Saito Dot Integration Details

Introduction

In order to facilitate integration with DOT and other Substrate-based tokens within the Polkadot ecosystem, Saito will be developing a prototype of an Open Infrastructure-type Decentralized Service which is enabled by Saito's ability to serve as an Open PKI Infrastructure and Transport Layer.

The Saito Lite Client facilitates interoperability with any cryptocurrency by implementing a Module which provides a respondTo with an "is_cryptocurrency" type interface(see Applications). By implementing the is_cryptocurrency interface a DAPP author can enable their Module to be selected as the user's Preferred Cryptocurrency within the Lite Client and allow other DAPPs, such as games, to interact with it's cryptocurrency through the is_cryptocurrency interface.

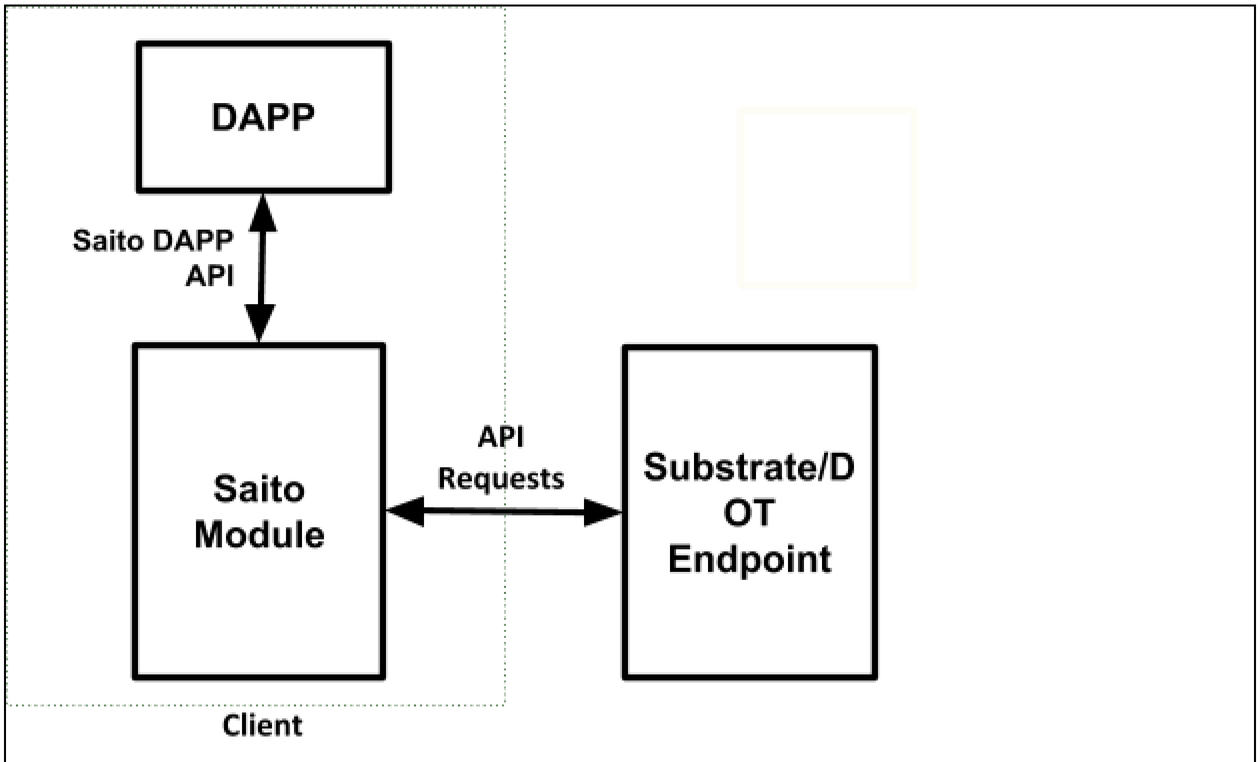
The functions provided by the Lite Client's Wallet API will then honor the user's Preferred Cryptocurrency, allowing a DAPP to seamlessly change from one cryptocurrency to another.

If a DAPP author also wishes to leverage Saito as an Open Infrastructure platform as well, all interactions with the cryptocurrency can be passed through Saito Transactions. This enables efficient distribution of the transactions to their endpoint services within the Saito Network and also allows the author to receive payment for access to the endpoints by requiring a micro-payment in Saito.

The first milestone to support this type of Service infrastructure is an API to support interoperability between Saito and the 3rd party cryptocurrencies. This is equivalent to the grant we have been awarded by the Web 3 Foundation, which is in a prototypal stage now.

If there is further interest in leveraging this system we also plan to extend the APIs further to better support an Open Infrastructure Service. For now, that work is left to the DAPP author and we encourage anyone interested in this to reach out.

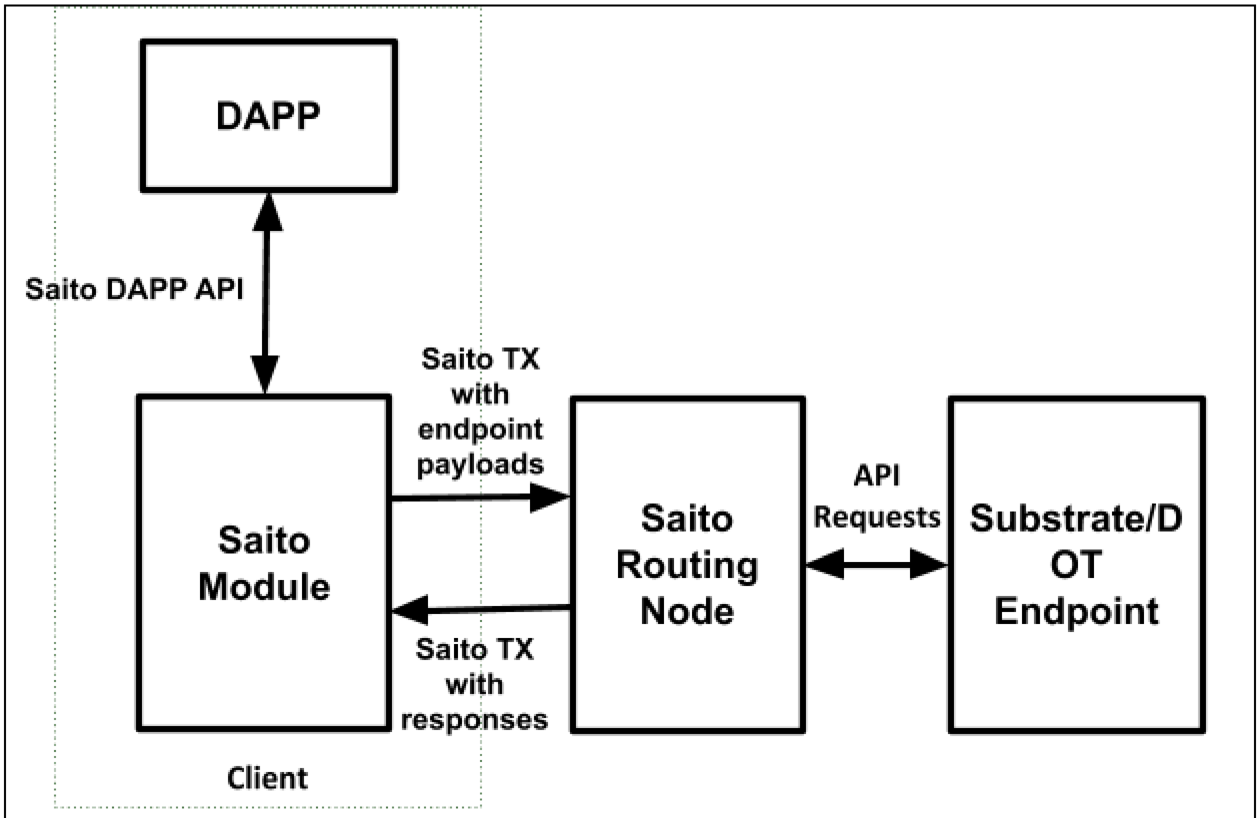
Architecture



Using Saito DAPP API to integrate directly with an open/free API

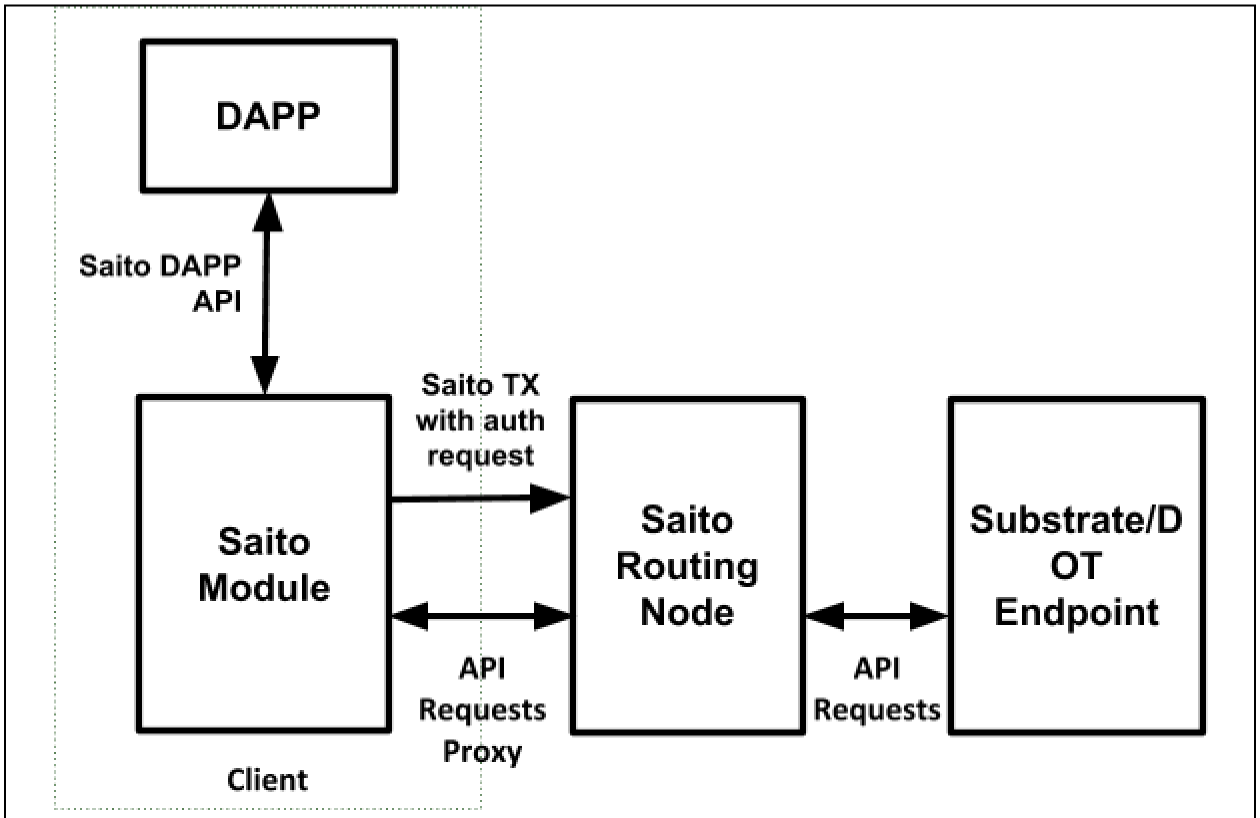
A DAPP author can integrate with the backend directly. In this case the module essentially acts as a standard wallet which can be upgraded later to support an Open Infrastructure via Saito Open PKI and Data Transport Infrastructure.

The DAPP API can be upgraded later to support another backend(like below) without the need to change other aspect of the frontend(for example keypair management). This can allow both DAPP developers or users to choose their backend.



Using Saito DAPP API to integrate with a Saito Open Infrastructure

In this case the wallet will route transactions through a Saito endpoint and require payment/authorization on a per-request basis.



Using Saito DAPP API to integrate with a Saito Open Infrastructure with time-based authorization

If the DAPP author wishes to allow users to access the endpoint with time-based authorization, this would allow API requests to be proxied directly, allowing things like websockets/push-notifications.

Future Work

The project is a work-in-progress and subject to change, but a general idea of the direction can be seen from the current proposed milestones, which will most certainly change in their details but not in general functionality.

Current Prototypal APIs(Milestone 1)

The current API is implemented through a type-free system in our Modules API called respondTo. Modules that wish to appear as 3rd party cryptocurrencies within the Saito Ecosystem can simply implement the appropriate respondTo interface(which we call is_cryptocurrency) and their Module will be interoperable with Saito and Saito's libraries.

Example:

```
respondTo(type = "") {
  if (type == "is_cryptocurrency") {
    return {
      name: this.name,
      description: this.description,
      info: this.info,
      ticker: this.ticker,
      getBalance: this.getBal.bind(this),
      transfer: this.transfer.bind(this),
      getAddress: this.getAddress.bind(this),
      estimateFee: this.estimateFee.bind(this),
      setConfirmationThreshold: this.setConfirmationThreshold.bind(this)
    }
  }
  return null;
}
```

Future Grant Work(Milestone 2)

SaitoAbstractWalletProperties:

hasKeys:

Boolean

Indicates whether the wallet has been initialized and is ready to use

Public Methods:

Sign(rawtx):

Signs a raw object_

GetKeypair/Keyring():

Get the keypair or keyring directly so that other modules might be able sign/send transactions. This is useful for cases where the endpoint may be very simple and just wants to send transactions directly from the keyring without having to build a signable payload.

LoadKeys(keys):

Loads keys from an external source, for example a key generated offline or wallet backup

GenerateKeys():

Generate keys for the user and store them.

GetEncryptedKeyBackup(passphrase):

Return an encrypted object holding the user's private keypair.

SetEndpoint(SaitoAbstractEndpoint):

Sets the endpoint which will be used for all calls to the backend endpoints.

Call(methodName, payload, callback):

Routes a call to the wallets endpoint.

SaitoAbstractEndpoint

Call(methodName, callback, payload):

Returns a promise which will resolve with the response.(i.e. is an async function).

Wraps a request to an endpoint. This can simply hit an endpoint directly, but will often wrap the request into a Saito Transaction to leverage Saito Open Infrastructure.

For example, a Saito Endpoint might implement calls like these:

```
Call("GetBalance", pubkey, callback)
Call("BuildSignable", {rawdata: ...}, callback)
Call("BuildSignableBasicTx", {to: …, amount: ...}, callback)
Call("BroadcastTransaction", {rawtx: …}, callback);
```

The Callable method names are at the discretion of the Saito Endpoint author and can wrap multiple calls to an endpoint. For example, to build a basic transfer transaction a Saito Endpoint Module may need to get both the balance and nonce for the sender, which may represent multiple calls to the actual endpoint.

An ETH-based SaitoEndpoint wrapping ETH might something like this: Call("eth_sendTransaction",...") And Call("eth_getBalance", …) for consistency with the ETH ecosystem.

CallStream(methodName, payload):

Returns an Stream Object(https://www.npmjs.com/package/stream, https://nodejs.org/api/stream.html)

Allows the user to keep a stream open the the Saito Endpoint for as long as it is authorized. The endpoint should close the connection automatically when authorization has ended.

Typically wraps a Websocket connection to the endpoint.

Note:

If a Saito Endpoint author wishes to grant unlimited access to typical HTTP/REST endpoints for a given period of time it is recommended to create a call like Call("RequestAuthorization",...) and to add a signed challenge object to subsequent calls to the HTTP endpoints.

For example, if a user has been granted 24 hours of access to a call to something like OnBalanceChange, this could be authorized via Call("RequestAuthorization",{pubkey: myPubkey}). Once authorized, subsequent calls to the endpoint would sign dated challenge tokens from the endpoint in order to gain HTTP/REST-style access to those endpoints.