applications/services could be processed by the resolver function on Litentry Network, thus provider user a trustworthy data, with allocate data from variant origins, user are able to get valuable user profile like health info, shopping history, etc.

## **Definitions**

### **User:**

The origin of data, it a person who holds identities or IoT devices.

### **Identity:**

It is a generalized concept of identity, not only include the identity of person, but also any thing could generate claims like IoT devices. A person could own multiple identities, like an identity only related to musics, an identity only for data in Germany, or an identity as a game player. It is an anchor to its related data, the id of it does not have any meaning.

### **External Data:**

data generated when using the applications/services and anchored with identity, like the shopping history when a user shopping in e-store, or the age data read from the aforementioned age proving request.

### **Authorization Token:**

A piece of data prove the read or write permission to identity's external data. Like the permission to read the age data of a person.

## **Protocol**

## Concept of Decentralization

The decentralization of Litentry includes following aspects:

- **Decentralization of identity storage**: User data, including identity credential, should be storage in the user's owned devices, instead of the central data server of service provider.
- **Dentralization of idenity authentication**: The identity validator connect to the decentralized network periodically, and it could validate the authentication request independently.
- **Decentralization of identity ownership**: The relationship of data, person, and identity could be validated with cryptographic calculation, and it is also record in the decentralized network instead of regular centralized serviec like Certificate Authority used in HTTPS protocol
- **Decentralization of Identity Data Allocating**: The user data generated when using third party applications/services could be processed by the resolver function on Litentry Network, thus provider user a trustworthy data, with allocate data from multiple applications/services, user are able to create valuable user profile like health info, shopping history, etc.

## Definitions

- **User**: The origin of data, it a person who holds identities or IoT devices.
- **Identity**: It is a generalized concept of identity, not only include the identity of person, but also any thing could generate claims like IoT devices. A person could own multiple identities, like an identity in Germany, an identity as E-Resident in Estonia, or an identity as an game player.
- **Authorization**: The permission in the reality or the claim in the blockchain world. It is a piece of data that could prove the ownership to a capability or a real thing. Like the permission to read the age data of a person, or the ownership of a 3D printer on a certain day.
- **External Data**: It is the data generated when using the applications/services, like the shopping history when a user shopping in e-store, or the age data read from the aforementioned age proving request.

## Network Interoperability

Based on Substrate Network, Litentry aims to become a fundamental part in the Web3 infrastructure.

// TODO

- **Network Layer**: Polkadot is here to connect different blockchains
- **Runtime Layer**: The Litentry Pallet could be used for other Substrate network builders.
- **Application Layer**: Small business could build smart contract on Litentry network.

# Token Economy

## Economy Participants

### Identity Staker:

The users who has an identity record on chain, and has stake the identity into identities pool.

### Identity Validator:

After the staking identity of identity staker is confirmed on chain, he will become identity validator for the next few blocks.

### External Storage:

An decentralized storage records all the related data of the identity (Currently IPFS, in the future we may add more database support)

### Node:

The maintainer of the network, it task is to record the state of the network, and respond to data matching queries, sending data access request to external storage, and use off-chain worker to validate the correctness of the identity staking data.

### Data Buyer:

An entity have either one of following two types of requirement is data buyer: Arbitrary identity data: request a matching identity (list) according to the types/requirement data of certain

identity: in this case, the buyer will need a authorization token from the identity.

### **Data Origin (Data Generator):**

there are three types of data origin:

1. Decentralized services / apps, it generate the data when user interact with dapp is signed by the data generator.
2. Traditional app / services, they may offer data migration services, it maybe signed or not. If it is signed, and data generator is register on the Litentry network, the data generator also benefits from data queries.
3. User generate the data by his own.

## **Identity Staking Process**

### **Identity Preparation:**

A identity staker who want to stake an identity into identities pool, need first has the required data type and format anchored to this identity, and the data is stored correctly in the External Storage.

### **Identity Data Picking:**

Then the user chose which kind of the data he want to staked into the pool, only the picked type will be available for data matching, also the more data he chose, the more benefits he get, staker will need also pay a validation fee to the network and a basic staking deposit.

### **Staking Identity Validation:**

The data will be send to random selected identity Validator on chain, identity validator will try to prove if the data is correct. Here is three possibilities:

- If any one of the validator reject the data, the staking process is failed, and part of thee fee will be returned, validator got no money.
- If all validator proved the data, and identity with authorized types will goes into identities pool, fee paid to validator.
- Same as above, If all validator proved the data, and identity with authorized types will goes into identities pool. But if in the next 30 block a malicious data is found for this

identity, a part of reward of all the approved validator will be slashed, this slashing amount is decided by the existing blocks number (in this case, 30) of the existing malicious data.

### **Staking Identity Finalization:**

The value of the identity data will be judged by its completeness and its relevance and according type and identity will be stored into on-chain storage. No identity data is stored, afterward each block the identity owner will receive rewards, and the reward is bound to the staking identity until it retire from the identities pool.

### **Staking Identity Retirement:**

After certain block, the identity will be retired from the identities pool, after that, the rewards bound to the identity and the deposit will be release. User could update the identity data and then staking it again.

## **Identity Query Process**

According to the different data type required by data buyer. There are two types of query

### **1. Matching Query**

#### **Data Matching Request:**

Data buyer require a matching data with selected data type and criteria, which consist of a matching query.

#### **Data Matching Pre-Making:**

Node in the network will now start generate a list with random picking identities has the required type of data. The length of the list is decided by the fees the data buyer paid, the more data buyer pay, the bigger the list is. And the on-chain randomness make sure each time the result list would be different, so user has motivation to make the query two times.

#### **Data Selection:**

The list is sent to the external Storage and the data is sent back to the network and is received by the off-chain worker. Now the off-chain worker will use the selection algorithms to get the best suitable identity data for the buyer, and send the result back to user. The match winner will get the most of the fee paid by the buyer, others fee goes to the others in the list and data origin, also Litentry get small part of the fees.

## 2. Target Identity Query

### Identity Data Request:

Data buyer require a matching data with selected data type and identity id, the query also include an authorization token signed by the identity owner.

### Request Validation:

Node in the network will check the authorization token issuer, receiver, and validate block number. And finally decide if it is a valid request.

### Request Finalization:

A data request event is triggered, the off-chain work start to request the certain data from external storage, once it receive it, it will send a http request back to the data buyer. Fee is all paid to the identity owner and its related data origin, Litentry get a small portion.

## Incentivization

Basics: LIT is the native token of Litentry Network, each block the network will give a fix amount reward to the identities owner of identities in identities pool. In the staking finalization process, the value of the staking identity will be quantitated, so that the block stake reward will be shared according to identities staking value.

### For Identity Staker:

The profit of identity staker are from two parts,

1. Block Reward: Once staker data is into identities pool, Identity Staker will get the reward each block.



2. **Matching Fee:** Once the matching success with staked identity, the owner will get paid. So in the early stage, the matching request are not big enough, the identity staker will mostly benefits from block reward. When the network get more user and become mature, the share of matching fee will more evenly distributed to the quality identity staker. Thus a staker's main benefit is matching fee.

### **For Identity Validator:**

They are motivated to become Identity Staker, so it is part of the responsibility to validate the correctness of the data. If the origin of data is generated by the identity itself, then the validator will get reward once the data is used, since the validator has proved the authenticity of the data.

### **For Dapp as Data Origin:**

Explicit benefits is the grants from Litentry Foundation, each success match will pay fee back to data origin. Implicit Benefits are: It is a new way to attract new users since users could harvest their own data. And for user from other DApp, there is no migration cost, they have motivate to make innovation instead building business protections.

### **For Node:**

As the network maintainer, they will get native token reward from the network.

### **For External Storage:**

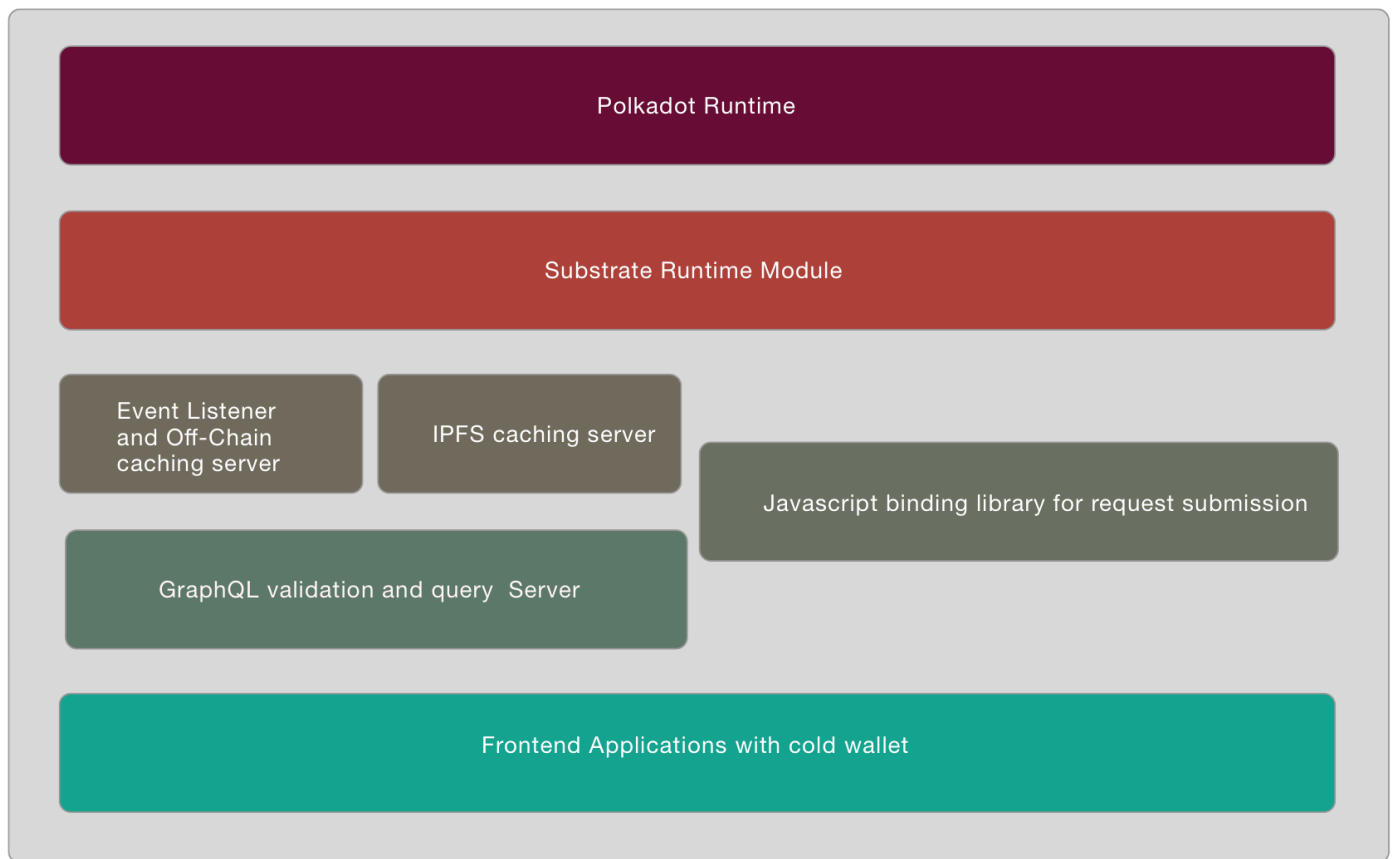
User pays their own storage fee to the external storage, so they are willing to provide their services to our network.

## **Architecture**

**The protocol is mainly constructed with following parts:**

- Litentry Runtime
- Litentry Authenticator Mobile App
- Litentry DApp Playground
- Litentry IPFS Data Center
- Litentry SDK

- Litentry GraphQL Caching Server



Other than web 2.0 architectures, we are suppose to build a decentralized ecosystem with Blockchain as backend services than cloud or single node server.

## Runtime

Litentry Runtime is built with Substrate, it inherits great features and best technologies in the industry.

We use offchain worker to generate identity related data, and thus remove the uncertain and privacy issue by the client side applications.

We aim to be the Parachain of Polkadot Network. We will also benefit from the thriving cross-chain ecosystem.

## User Side

User has full control of identity data, data generated from Apps flows to user's decentralized storage like IPFS or Arweave. User's identities are anonymous, cryptographic separated.

## **Litentry Authenticator**

On user side we have Litentry Authenticator as user's mobile data hub.

Personal users would like to use an application to manage all its identities, it could also become a Hub connected to different interest IoT devices. For example, directly buying the authorization or the data from other IoT devices. With the advantage of GPS of mobile phone, it could further integrate with LBS (Location Based Services).

In order to work in a fully decentralized scenario, itself also need to integrate a cold wallet, where could keep a user's private key in a secure environment provided by Android or iOS.

## **DApps build with Litentry Protocol**

With Litentry SDK, developers could easily build fully decentralized Apps or Services. User could directly signin without password, without registration. Simply with Cryptographic QR code. App could use IPFS, Arweave or even on-chain key value database for storing user data, instead of storing data on backend server.

By this we are convert to an app-centric internet to a user-centric one.

## **Litentry IPFS Data Center**

Litentry uses OrbitDB to offer an IPFS database support. In Data Center, user may check their identity related data and tokens.

In the future we will implement Arweave and on-Chain key value storage.

## **Middleware Layer**

It mainly includes:

- Event listener and off-chain caching server: With cached data it reduces the query load on the blockchain, furthermore, it saves the caching data on the centralized database in order to improve the speed of application-based blockchain query, like Infura for Ethereum. A relay script server is also built here, to automatically trigger an event on periodically regarding block generation.
- GraphQL caching server: Since IPFS is still under testing, we currently use graphql caching server to improve user experience for sync the data, it also caching anonymous data, and improve data query speed.

- Validation and query server: validate the authorization tokens with HTTPS request for IoT devices or application.
- Client side SDK library: The javascript binding library will directly connect the front-end applications with Blockchain, for example, React or React Native applications.

## Privacy data feeding and protection

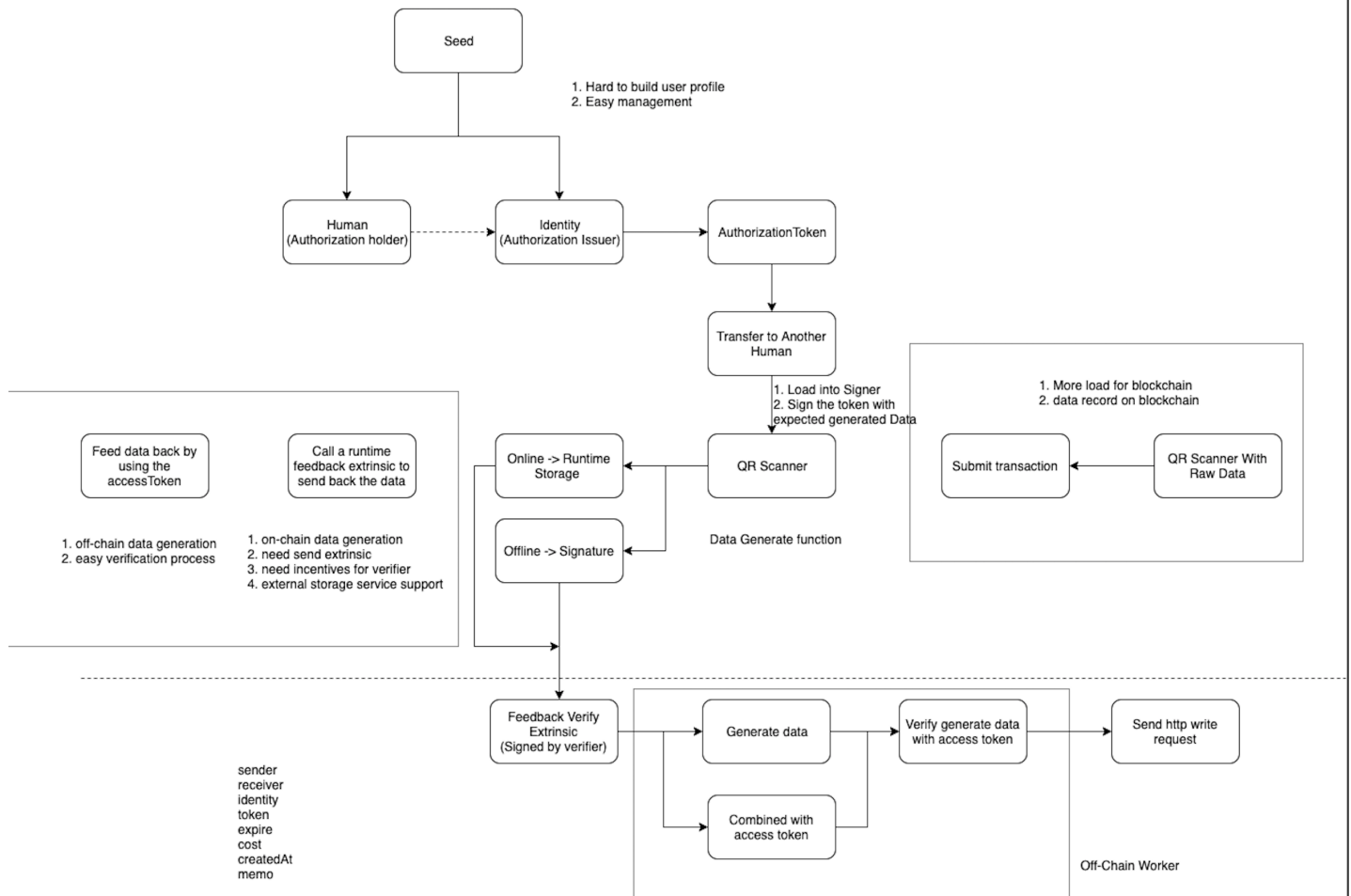
The current verification process has problems of privacy data generation.

1. If a user come to Hotel with his digital key (claim). The smart lock itself could generate data, and each time user come into the room, these pieces of data is generated, but they are harvested by the hotel, and never reach to user.
2. The Data generated are combined with user's accountID (public address), when these authorizations (claims) be more enough, it will be practical for the external companies to monitor the on-chain state, and build user profile.

To resolve this problem, and let the user get their data, we have designed our new method based on Substrate offline worker and crypto algorithms based on Schnorr25519.

1. To protect user's data, we migrate the data generation process on to Substrate, after the verification process, the crucial arguments like user address, verifier, time, token hash will be send into blockchain with a `verifier` extrinsic. Which is independent with off-line verification, thus it will not affect the verification time. Then the data will be generated in offchain worker, generate access token to user's storage, and then feed the data back to user.
2. To protect the user data, we will use HDKD feature to periodically generate new soft derivated key pairs for certain user. Verifier could prove the the address is belongs to the parent keypairs with crypto algorithms. Thus user will not need to always show one certain public address. Another key point is to use ring signature, which will hide user's signature behind bunch of people. Child address generation feature is already implemented in our crypto wallet and Ring signature is our current experiment direction.

The workflow of user data feed could be seen in the following diagram



Related Article about Off-Chain Worker: <https://www.parity.io/substrate-off-chain-workers-secure-and-efficient-computing-intensive-tasks/>

## Runtime

Github Repository: <https://github.com/litentry/litentry-node>

There are two pallets from litentry both account-linker and offchain-worker.

<https://github.com/litentry/litentry-node/blob/develop/pallets/account-linker/src/lib.rs>

<https://github.com/litentry/litentry-node/blob/develop/pallets/offchain-worker/src/lib.rs>

## Abstract

The identity runtime protocol link the all cross chain accounts to make an unique Litentry Identity. After accounts linked, Litentry user can trigger the asset claim via transaction. Litentry offchain worker will query your assets in other blockchain network, generate asset prove on-chain. Any Defi or other Dapp can use the information for their service:

1. Account Linking: Litentry user sign the specific data with private key.
2. Asset Claim: Litentry user ask runtime to query its asset and generate prove.
3. Defi or other Dapp: Use the Litentry ID and its assets prove for their service.

## Stakeholders

1. Litentry user: They register into Litentry network, can get the token incentive if their data queried by DeFi or other Dapps. Their information is protected with encryption and used in a secure runtime environment.
2. Defi: Litentry user's ID and their assets prove is critical factor to determine their financial service.
3. Dapps: Litentry user's cross chain information includes all activities in the whole cryptocurrency world. Litentry is the unique and unified entry point for their service.
4. Litentry node runner: Except get the block produce incentive, they can also run offchain worker for cross chain asset query service. They will be rewarded if the query result they submitted is correct.

## Scenario:

### Defi:

- Identity Registry: Litentry register and get Litentry ID with LIT token;
- Asset claim: If Litentry user also own assets in Bitcoin and Ethereum, they can link their account in Bitcoin and Ethereum at first. Then require Litentry to query their balance and create asset prove;
- Defi service: Litentry provide the SDK and API for Defi application, Defi can access Litentry ID and assets prove. Base on these information, Defi can put in into their algorithm model, then decide the loan amount, interests and credits score.

### Cross Chain Identity:

- Identity Registry: Litentry register and get Litentry ID with LIT token;
- Dapp: Dapp service can get the aggregated cross chain account information via Litentry Id. Then Dapp can avoid implement every API for different blockchain network;

Smart contract with credit score algorithm:

- Identity Registry: Litentry register and get Litentry ID with LIT token;
- Smart contract: Litentry runtime provide raw data to smart contract deployed in Litentry network. Different smart contract can use diverse algorithm to compute the credits score. The applications can use one smart contract's data or use data source from different smart contracts to get their customer's financial profile.

## In the future

The Litentry runtime is under very active development now. More pallets will be implemented and integrated into Litentry runtime. Litentry network will acquire the slot of both Kusama and Polkadot. Cross chain ID aggregation and query will be realized.

# Chain State Storage

## Identity Related

```
Identities: T::Hash => IdentityOf<T>;
IdentityOwner: T::Hash => Option<T::AccountId>;

IdentitiesCount: u64;
IdentitiesArray: u64 => T::Hash;
IdentitiesIndex: T::Hash => u64;

OwnedIdentitiesCount: T::AccountId => u64;
OwnedIdentitiesArray: (T::AccountId, u64) => T::Hash;
OwnedIdentitiesIndex: T::Hash => u64;
```

## Token Related

```

AuthorizedTokens: T::Hash => AuthorizedTokenOf<T>;
AuthorizedTokenOwner: T::Hash => Option<T::AccountId>;
AuthorizedTokenIdentity: T::Hash => Option<T::Hash>;

AuthorizedTokensCount: u64;
AuthorizedTokensArray: u64 => T::Hash;
AuthorizedTokensIndex: T::Hash => u64;

OwnedAuthorizedTokensCount: T::AccountId => u64;
OwnedAuthorizedTokensArray: (T::AccountId, u64) => T::Hash;
OwnedAuthorizedTokensIndex: T::Hash => u64;

IdentityAuthorizedTokensCount: T::Hash => u64;
IdentityAuthorizedTokensArray: (T::Hash, u64) => T::Hash;
IdentityAuthorizedTokensIndex: T::Hash => u64;

```

# Extrinsic

## issueToken

```
fn issueToken(to, identity_id, cost, data, datatype, expired)
```

Issue a token of an owned identity to certain account.

- `to` : Hash, the receiver identity id
- `identity_id` : Hash, the issuer identity id
- `cost` : Balance, the transfer cost of the token
- `data` : byte, The data stored in the token
- `data_type` : byte, the type of the data represent in byte
- `expired` : byte, which define the expired date of the token

## registerIdentity

```
fn registerIdentity()
```

Register a new identity for this account.

## registerIdentityWithId

```
fn registerIdentityWithId(identity_id)
```



Register a new Identity with existed identifier as Identity ID, this is useful when register a third party devices or services. For example, a user buy a new IoT camera, there could already exist a identifier on the back of device reserved for the buyer.

- `identity_id`: Hash, the identifier to be used as identity ID

## transferToken

```
fn transferToken(to, token_id)
```

Transfer a owned token to another account

- `to`: AccountId, the future owner of the token
- `token_id`: Hash, the ID of the transferred token

## Types

```
{
  "Address": "AccountId",
  "LookupSource": "AccountId",
  "IdentityOf": {
    "id": "Hash"
  },
  "AuthorizedTokenOf": {
    "id": "Hash",
    "cost": "Balance",
    "data": "u64",
    "datatype": "u64",
    "expired": "u64"
  }
}
```

When using with Polkadot.js App you will need to add the above Litentry's types so it can encode and decode the extrinsic correctly, copy the following code to the text field on "developer" tabs: <https://polkadot.js.org/apps//#/settings/developer> .

GeneralMetadataDeveloper

Additional types as a JSON file (or edit below) ⓘ  
Address, LookupSource, IdentityOf, AuthorizedTokenOf

```
1 {
2   "Address": "AccountId",
3   "LookupSource": "AccountId",
4   "IdentityOf": {
5     "id": "Hash"
6   },
7   "AuthorizedTokenOf": {
8     "id": "Hash",
9     "cost": "Balance",
10    "data": "u64",
11    "datatype": "u64",
12    "expired": "u64"
13  }
14 }
```

If you are a development team with at least a test network available, consider adding the types directly to the apps-config, allowing out of the box operation for your spec & chain-specific UI, however doing so does help in allowing users to easily discover and use with zero-config.

# Authentication Mobile App

Github Repository: <https://github.com/litentry/litentry-authenticator>

Download Litentry Authenticator v1.1.1 from [Android App Store](#)

## Mobile App Abstract

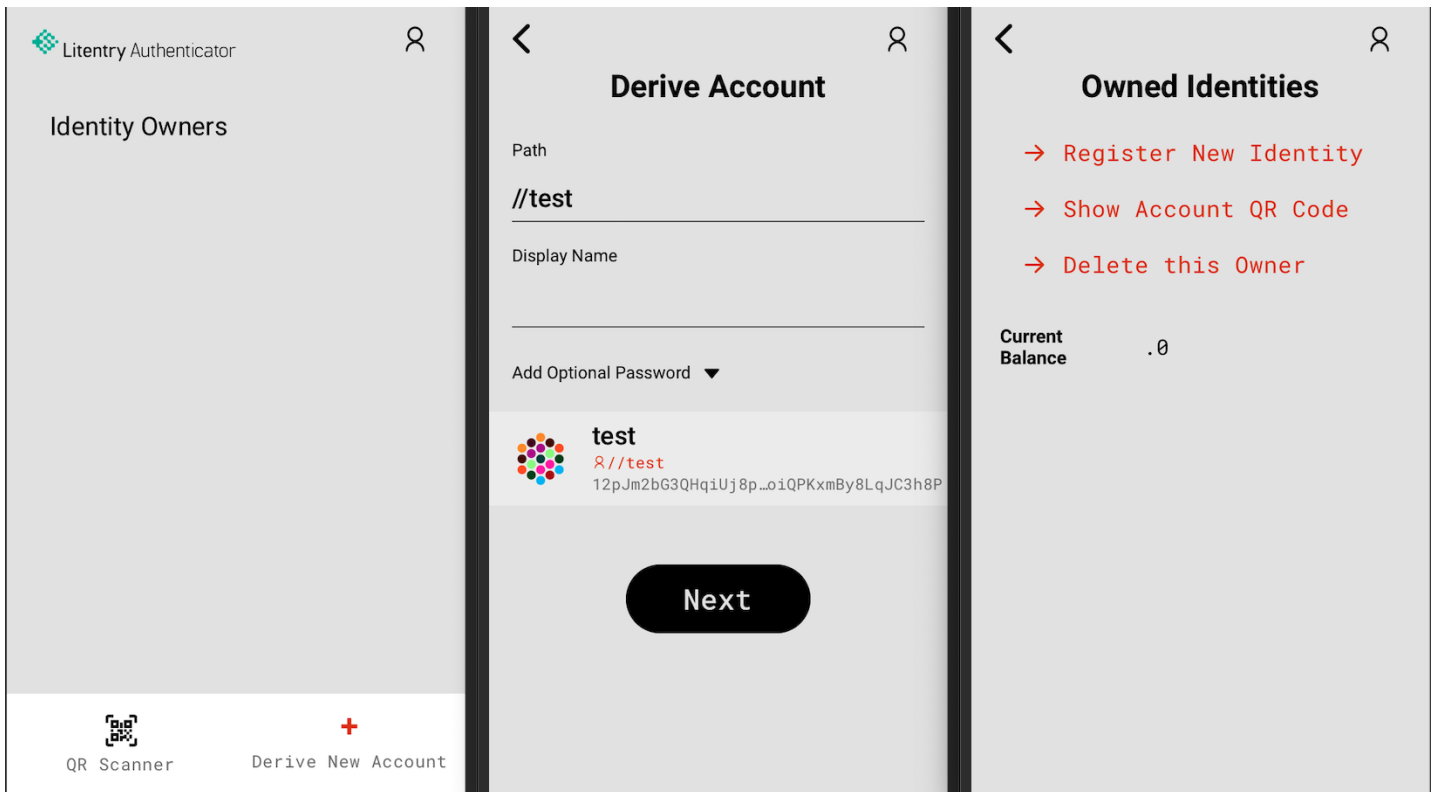
Personal users would like to use an Wallet Application to manage all its identities, it could also become a Hub connected to different interest IoT devices. For example, directly buying the authorization or the data from other IoT devices. With the advantage of GPS of mobile phone, it could further integrate with LBS (Location Based Services).

In order to work in a fully decentralized scenario, itself also need to integrate a light client, where could keep a user's private key in a secure environment provided by Android or iOS.

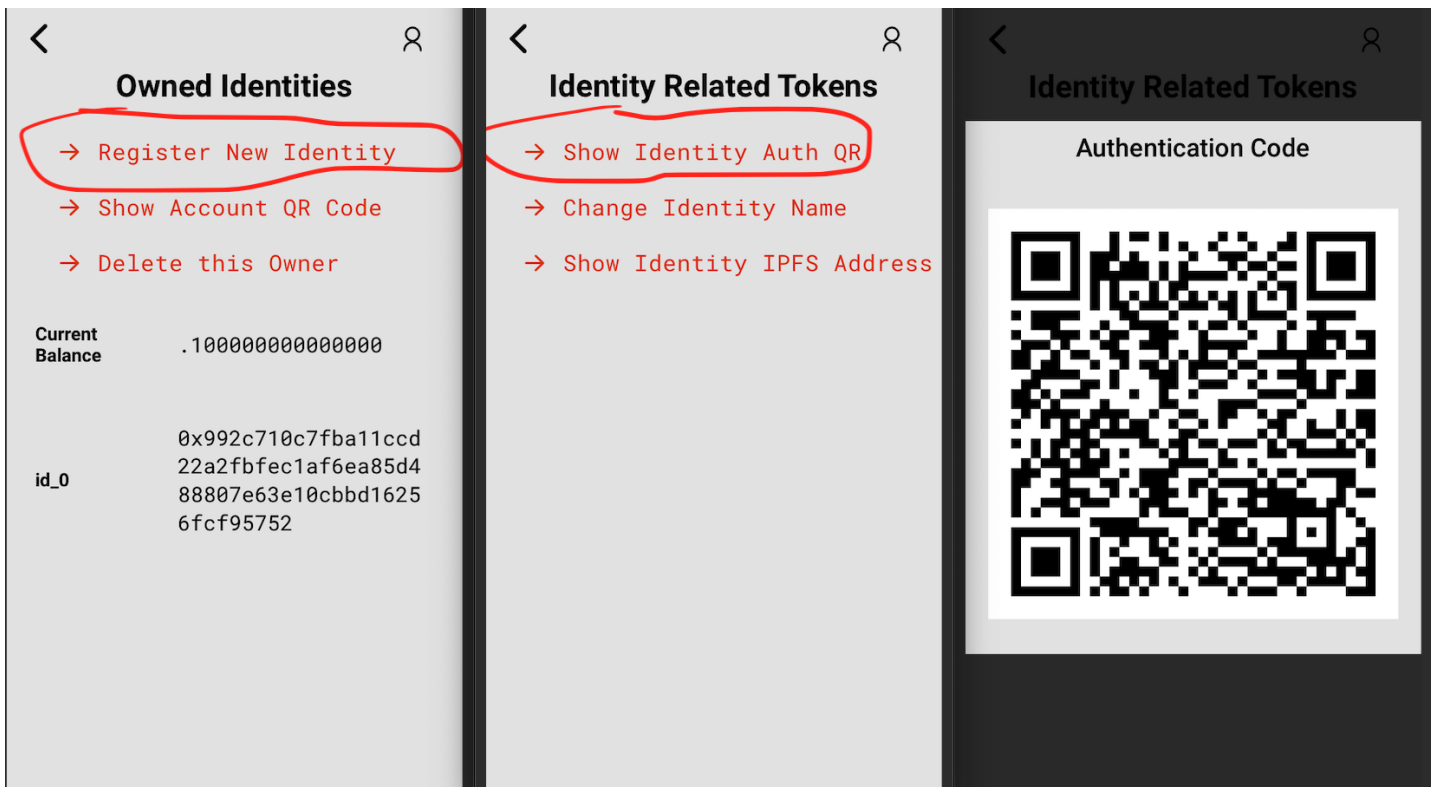
Mobile Application Screenshots:

For the step to step guide on use Litentry Authenticator please refer to [this article](#).

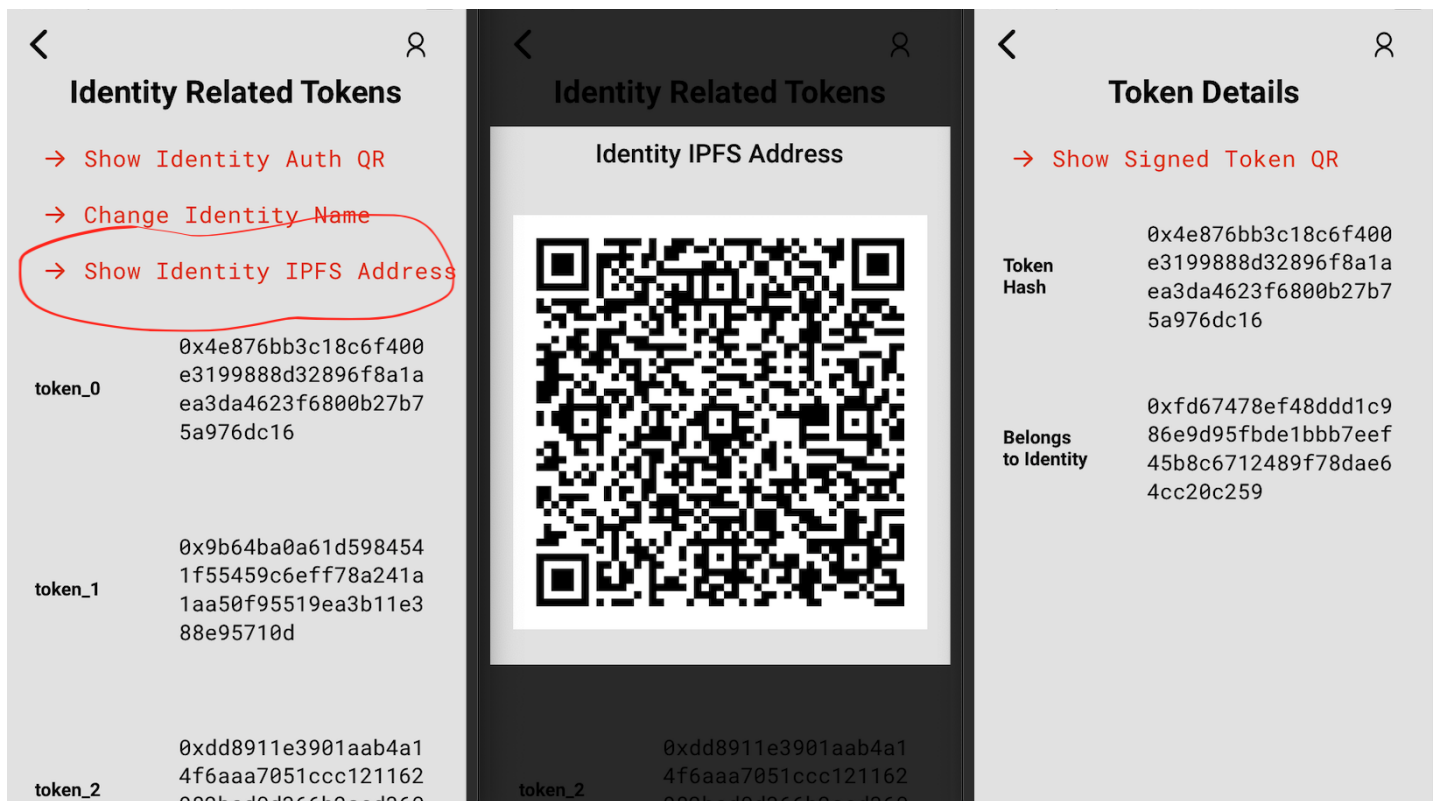
## Recover Seed and Manage Account



## Register Identity and 2FA Authentication



## Check Token Related Data



# Litentry SDK

Github Repository: <https://github.com/litentry/litentry-sdk>

This library provides useful functions to interact with the state on Litentry and user identity data related IPFS Storage.

It helps developer to build client side decentralized applications

## Getting Start

```
yarn add litentry-sdk
```

Import for start using it, SDK mainly includes three part, `hooks` , `query` and `ipfsApi`

```
import {hooks, query, ipfsApi} from 'litentry-sdk';
```

## React Hooks for Litentry

```
//Api loading State
hooks.useApi(): boolean

//Get Identities, use updatedIndex to force refresh
hooks.useIdentities(account: string, updateIndex: number): string[]

//Get Identity current owned tokens
hooks.useTokens(identityHash: string): string[]

//Get the owner of the token
hooks.useTokenOwner(tokenHash: string): string

//Get account balance of LTT
hooks.useBalance(account: string): string

//Help async function for query issuer Identity of the token
hooks.getTokenIdentity(tokenHash: string): Promise<string>

//Help async function for getting the last issued identity
hooks.getLastIdentity(account: string): Promise<string | void>

// react hooks for using native extrinsics on Litentry
hooks.useExtrinsics(): {
  registerIdentity: SubmittableExtrinsicFunction<'promise'>;
  issueToken: SubmittableExtrinsicFunction<'promise'>;
}
```

## Identity Data Query

Identity data are stored in the IPFS network and cached in Litentry GraphQL data server.

functions to query the latest data on IPFS:

```
ipfsApi.getAddress(identity: string): Promise<string | null>

ipfsApi.getData(identityId: string): Promise<string[]>

ipfsApi.registerIdentity(identity: string): void
```

functions to construct query http request from GraphQL:

```
query.getData(identity: string): string  
query.setData(identity: string, data: string): string  
query.method(methodName: string, identity: string): string
```

# Litentry DApp Playground

live site on <https://dapp.litentry.com/>

Repository: <https://github.com/litentry/litentry-web>

On how to start with Litentry DApp Playground please refer to [this article](#).

Litentry Playground is a hub of decentralized web app applications to achieve a Substrate based authentication.

User could use DAPPs with Litentry decentralized 2FA mobile App. No registration, no password, no App migration barriers.

More about the Authentication (Sign in Process):

## The Object of Litentry Authentication

- It should allow users to use his/her owned Substrate account related identity to login to a third party website (that supports this login method).
- It should be easy to use and reasonably easy to setup.
- It should not compromise the security of the user's Substrate account.
- It should allow users to recover their credentials in case of loss or theft.
- It should not require knowledge of Cryptography or Blockchain with authentication.
- It should have reasonable latency for a login system.
- It should not cost users gas (or money) to login.
- It should be reasonably easy for developers to implement in their apps.

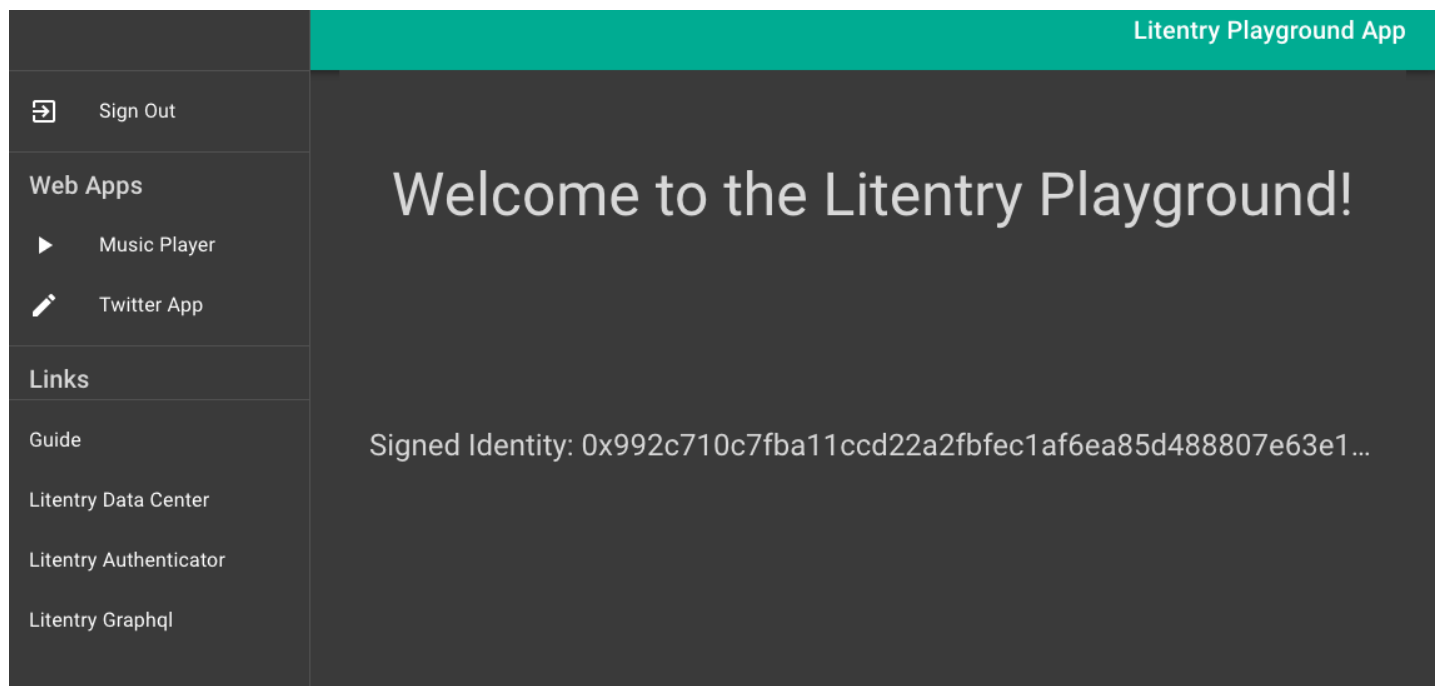
## The Implementation of Litentry Authentication

1. User on a third party website and click login with Litentry Button
2. A preset account will come if user has logged before in the computer, or a QR scanner comes out for user to scan his/her Substrate account QR code.
3. At this time the third party website send a transaction to Litentry network with a challenge string and its receiver server address. And sign a JWT with challenge and

server address embedded in it.

4. User now has to open the mobile app (best integrated with Substrate light client), it has watched the event of the auth request, and then sign the JWT with its private key and then send the signed double signed JWT to the server address.
5. The server proved the token received by the user and then finish the logging process.
6. After the usage of the third party web application or service, third party allocated the user browsing data/history and query the user's data resolver address, then user data is send back to the resolver and being process and harvested into user's own database.

## Sign in to DApp Playground



## Star Songs in dSpotify App

Sign Out

Web Apps

▶ Music Player

✎ Twitter App

Links

Guide

Litentry Data Center

Litentry Authenticator

Litentry GraphQL

# dSpotify Music Player

All the play and star record will be record, an click generate data token on blockchain (check Litentry Authenticator), and data record on IPFS (check data center)

Pop Music

▶ Dynamite  
BTS ☆

▶ Watermelon Sugar  
Harry Styles ☆

▶ Savage Love  
Jawsh 685 x Jason Derulo ☆

▶ Midnight Sky  
Miley Cyrus ☆

▶ Be Like That  
Kane Brown, Swae Lee, Khalid ☆

▶ Rain On Me  
Lady Gaga & Ariana Grande ☆

Country Music

▶ New Normal  
Cooper Alan ☆

▶ Starting Over  
Chris Stapleton ☆

▶ Got What I Got  
Jason Aldean ☆

▶ I Hope (feat. Charlie Puth)  
Gabby Barrett ☆

▶ One of Them Girls  
Lee Brice ☆

▶ Happy Anywhere  
Blake Shelton ☆

## Record mood with dTwitter App

Sign Out

Web Apps

▶ Music Player

✎ Twitter App

Links

Guide

Litentry Data Center

Litentry Authenticator

Litentry GraphQL

# dTwitter

each twitter will generate data token on blockchain (check Litentry Authenticator), and twitter content is recorded on IPFS (check data center)

Write something down!

Here we go

PUBLISH THIS!

# Litentry IPFS Data Center


Github Repository: <https://github.com/litentry/litentry-ipfs-data-center>

Live Site on <https://data.litentry.com/>



Litentry uses OrbitDB to offer an IPFS database support. In Data Center, user may check their identity related data and tokens.

In the future we will implement Arweave and on-Chain key value storage.

 LITENTRY DATA CENTER

Q Search...

Systems: IPFS OrbitDB

← DATABASE

/orbitdb/zdpuB1L2Br5RYh9V4mQyWg3mvaenBVxp8J9ERbuo3h7Qq9Udp/0x992c710c7fba1ccd22a2fbfec1af6ea85d488807e63e1c

Name: 0x992c710c7fba1ccd22a2fbfec1af6ea85d488807e63e10cbbd16256fcf95752

Type: eventlog

Permissions:

\*

Entries: 4

Latest 10 events

{ "playgroundRecord": "write:post on IPFS" }

{ "playgroundRecord": "star:Watermelon Sugar" }

{ "playgroundRecord": "star:Dynamite" }

{ "playgroundRecord": "identity Created" }

Add an event to the log

Value

+ Add

# Middleware

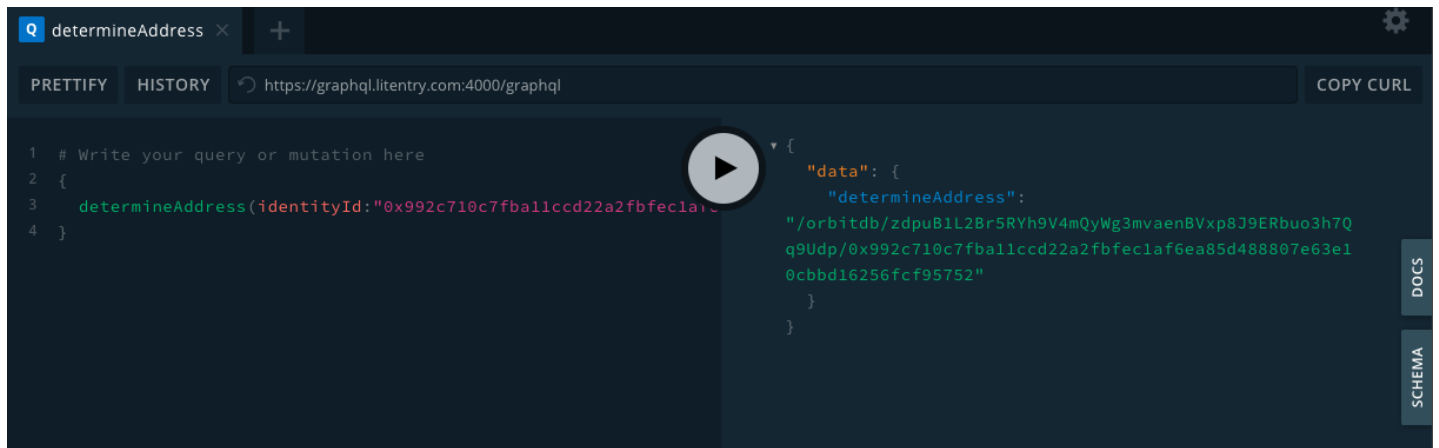
API of data server

Live Server: <https://graphql.litentry.com:4000/playground>

Github Repository: <https://github.com/litentry/litentry-ipfs-graphql>

Currently, we provide a GraphQL caching server for recording event on the Litentry blockchain and caching data from IPFS.

IPFS is still under testing, graphql caching server could improve user experience for sync the data, it also caching anonymous data, and improve data query speed.



## Query types

```
type Record {
  ${recordKey}: String
}

type Query {
  registerIdentity(identityId: String): String
  determineAddress(identityId: String): String
  addData(identityId: String, data: String): String
  addDataAddress(address: String, data: String): String
  getData(identityId: String): [Record]
}
```

Example for query `playgroundRecord` data of certain identity

```
https://graphql.litentry.com:4000/graphql?query=
{getData(identityId:"0x992c710c7fba1ccd22a2fbfec1af6ea85d488807e63e10cbbd16256fcf95752")
{playgroundRecord}}
```

query IPFS address of certain identity

```
https://graphql.litentry.com:4000/graphql?query=
{determineAddress(identityId:"0x992c710c7fba1ccd22a2fbfec1af6ea85d488807e63e10cbbd16256fcf95752")}
```

# Light Client Services

## Concept

Light Client should support all the networks on Substrate, and each of them have a node, so we should not pre-package all the nodes into the App, but support the app to download the most recent binary files and save them into the app. It should have at least “choose network”, “start service”, and “stop service” button. And most of the time it should be running in the background and work as an interface for all the apps to interact with a certain network.

In addition to giving apps a decentralized feature that talk to the network. An app could register an event subscribing on the light client service and push notifications to the user. So that the notification does not need to go through Android’s or Apple’s message center. A challenge would be how to always keep the light client app alive, and when the user powered off the phone, the light client would stop receiving messages (a breakpoint resume function could be implemented for the user to get events from the last fetched block).

## Android Implementation

The binary could successfully be compiled with the guide here and run on Android Devices.

To keep the light client always living in the background, we need to use Service, but the policy changed from time to time.

a minimal implementation: <https://github.com/hanwencheng/LightClientServices>, which supports users to download the binary of Flaming Fir, and run the service in the background.

## FAQ

---

What is the common use case of the identity protocol?

---

In our protocol, we basically analog most authorization scenarios into a common protocol with four elements: d1. Person or IoT devices Identity d2. Authorization d3. Member's who holds the Authorization d4. External data related to the Identity. Here the Authorization token is similar to the ERC721 standard but not the same, it could be used to help understand the idea. The common idea is that the authorization is a piece of data created by the d1. Person or IoT devices Identity and send to a d3. Member. The Specification of each d1. Person or IoT devices Identity will be saved as d4. External data. This ownership and specification could be proved by the whole network.

Take door lock for example: An Airbnb host could use a smart lock to issue the digital key on our protocol, the user who has Parity-Signer Wallet (for example) could fetch the token data (d2. Authorization) and send it to the lock by a p2p way (Bluetooth for example) for validation.

The lock will interpret the token data with its own pre-defined interpreter function (on-chain or off-chain, still need to think), like how many time a user may enter, what is the entry duration. By this way, the d1 Person or IoT devices could be shared in a decentralized and a securer way. We suppose such kind of authorization would fit the normal business (hotel) and sharing economy very well.

---

Why use non-fungible-token as key rather than a signed message, e.g. a message signed using the digital key and can be validated by the lock off-chain.

---

Signing transaction way works in authorization and it is simple. The basic of token authorization model is to validate a piece of information which is sent by the owner. These pieces of messages could be encrypted in the transaction as a message memo, and even transfer to others with last message data include or using a UTXO model.

In comparison, The "non-fungible-token way" or "ERC721 Standard way" gives a state map of all the authorizations and identity of the IoT device or people. The advantages are:

1. The relationship between d1. Person or IoT devices Identity d2. Authorization d3. Member's who holds the Authorization could be checked easily, and this is very useful in many scenarios. For example, in an apartment sharing business, the user could know how many keys (tokens) are existed for a certain lock at the moment and further check the ownership of all other keys (tokens).
2. Bulk transactions are more efficient and easy to update (even no need for loops). For example, an identity could easily recall all his authorized tokens, or updates existed tokens information.
3. A state explicitly shows the relationship and will be easy for understanding, and cross-chain programming. The further cross-chain function call will be based on the state information on different parachains.

In addition, the off-chain validation is still possible since the proof of token could directly be the identity (hash) of the token, and its information is retrievable from the state map. User with this information could send it to an off-line validator. ( Or a compromise way is that a validator regularly syncs with blockchain for the tokens and owners of identity, or totally an online validator).

---

why do we need a GraphQL server? Is it worth the trade-off (complexity, centralization) as opposed to requests being validated directly on chain?

---

A GraphQL is basically an API server, we have two different views of its usage.

1. In a short view, since currently there are not so many libraries like oo7 which can directly connect with Substrate Runtime and external IPFS storage. An API server will currently offer fast development and could be used as a good demo for application developers, it could be regarded as Infura on Ethereum.
2. In a long-run view, we supposed it could be a query server for historical and caching server, which could offer fast access for the historical data, good infographics from database, caching the data by event listeners and queries, and reduce the load of the blockchain. Though it is with tradeoffs and comprises to the decentralized architecture, the most important validation and issuing functions will still be accessible directly by Blockchain.

So in summary, the GraphQL API server is a necessary-to-have thing, but whether to use it will be mostly chosen by the application developer.

---

What is the difference between ID Chain and Lock Chain and the need for each?

---

Litentry is aimed to offer a basic protocol and related framework, which could be used for fast application development. Both the ID chain and Lock chain are one application based on it.

The traits of these two chains are in the same protocol with slight differences. Take ID chain, for example, The registry could be the citizen offices in a different city. Permissions would be that the different Right bind with certain people, like (boolean value for over 18 age, int value for tax level, etc). Permissions could be assigned to only one Member, but a member could have more than one Permissions.

The same in lock chain is that the Locks are a list with all different kinds of lock, the lock owner who has the private key of the lock may issue the access right token to the member, like multiple entry token, or a one-day-pass token. A member could have a different token with different locks, but one token could be only assigned to one person.

Though there are similarity and difference based on the implementation of the protocol of different traits of parachain. The main difference between these parachains is that they have different cross-chain function based on that. The cross-chain function could be e.g. A insurance (which authorized insurance token) calculate the healthy insurance working permit of ID Chain; the Lock chain will issue the entry token of a shisha bar with validating the age on ID chain; A gym will offer a special membership token to the people who have the entry token of co-working space, etc. All these cross-chain issue and validation will be defined by the cross-chain function on different para chains with the same protocol.

In addition to that, separate these chains will gain the flexibility to different business scenarios by adding the parachain-scope function. For example, an IoT device Chain with temperature monitors of different locations. A value of all the registered temperature monitors on 24:00 every day could be harvest directly to external storage. But if a user wants to get a piece of detail information on a certain time, he needs to has an authorization token for querying that data.