FIDO2 Vault

Overview

FIDO2 Vault is an online data store for user credentials that can be used for authorizing keys to their Algorand blockchain account.

- Users can register and add multiple credentials from any FIDO2 supported device
- Users can create recovery codes based on ED25519 cryptography
- Users then create an Algorand account as a contract account (Logsig) based on the credentials and recovery codes they have
- Users can interact with a dApp through a WebAuthN client (standard in browsers) to authorize transactions

Use-Cases

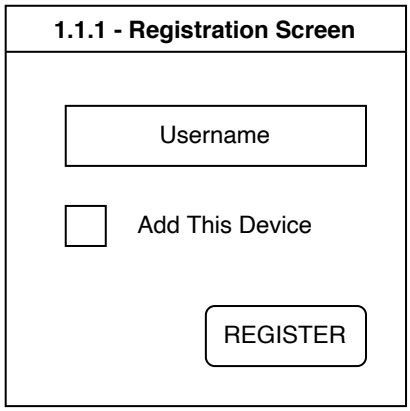
The vault is an online web-app with the following main components:

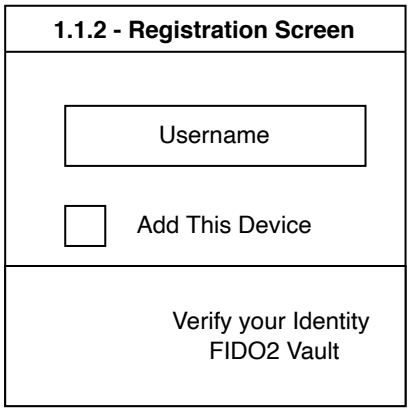
1.0 - user registration and authorization

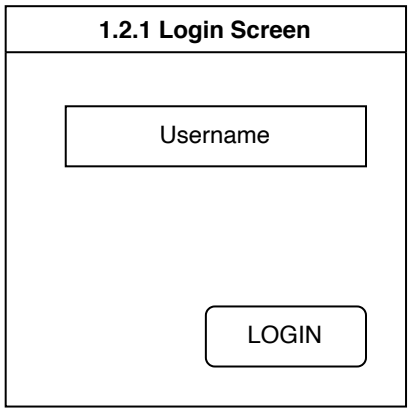
2.0 - user account management

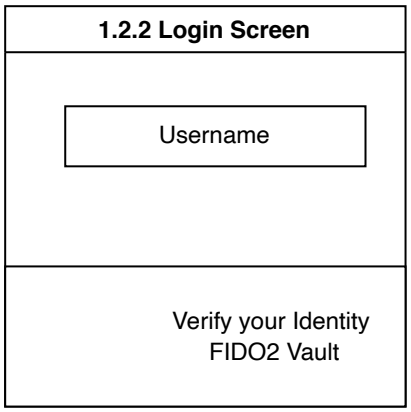
3.0 - user transaction confirmation

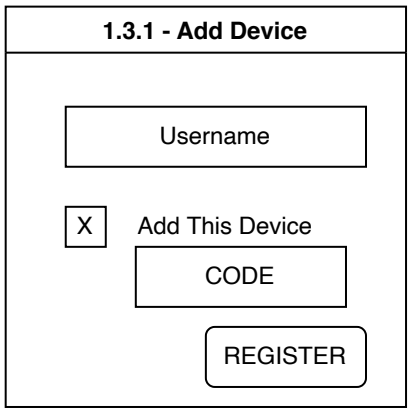
1.0 User registration and authorization component

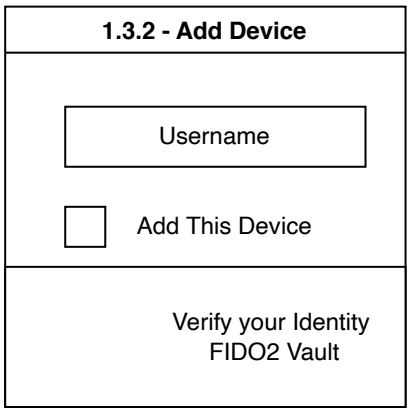












There are three feature-sets:

1.1 - User registration

1.1.1 - User is prompted to enter a username in order to register

1.1.2 - User is then prompted by their device to register with FIDO2 using their biometric (or pin/password)

1.2 - User authentication

Once registration is complete, a user can sign back in to the vault at any time using their FIDO2 device

1.2.1 - User is prompted to enter a username

1.2.2 - User is prompted by their device to authenticate with FIDO2

1.3 - Add device

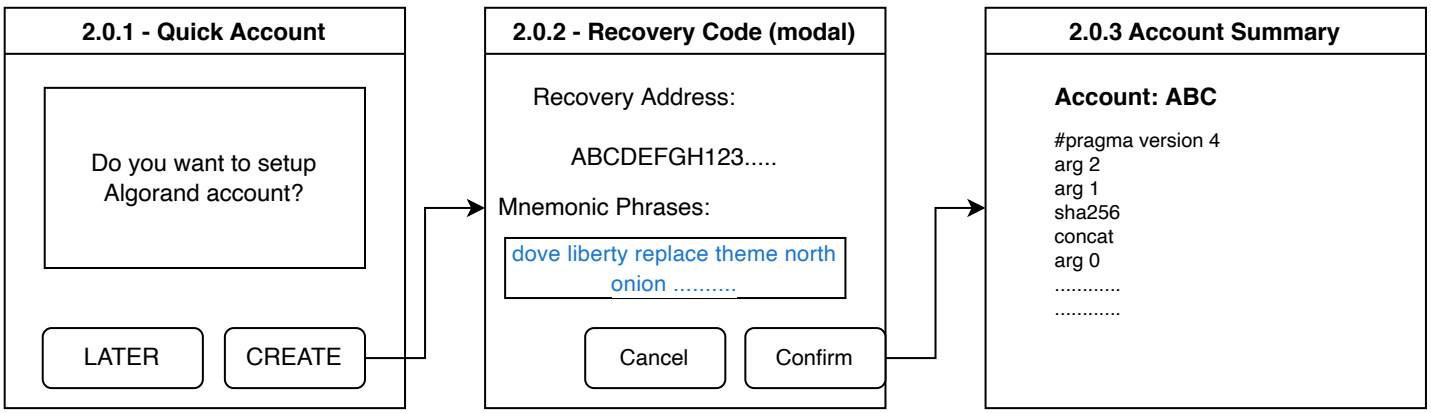
The *Add device* feature allows users to add multiple FIDO2 devices to their set of credentials. This will allow users to authenticate with any of their devices to their accounts. This process requires a user to obtain the registration code from the device they have already successfully registered (described later in 2.3.3).

From a new device, a user will get to the registration page (1.1.1). A user is required to enter their username and select the Add device checkbox (1.3.1). This will show an input box where they can enter the registration code they had obtained in 2.3.3. The user is then required to perform FIDO2 attestation to complete the process.

2.0 User account management component

This component has many feature-sets and is only accessible after the user performed authentication using their FIDO2 device.

For the first time user, the vault will present **Quick Algorand** account setup widget.

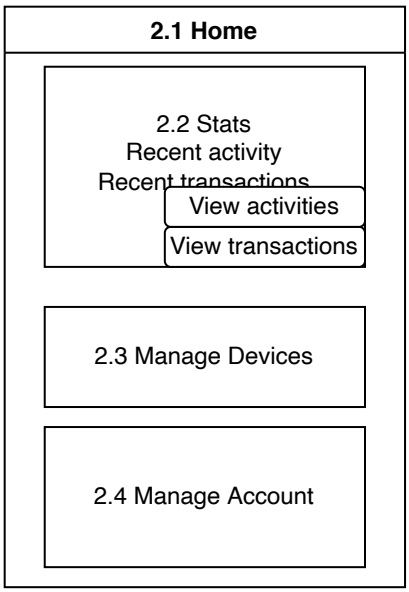


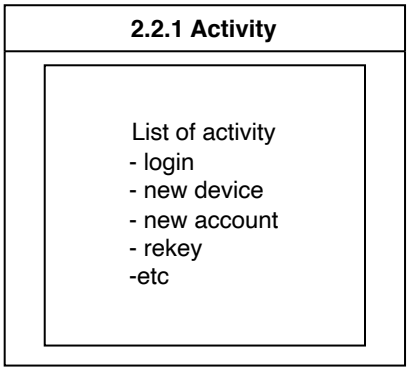
2.0.1 prompt the user to create an Algorand account based on their FIDO credential

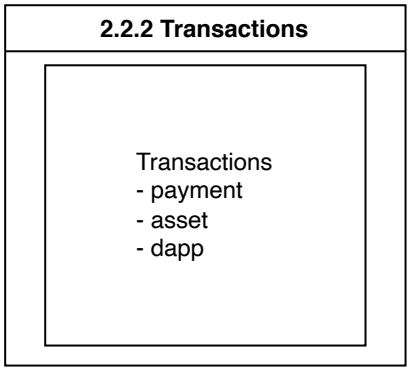
2.0.2 display the recovery address and key (mnemonic phrases) as a backup to their FIDO credential

2.0.3 display the Algorand address and its teal script based on user FIDO key and recovery key

2.1 - Home and statistics (2.2)







After users login they will land at the home section 2.1. There will be a quick summary of their recent activities and transactions. They can view the detailed statistics by selecting the appropriate menu to view their activities or transactions.

2.2.1 Activity view

This page shows a data set of a users recent activities:

- login action
- add device action
- create Algorand account action
- activate Algorand account action
- rekeying Alogrand account action

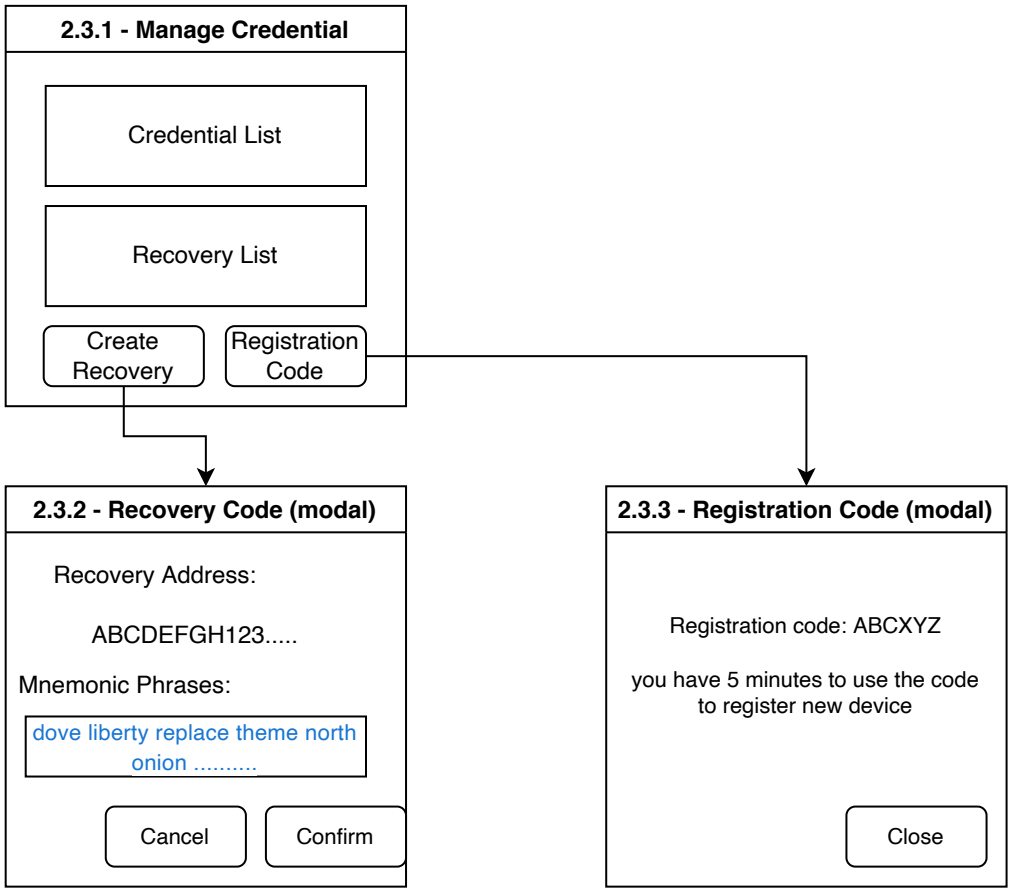
2.2.2 Transaction view

This page shows a data set of the users Algorand's transaction and account holdings:

- payment transaction
- asset transfer transaction

- dApp opt-in transaction
- dApp call transaction

2.3 Manage Devices



In this section (2.3.1) a user can view their current credentials and list of recovery codes (public-key). There are options for a user to create recovery codes and registration codes.

2.3.2 Recovery code

Recovery code is an ED25519 cryptographic key that is used as a backup of the users Algorand account in case their devices are damaged or lost. This code will allow the user to issue a rekeying action to the chain to update their FIDO2 signature script with a new device.

When the user selects recovery code, a modal window appears showing them the recovery address and Mnemonic phrases. A user is required to make a cold copy backup of the mnemonic phrases (private-key).

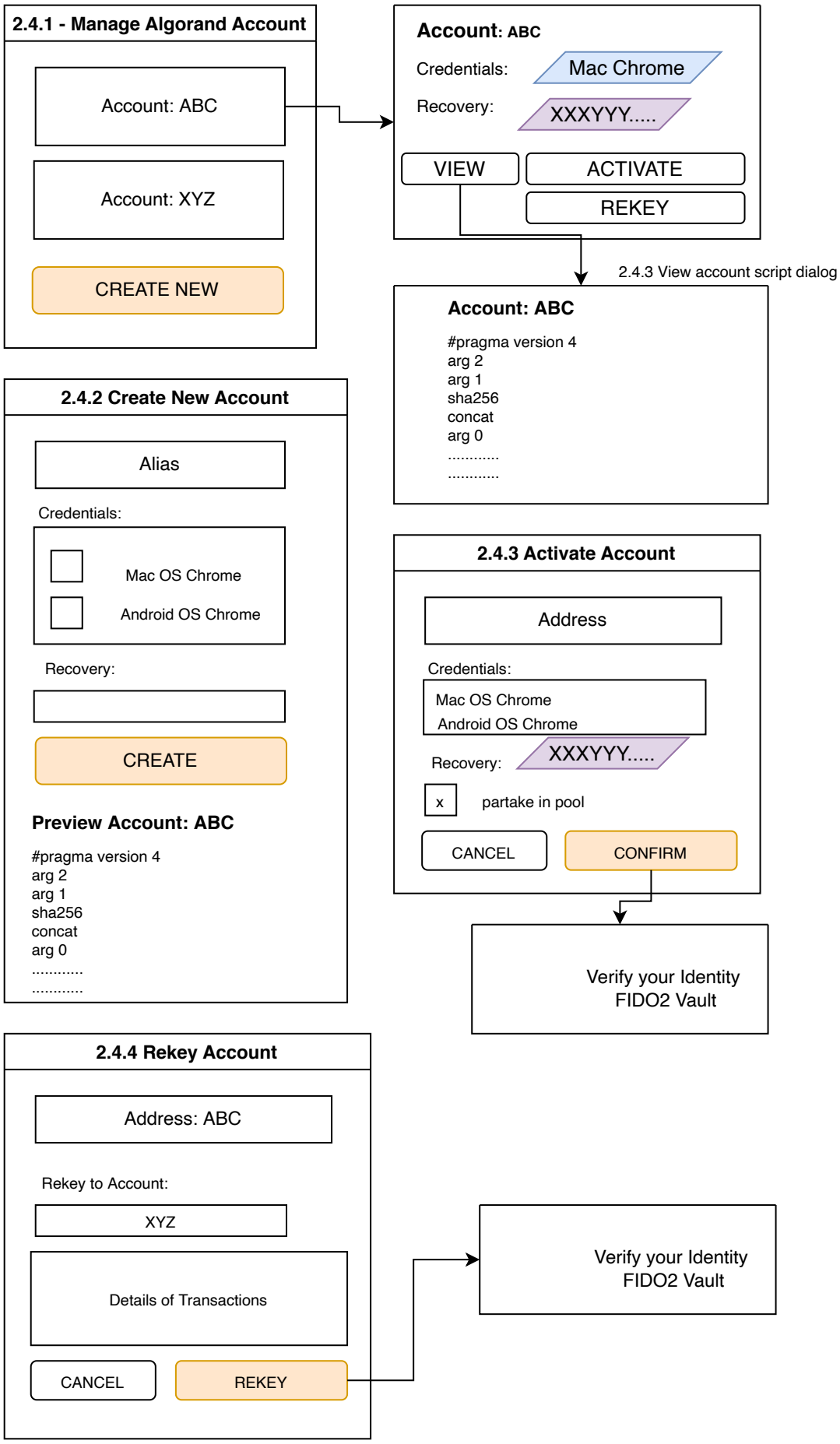
2.3.3 Registration code

A registration code is a short six-digit code that will expire within five minutes. This code will allow a user to register a new device with the account.

When the user selects the **create registration code** button, a modal window appears showing the code which a user can then use as part of the add device flow shown in 1.3.1 and 1.3.2.

2.4 Manage Algorand

In this section, 2.4.1, the user can view the list of current Algorand accounts they have. The user can create multiple accounts for signing transactions or for rekeying purposes.


2.4.1 - Manage Algorand Account
Account: ABC
Account: XYZ
CREATE NEW
Account: ABC
Credentials:
Mac Chrome
Recovery:
XXXYYY.....
VIEW
ACTIVATE
REKEY
2.4.3 View account script dialog
Account: ABC
#pragma version 4
arg 2
arg 1
sha256
concat
arg 0
.....
.....
2.4.2 Create New Account
Alias
Credentials:
Mac OS Chrome
Android OS Chrome
Recovery:
CREATE
Preview Account: ABC
#pragma version 4
arg 2
arg 1
sha256
concat
arg 0
.....
.....
2.4.3 Activate Account
Address
Credentials:
Mac OS Chrome
Android OS Chrome
Recovery:
XXXYYY.....
x
partake in pool
CANCEL
CONFIRM
Verify your Identity
FIDO2 Vault
2.4.4 Rekey Account
Address: ABC
Rekey to Account:
XYZ
Details of Transactions
CANCEL
REKEY
Verify your Identity
FIDO2 Vault

Algorand account creates what is considered to be a contract account. This is a TEAL script that contains the verifiable logic for a user's FIDO2 device assertion and recovery code. The detail of the TEAL script will be described in the technical section.

2.4.2 - Create New Account

A user is required to specify the following fields:

- alias - short name of this account
- credentials - this is a multi-select option where users choose any combination of created FIDO2 credentials
- recovery - this is a single select option where users choose one of the recovery addresses they had created in 2.3.2

Once all the fields are selected, a preview display of the TEAL scripts is generated. The TEAL preview will include the address of the account from the Algorand blockchain. The Address and TEAL scripts are unique per credentials and recovery combination. Duplicate combinations will be rejected.

2.4.3 - Activate Account

Users can sign transactions with an account once they have funded a minimum amount of 1 Algo. Once the user has funded (TBD method) the minimum requirement to use the account, they can choose to **Activate** the account within the vault to be used as a default transaction account.

2.4.4 - Rekey Account

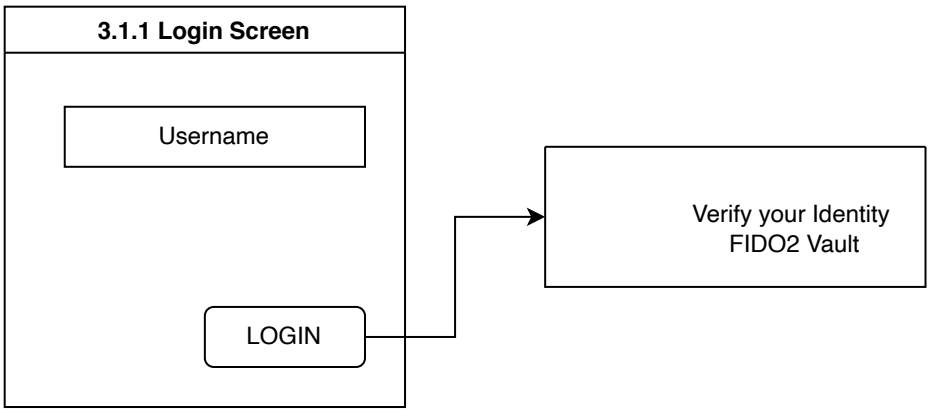
This page allows a user to rekey an active account to a different non-active account while maintaining activated account address and balance. Usually it is performed in an event of adding or deprecating devices relating to the current active account. This action requires FIDO2 transaction confirmation in order to authorize a **rekey** transaction on the blockchain.

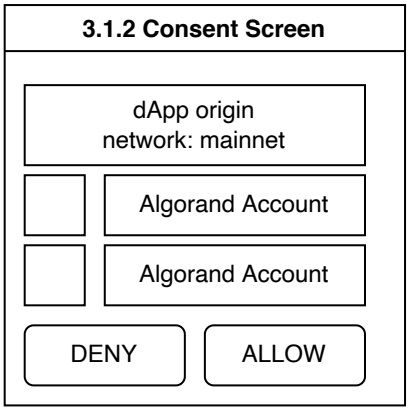
3.0 User transaction confirmation

Once users have an active Algorand account, they can start authorizing transactions with dApp that is integrated with the Vault defined in the transaction SDK of the **Design Overview** section.

3.1 User's consent/discovery call (Enable)

In order for the dApp to make any network transaction calls with the user's Vault the dApp is required to retrieve the user's consent. This call allows the dApp to request the permission to access the user's Algorand accounts based on a specific network type such as mainnet, testnet, etc. This transaction is called `enable` which follows the guideline defined by ARC-0006.





1. The `enable` call opens a popup window to the Vault site. If the user does not have an active login session then it will redirect the user to the login screen in 3.1.1. Upon a successful login, the Vault presents the consent screen to the user (3.1.2).
2. The user selects a list of active Algorand accounts from the Vault which they want to use to interact with the dApp.

3.2 Users transaction confirmation

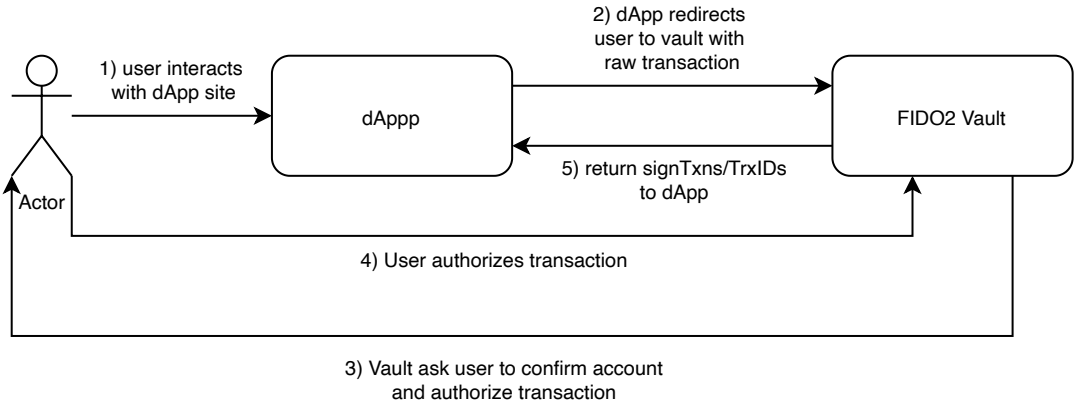
Once the dApp discovered the users Algorand account, the dApp can construct any set of transaction requests to the Vault for signing (`signTxns`) and posting (`signAndPostTxns`).

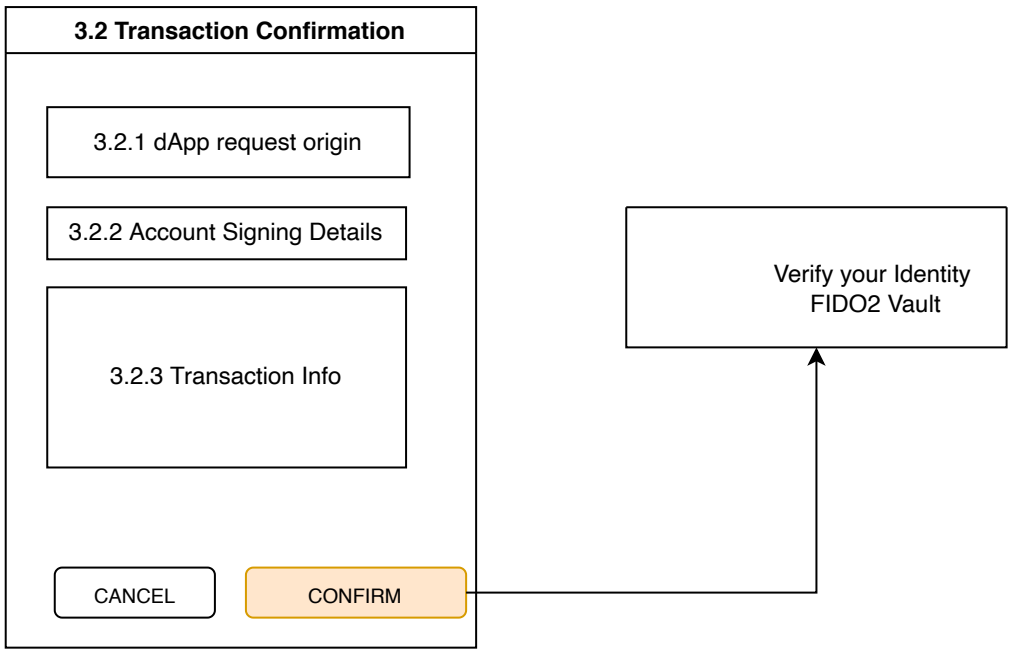
- The signing transaction follows ARC-0001 guidelines.
- The posting transaction follows ARC-0008 guidelines.

For the initial release phase, the Vault will support a single transaction request and filtered the transaction based on the following types:

- Payment - send Algo to another account
- Asset transfer - send any user owned asset to another account
- dApp opt-in - allows users to consent to dApp opt-in transactions
- dApp call - allows users to invoke method on the dApp

Here is a typical flow for a user transaction interaction with the Vault:





1. User interacts with dApp site ready to make a transaction with the dApp with their Vault account
2. dApp build payload is defined in the SDK and redirect user to the Vault transaction confirmation page (3.2)
3. User validates the transaction info (3.2.3) and signing detail (3.2.2).
4. User performs FIDO2 authentication to sign the transaction.
5. For a posting transaction `signAndPostTxns` , the Vault will submit the transaction to the blockchain with the FIDO2 signature data obtained from the users device and confirm the success or failure of the transaction by returning the txnID. The Vault returns the user back to the dApp site with the result of the transaction.

Design Overview

The Vault design is separated in two architectural components:

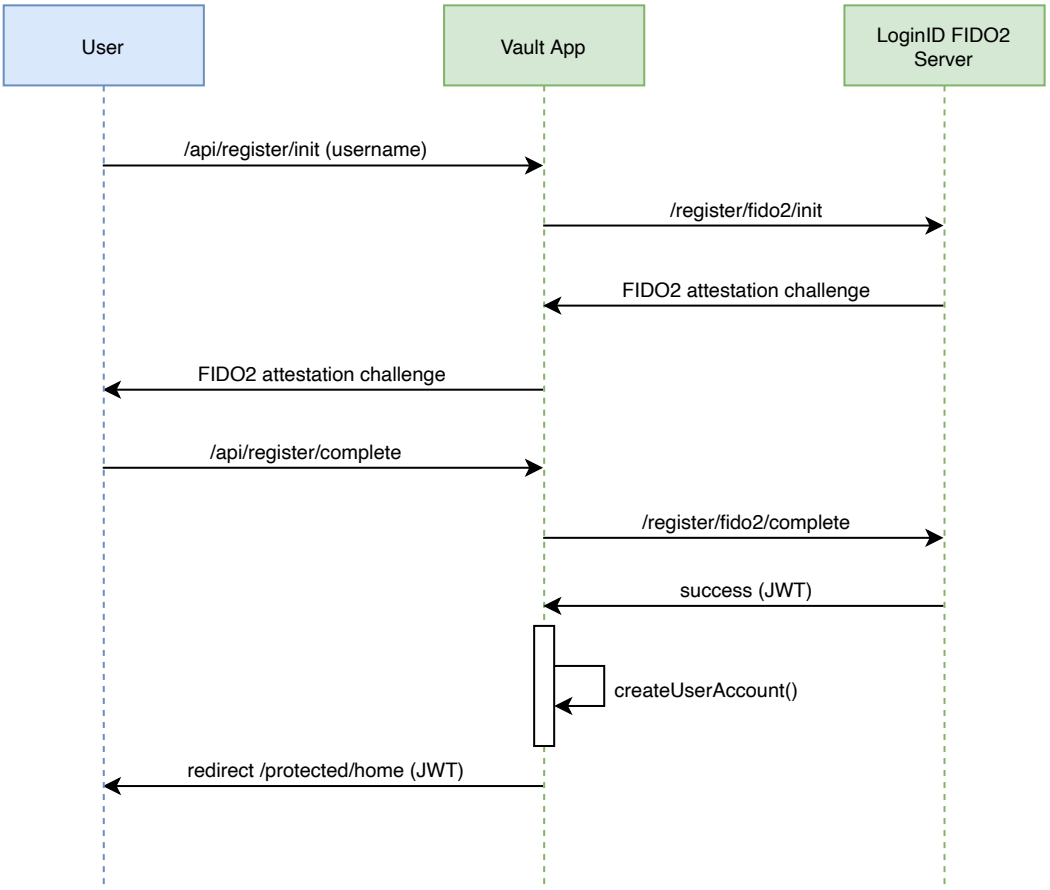
- The front-end component is built using the React framework
- The back-end component is built using Golang with PostgreSQL as its data store

Here we are going to look at some key design considerations for the use cases described in this documentation.

Registration and Authentication

The Vault will use the LoginID SDK to handle the FIDO2 attestation and authentication aspect between the users devices and Vault web app.

Here is the sequence diagram flow of the user registration page:



For the most part, the Vault acts as a proxy between the user and the LoginID service. At **createUserAccount()**, the app will need to extract the FIDO public key from the attestation payload and create a user account with the associated FIDO credential.

The **Authentication** and **Add Device** page interactions will be similar to the registration flow.

Algorand Management

To create an Algorand account the user chooses any combination of available FIDO credentials and one recovery code. These selections will be then substituted into the variables of the following PyTEAL script.

```
from pyteal import *
from inlineasm import *

def verify_fido(pk_x,pk_y,sig_r,sig_s,clientData,authData,server_challenge,tx_b64):
    decode_tx = InlineAssembly("base64_decode URLEncoding", tx_b64, type = TealType.byte
    compute_challenge = Sha256(Concat(tx_b64,server_challenge))
    extract_challenge = InlineAssembly("json_ref JSONString", clientData,
                        Bytes("challenge"), type = TealType.bytes)
    decode_challenge = InlineAssembly("base64_decode URLEncoding", extract_challenge,
                        type = TealType.bytes)
    message = Sha256(Concat(authData, Sha256(clientData)))
    # verify ecdsa
    verify = InlineAssembly("ecdsa_verify Secp256r1", message,  sig_r, sig_s,
            Bytes("base64", pk_x),Bytes("base64",pk_y),   type=TealType.uint64)
    return And(Txn.tx_id() == decode_tx, decode_challenge == compute_challenge, verify)

def verify_recovery(public_key, signature):
    return Ed25519Verify(Txn.tx_id(), signature, Addr(public_key))

def fido_signature(fido_pk1x,fido_pk1y,fido_pk2x, fido_pk2y, recovery_pk):

    sig1 = Arg(0)
    sig2 = Arg(1)
    clientData = Arg(2)
    authData = Arg(3)
    server_challenge = Arg(4)
    tx_b64 = Arg(5)
    return (
        If(verify_fido(fido_pk1x,fido_pk1y,sig1, sig2, clientData, authData, server_cha
        .Then(Int(1)) # exit success if fido_pk1 successful
        .ElseIf(verify_fido(fido_pk2x,fido_pk2y,sig1, sig2, clientData, authData, serve
        .Then(Int(1)) # exit success if fido2_pk2 successful
        .ElseIf(verify_recovery(recovery_pk,sig1))
        .Then(Int(1)) # exit success if recovery successful
        .Else(Int(0)) # exit fail
    )

if __name__ == "__main__":
    fido_1_x   = "FIDO1111XXXX"
    fido_1_y   = "FIDO1111YYYY"
    fido_2_x   = "FIDO2222XXXX"
    fido_2_y   = "FIDO2222YYYY"
    recovery_template = "XQWXNV7XXLCOWY5TKIKKSDAEEIN7J4FRF3SQJ4GES4KL43TGDT2FXY7UVI"
    program = fido_signature(
        fido_1_x,fido_1_y, fido_2_x, fido_2_y, recovery_template
    )
    print(compileTeal(program, mode=Mode.Signature, version=5))
```

The PyTEAL script is compiled and then presented to user for preview.

Transaction Confirmation SDK

The Vault will provide a public Typescript SDK for making dApp transactions. Here are the APIs supported by the SDK described in use cases section 3.0:

- `enable` - an account discovery function which returns the list of account addresses authorized by the users Vault
- `signTxns` - a transaction signing request function which returns the signed transaction in msgpack encoding
- `signAndPostTxns` - a transaction posting request function which returns the successful transaction ID

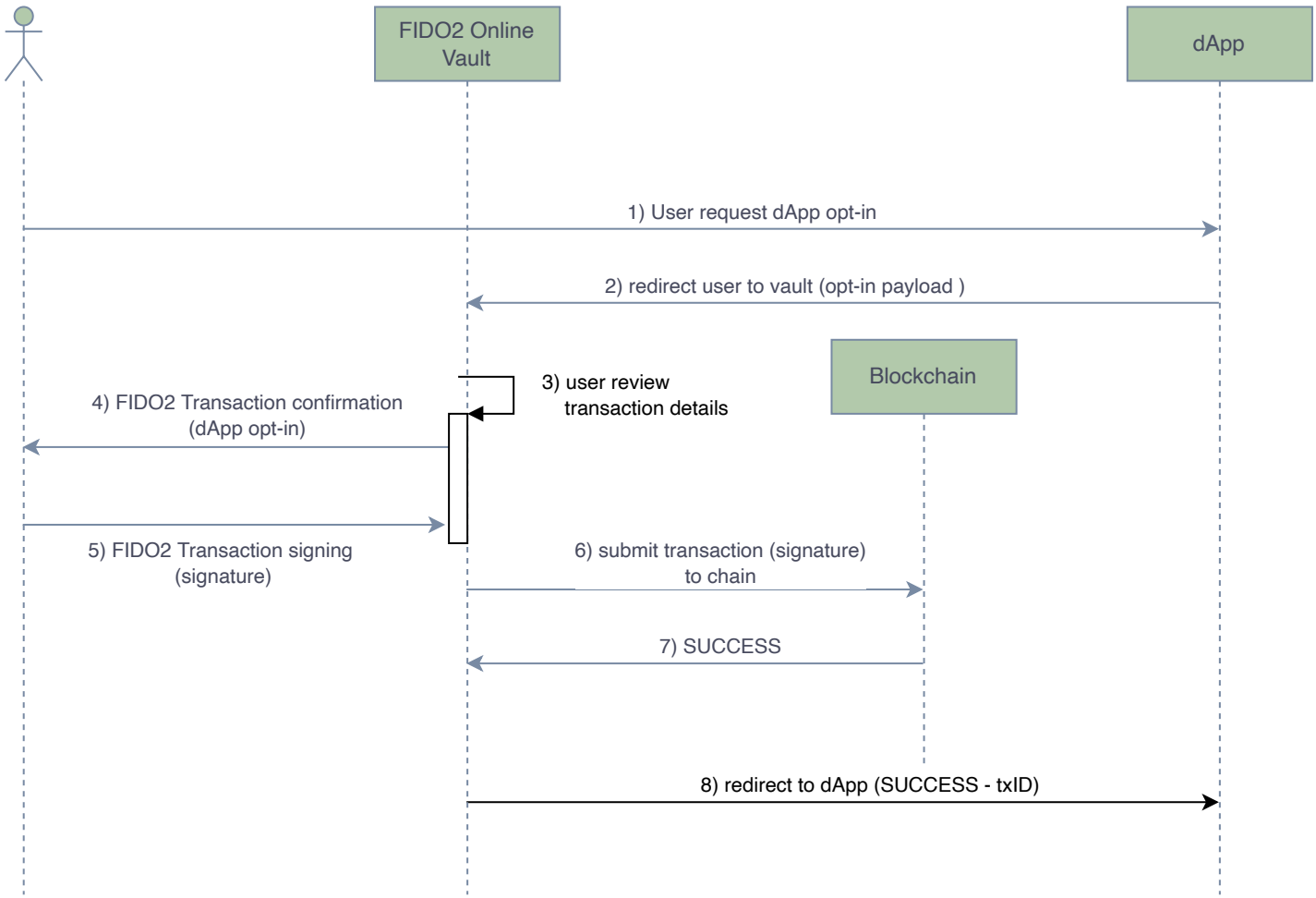
```
class VaultSDK {

    async enable(network: EnableOpts): Promise<EnableResult | null>  {
        /*....*/
    }

    async signTxns(txns: WalletTransaction[], opts?: SignTxnsOpts): Promise<(PostTxnsRes
        /*....*/
    }

    async signAndPostTxns(txns: WalletTransaction[], opts?: SignTxnsOpts): Promise<(Post
        /*....*/
    }
}
```

```ts
export interface EnableOpts  {
    network?: string;
    genesisID?: string;
    genesisHash?: string;
}
export interface EnableResult {
    genesisID: string;
    genesisHash: string;
    accounts: AlgorandAddress[];
}
export type SignTxnsOpts = {
    message?: string;
}

export interface TxnsResult {
    txnIds: TxnId[];
    signTxn: string[];
}

export interface WalletTransaction {
    /**
     * Base64 encoding of the canonical msgpack encoding of a Transaction.
     */
    txn: string;

    /**
     * Optional list of addresses that must sign the transactions
     */
    signers?: AlgorandAddress[];

    /**
     * Optional base64 encoding of the canonical msgpack encoding of a
     * SignedTxn corresponding to txn, when signers=[]
     */
    stxn?: string;

    /**
     * Optional message explaining the reason of the transaction
     */
    message?: string;

    /**
     * Optional message explaining the reason of this group of transaction
     * Field only allowed in the first transaction of a group
     */
    groupMessage?: string;
}
```

Sample Usage from dApp

```
// reference from ARC-0010 (https://github.com/algorandfoundation/ARCs/blob/main/ARCs/a

async function main(wallet) {

  // let wallet = new VaultSDK();
  // Account discovery
  const enabled = await wallet.enable({network: 'testnet-v1.0'});
  const from = enabled.accounts[0];

  // Querying
  const algodv2 = new algosdk.Algodv2(await wallet.getAlgodv2());
  const suggestedParams = await algodv2.getTransactionParams().do();
  const txns = makeTxns(from, suggestedParams);

  // Sign and post
  const res = await wallet.signAndPostTxns(txns);
  console.log(res);
}
```

For the sign and post `signAndPostTxns` api calls, the following sequence diagram demonstrates the interaction between the user and the dApp:



1. The user agrees to partake in a smart contract created by dApp

2. The dApp creates the opt-in transaction payload and submits using the `signAndPostTxns` api. This results in a popup window to the Vault's transaction confirmation
3. The Vault decodes and validates the transaction payload to confirm the transaction is safe and presents the breakdown view of the expected transaction. In this case, it will highlight the appID, creator address and type (opt-in) of this transaction payload
4. Upon confirmation, the user will receive a FIDO biometric prompt
5. Vault relays the FIDO signing data to the backend FIDO server for validation
6. The Vault extracts the signing data and builds the transaction logic signature and submits it to the blockchain
7. The result of the transaction is return to the dApp