Riddle & Code
LIBRDDL SDK

Table of Contents

Table of Figures

1 DESCRIPTION

This document includes usage method, software architecture and test results of RDDL SDK.
< What is RDDL SDK? Purpose of RDDL SDK>
< Mention libRDDL repo >
< Mention Planetmint and Liquid>

2 SOFTWARE ARCHITECTURE

Figure 1 indicates library structure. The library consists of 3 layers:

- libRddlApi: This layer includes user functions.
- libRddlAbst: This layer includes hardware specific functions
- libRddl: This layer includes common software function which are platform independent.

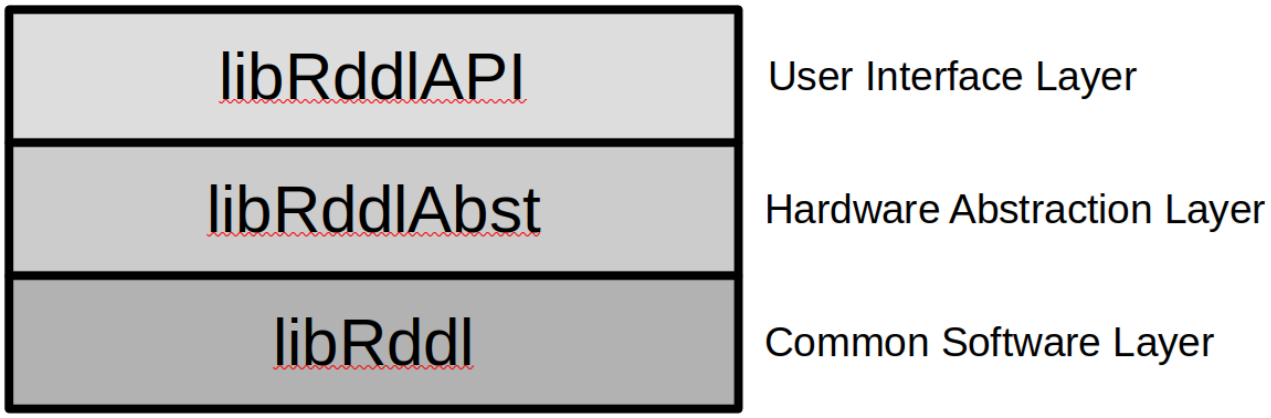

Figure 1: General System Overview

3.1 API FUNCTIONS

User can call libRddlAPI functions to use the library:

void runRDDLNotarizationWorkflow(const char* data_str, size_t data_length): This function has two separate tasks. If the device is registered to the network, it creates a transaction using the given data and broadcasts it. If the device is not registered to the network, it creates a registration transaction and broadcasts it. Figure 2 gives flowchart of this function.

The getPlntmntKeys function prepares public and private keys to be used in other functions. It generates the master key from the seed stored in the device. It generates planetmint keys from masterkey with the following derivation path /44'/8680'/0'/0/0 and liquid keys from /44'/1776'/0'/0/0. It also derives the public key from the private key embedded in the machine.

The hasMachineBeenAttested function checks whether the device has been registered before by querying *https://testnet-api.rddl.io/planetmint/machine/get_machine_by_public_key*. It uses the planetmint external public key for the query which is serialized of planetmint node with PLANETMINT_PMP version.

The registerMachine function creates a message in the google protobuf structure containing machine information. This machine information includes the generated public keys, the machine's domain and device information.

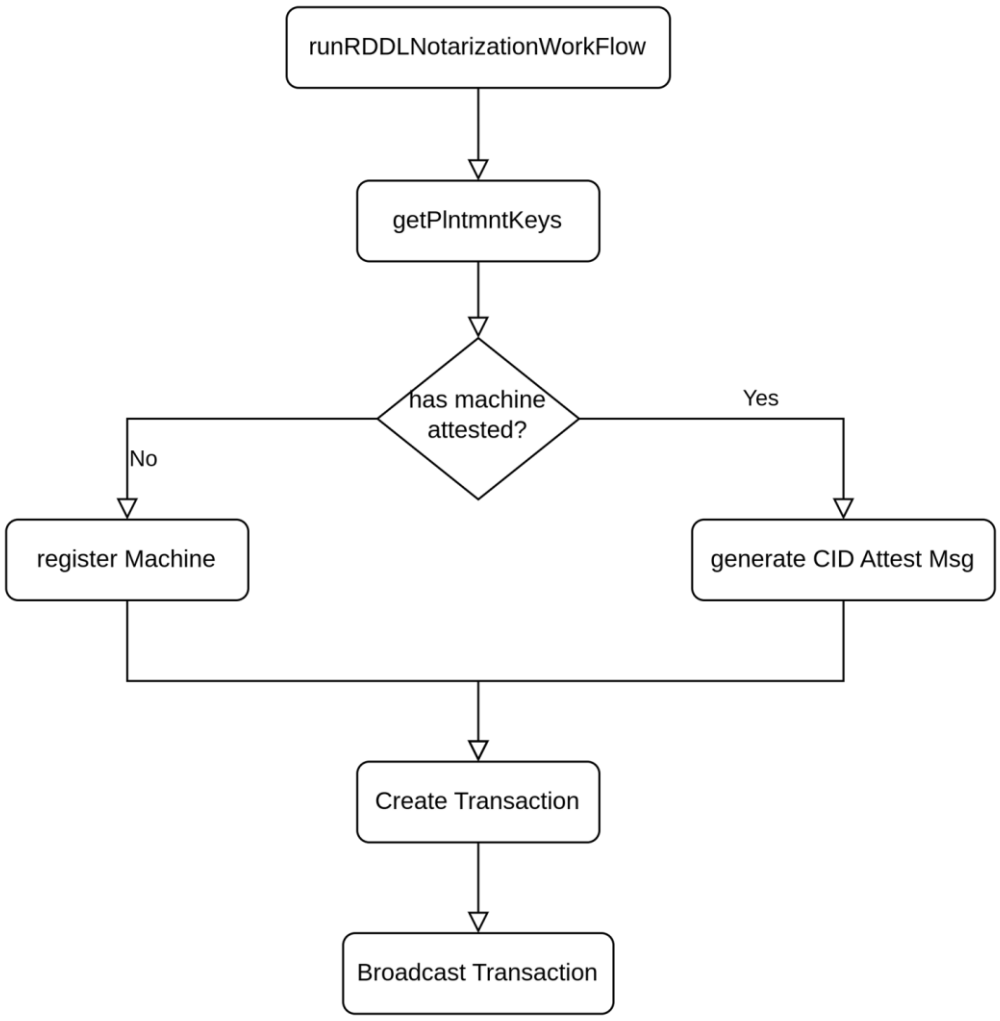

Figure 2: flowchart of runRDLLNotarizationWorkflow function

The create transaction function prepares the transaction using the generated message in the google protobuf structure, planetmint keys, denom, sequence, chain id and account id. Denom and chain-id information are retrieved from the device. If it is not saved in the device, default values are used which are plmnt and planetmint-testnet-1 respectively. Sequence and account id information is retrieved by querying *https://testnet-api.rddl.io/cosmos/auth/v1beta1/account_info/*. The hash160 version of the planetmint public key is used for this query. If the query returns negative, information is retrieved from the device. If it is not saved in the device, default values are used which are 0.

Finally, broadcast transaction function sends the created transaction to *https://testnet-api.rddl.io/cosmos/tx/v1beta1/txs* address.

char* sdkSetSeed (**char*** pMnemonic, **size_t** len): Generates seed for the device from the given mnemonic. If a mnemonic is not given, it generates itself. Returns the used mnemonic.

void sdkStoreSeed (**char*** new_seed): Writes the seed to the non-volatile memory of the device. If new_seed is NULL, it writes the seed produced by sdkSetSeed function. If it is not NULL, it writes the value in new_seed.

void sdkReadSeed (**char*** seed_arr, **int*** seed_size): Returns embedded seed value and it's size. The size of seed is currently 64 bytes.

3 TESTS

Compile library with following command on the project directory. It will also generate test executable.

bash build.sh

To run test code, use following command on the project directory:

bash runTest.sh

If the test device is not registered to the network, the following errors will be received when the test program is run.

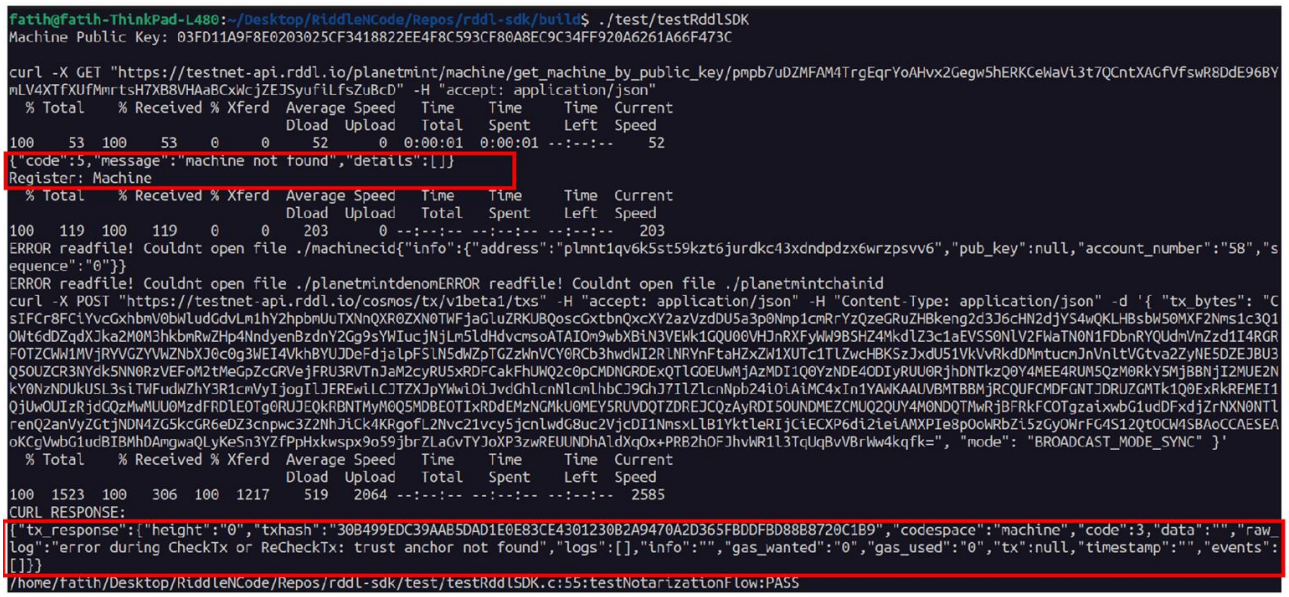

Figure 3: Message for Trusted Anchor that has not registered the network

The first message in the red frame indicates that the seed of device is not registered to the network. Second one indicates that the Trusted Anchor used is also not known by the network. Note that Trust Anchor's private key is embedded in the code. If it will be used as a different device, it must be changed. It is called private_key_machine_id under rddl.c file.

First, Trust Anchor needs to be registered to the network. To register the Trusted Anchor, we must send a request to the network using the 32-digit Hex number specified by the “Machine Public Key” in Figure 3. This number is public key of Trusted Anchor. To send registration request, we need to use the following message;

curl -X POST https://testnet-ta.rddl.io/register/<32DigitHexMachinePubKey>

```
fatih@fatih-ThinkPad-L480:~/Desktop/RiddleNCode/Repos/rddl-sdk/build$ curl -X POST https://testnet-ta.rddl.io/register/303F98F8EE26BB54384D1627E3A5E2
38D8031838183786A22E877C298A736FEA
```

Figure 4: Registration Request for Trusted Anchor

After registering the machine to the network, we will load some coins on it so that it can pay the transaction fees. Otherwise, transactions will be rejected due to insufficient coins.

To do this, open https://testnet-faucet.rddl.io/#/default/post_ . Then click POST and Try it out, respectively and upload money to the account ID. The Address is the value specified with “address:” and starting with plmnt as in Figure 3. After entering the address as shown in Figure 5 and writing down how much you will send to the coin section, you can click Execute and load the coin.

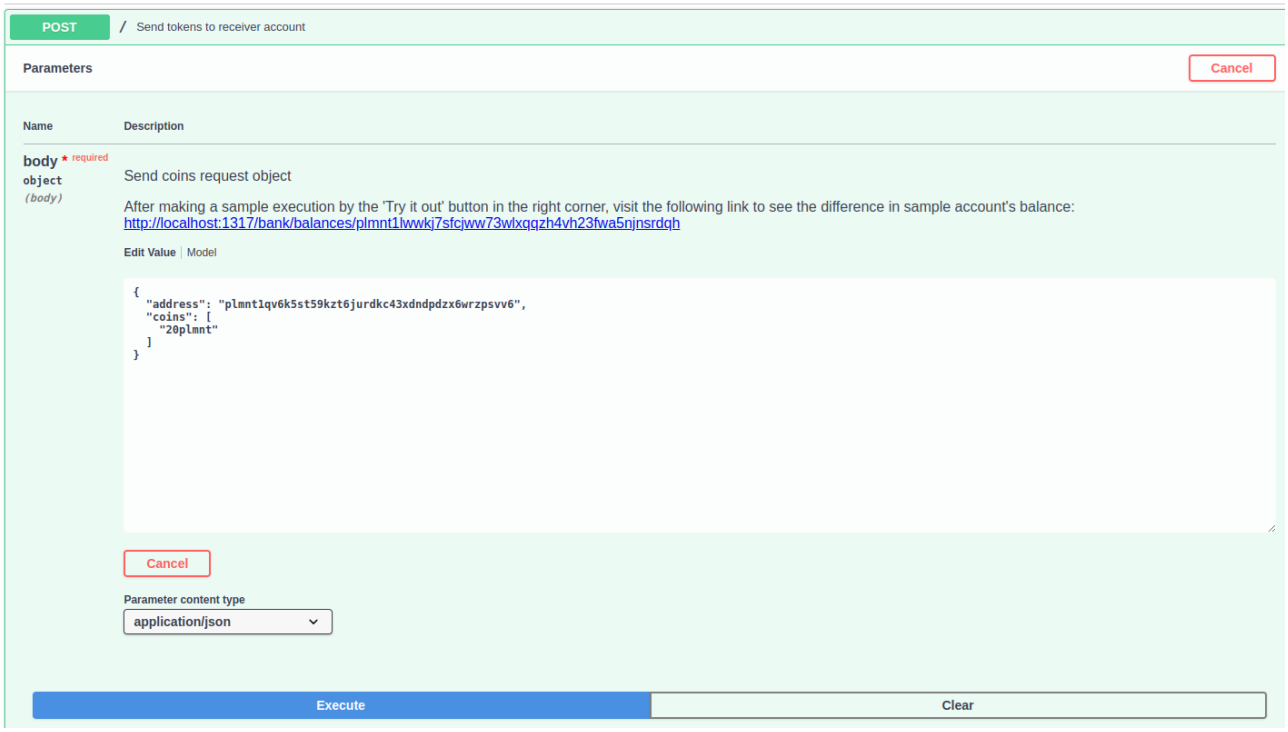

Figure 5: Sending coin to the device

Now, when we run the same test code again, it will first perform the registration process as in Figure 6.

Figure 6: Registration of Seed

After the seed is registered, every time we call the test code, it will perform a transaction that contains the data as in Figure 7.

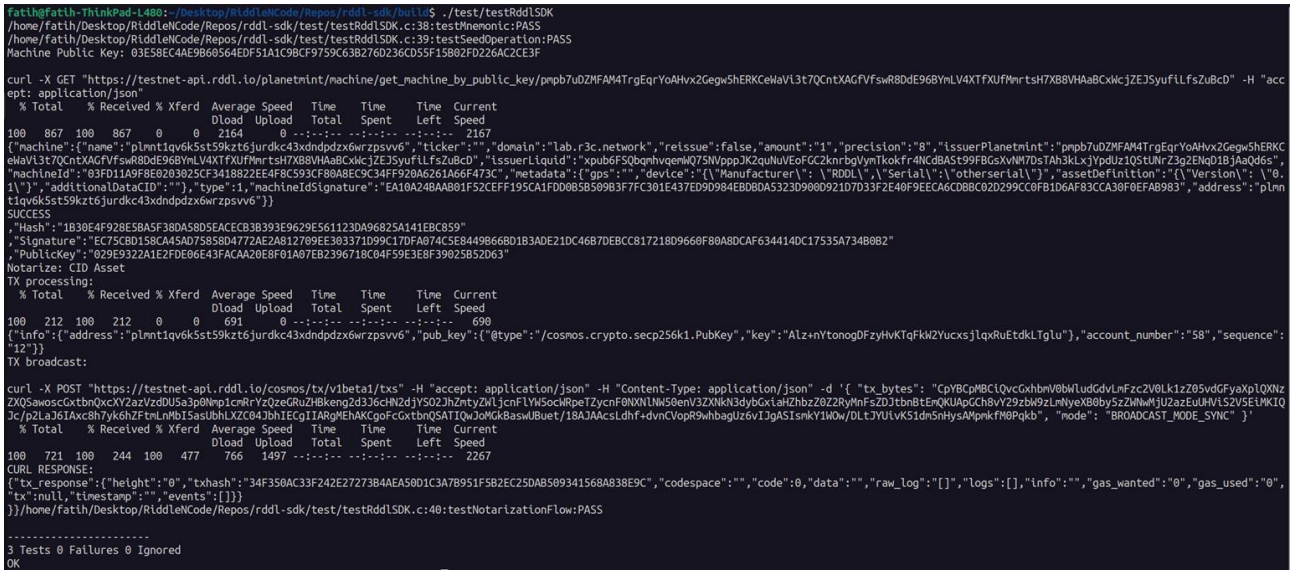

Figure 7: Data Transaction into RDDL Network