FIDO2 Vault

Overview

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

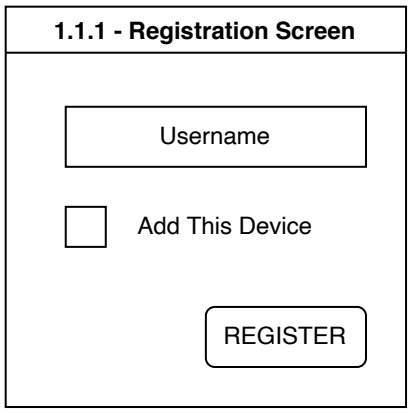
- Users can register and add multiple credentials from any FIDO2 supported devices
- Users can create recovery code based on ed25519 cryptography
- Users then create an Algorand account as a contract account (Logsig) based on the credentials and recovery code they have
- Users can interact with a dApp through a WebAuthn client (standard browsers) to authorize any transactions

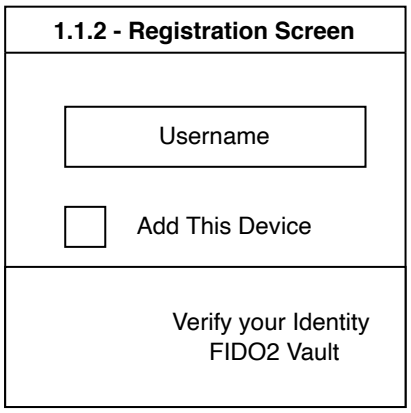
Use-Cases

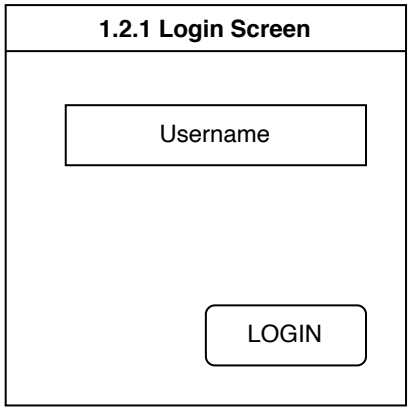
The vault is an online web-app with the following main components:

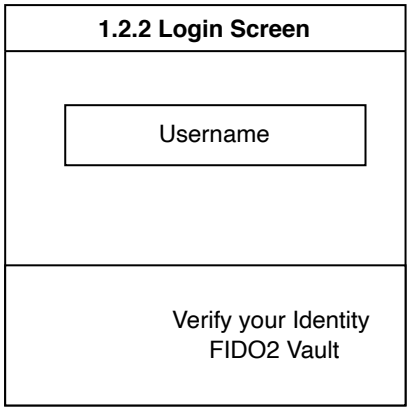
1.0 - user registration and authorization
2.0 - user account management
3.0 - user transaction confirmation
4.0 - dApp transaction SDK REST api

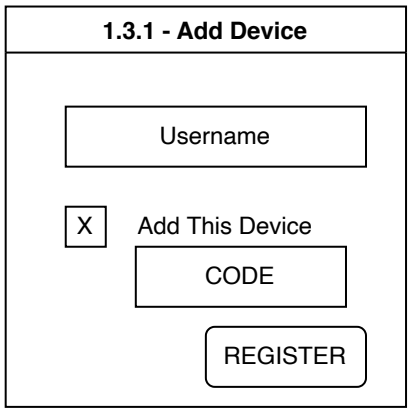
1.0 User registration and authorization component

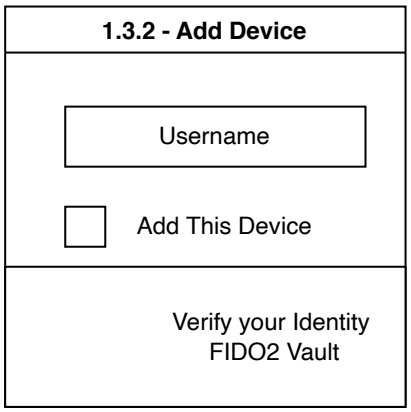












There are three feature-sets

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 completed, a user can sign back 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

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

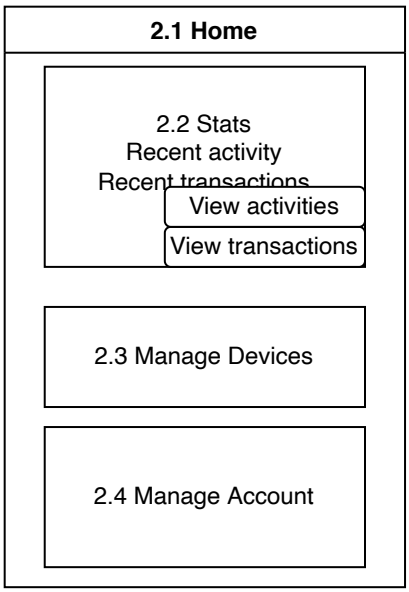
From a new device, a user will get to the registration page (1.1.1). A user is required to enter their username and select the Add device checkbox (1.3.1). This will show an input box where they can enter the registration code they had obtained in 2.3.3. The user is then required to perform FIDO2 attestation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

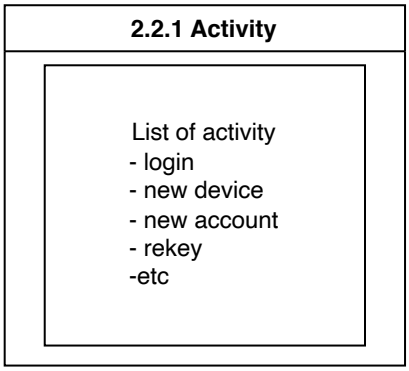
2.0 User account management component

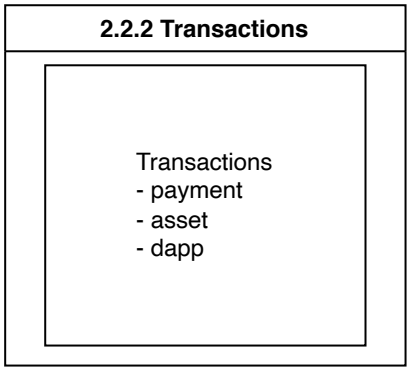
This component has many feature-sets and is only accessible after the user performed authentication with their FIDO2 device.

Here are the main feature-sets:

2.1 - Home and statistics (2.2)







After a user 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 detail statistics by selecting from the menu to view their activities or transactions.

2.2.1 Activity view

This page shows a data set of their recent activities

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

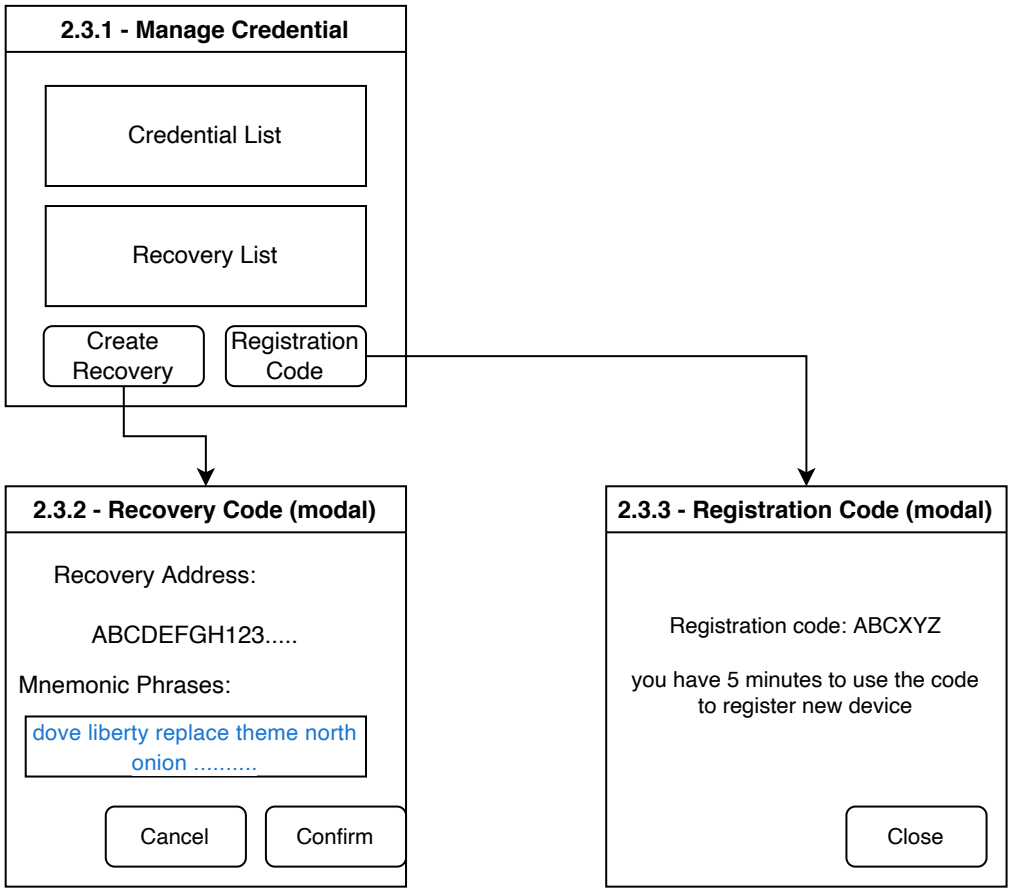
2.2.2 Transaction view

this page shows a data set of user Algorand's transaction and account holding

- 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 user Algorand account in case their devices are damage/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 selected 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).

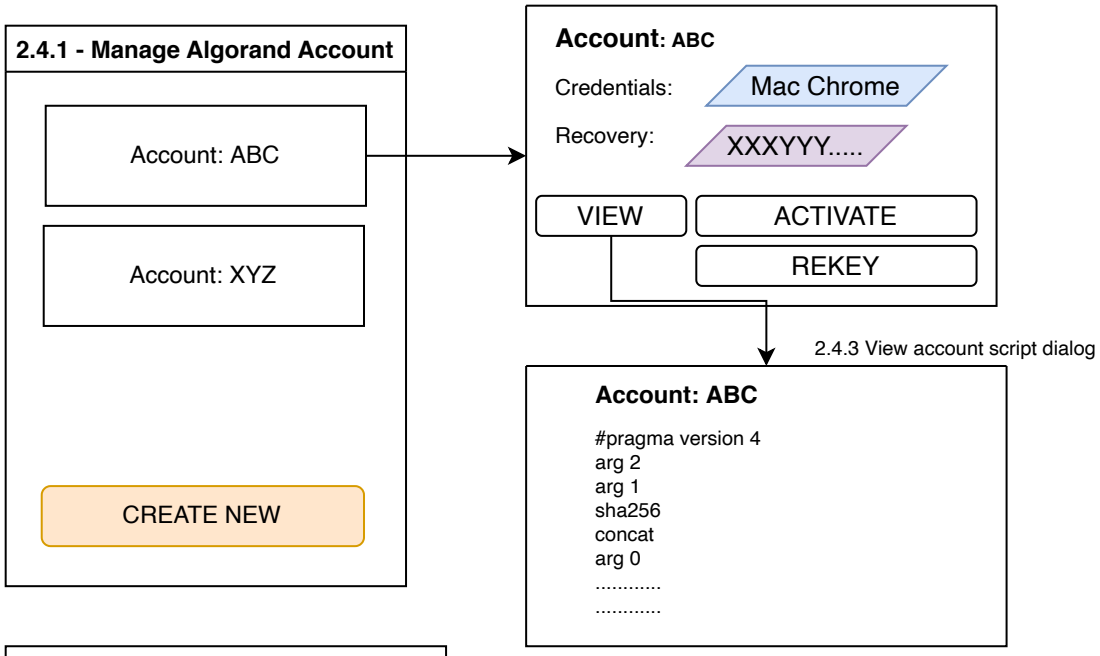
2.3.3 Registration code

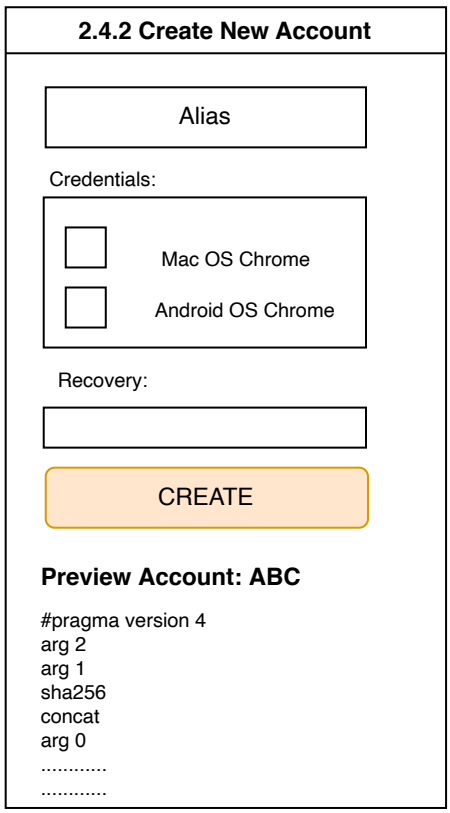
A registration code is a short six-digit code that will expire within five minutes of the issuing time. This code will allow a user to register a new device with the account.

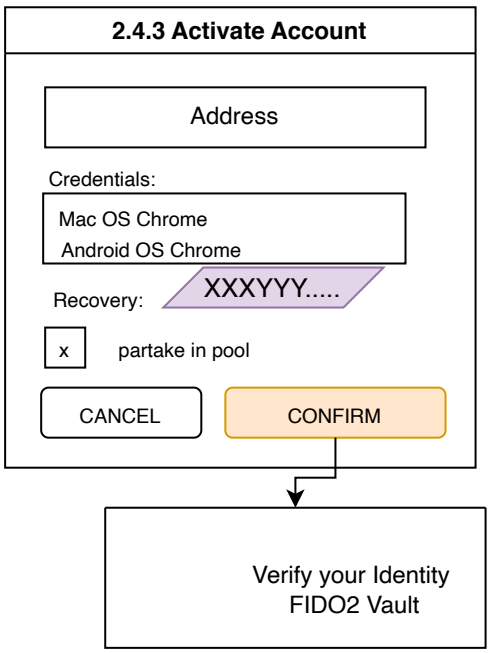
When the user selects the "create registration code" button, a modal window appears showing the code which a user can then use as part of the add device flow shown in 1.3.1 and 1.3.2.

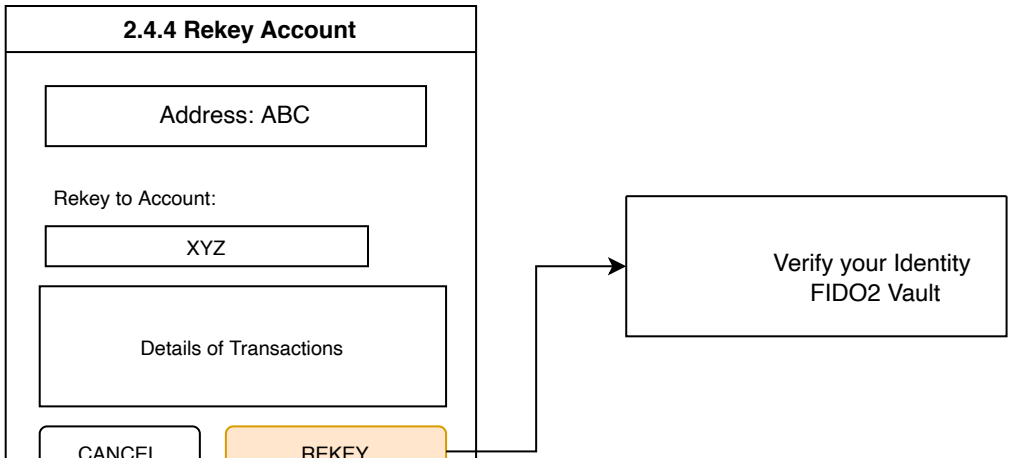
2.4 Manage Algorand

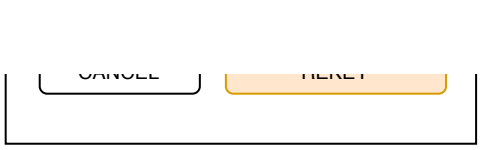
In this section 2.4.1, the user can view the list of current Algorand accounts they have. The user can create multiple accounts for signing transactions or for rekeying purposes.


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 TEAL script will be described in the technical section.

2.4.2 - Create New Account

User required to specify the following fields:

- alias - short name of this account
- credentials - this is a multi-select options where user chooses any combination of created FIDO2 credentials
- recovery - this is a single select option where user chooses one of the recovery address they had created in 2.3.2

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

2.4.3 - Activate Account

User can sign transactions with the account once they had funded a minimum amount of 1 Algo. Once the user had deposit (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.

In addition, there will be option for user to participate in the Algorand Consensus Protocol. This action requires FIDO2 transaction confirmation in order to authorize a **keyreg** transaction on the blockchain.

2.4.4 - Rekey Account

This page allows user to rekey an active account to a different non-active account while maintaining activate account address and balance. Usually it is performed in 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.

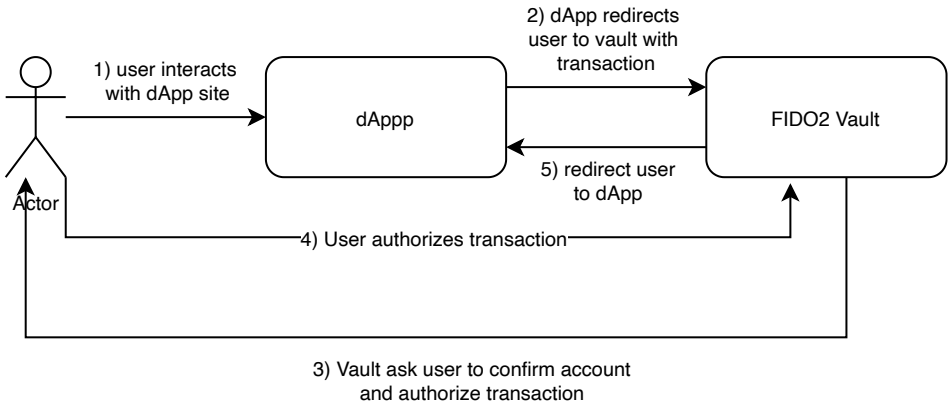
3.0 User transaction confirmation

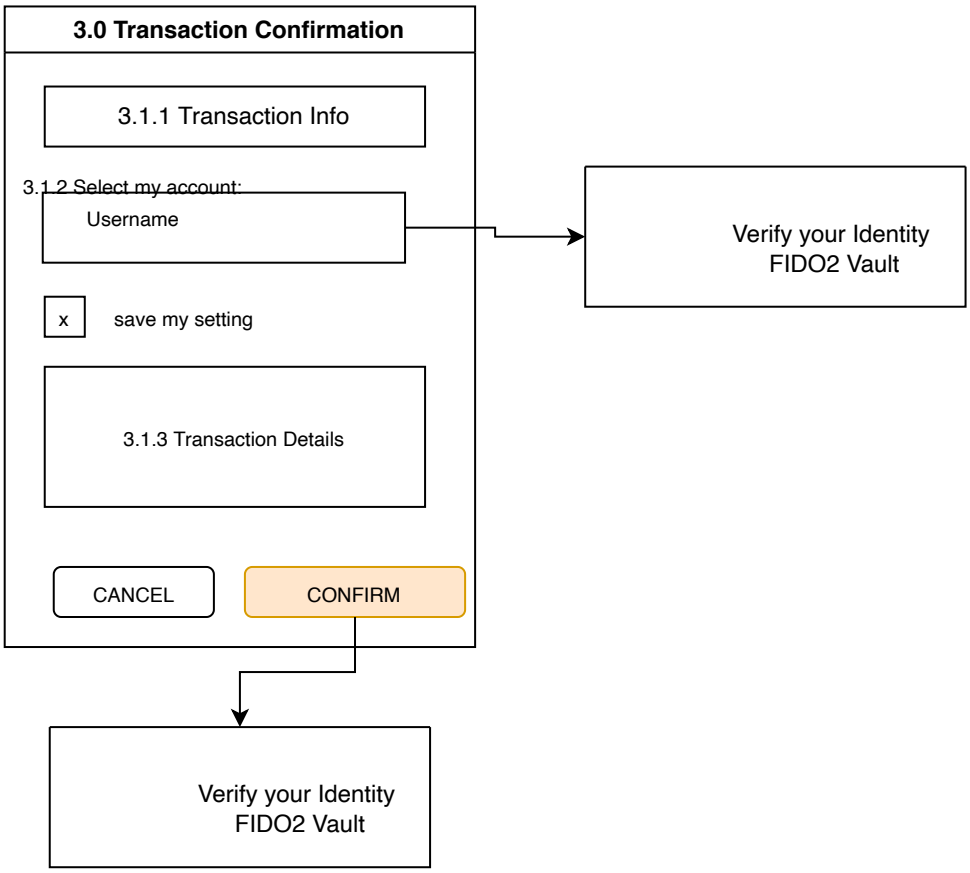
Once the user had an active Algorand account setup they can start authorizing transactions with dApp that integrated with the Vault transaction SDK defined in section 4.0.

Here are the type of transactions that is supported by the vault:

- 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 (section 4.0) to redirect user to the Vault transaction confirmation page (3.0)

3. User validates the transaction info (3.1.1) and selects an active account to authorize the transaction (3.1.2). Once a user has selected an account a detail transaction payload (3.1.3) is shown that will be submitted to the Algorand network.
    - By default, the Vault detected the user account based on their saved token stored on browser storage
    - If there is no token stored, the vault will prompt the user to enter username and FIDO2 authentication to retrieve the active Algorand account
4. User performs FIDO2 authentication to sign the transaction.
5. Vault will submit the transaction to the block with the FIDO2 signature data obtained from the user device and confirm the success or failure of the transaction. Vault redirects the user back to the dApp site with the result of the transaction.

> **3.1.3 - The transaction detail may include rating/warning for dApp transaction in future based risk assessment.**

4.0 dApp transaction SDK

The vault will support the following transaction SDK request by any dApp (describe in section 3.0).

4.1 - Payment request

4.1.1 request fields:

path: /api/transaction/request

Field Name	***Description***
type	payment
receiver	receiver address (rcv)
amount	amount in micro Algo(amt)
request-id	unique id associate with this request (lease)
note	optional note to add to the transaction
callback	url to callback after transaction result

```
    {
        "type": "payment",
        "reciever": "GD64YIY3TWGDMCNPP553DZPPR6LDUSFQOIJVFDPPXWEG3FVOJCCDBBHU5A",
        "amount": 5000000,
        "note": "SGVsbG8gV29ybGQ=",
        "request-id": "72635909-294e-4b0a-8776-6ee1ef26fbc0",
        "callback":"https://example.com/my-store/"

    }
```

4.1.2 response fields:

Field Name	***Description***
txid	transaction ID of the request
request-id	return request id associate with this transaction
status	success or error status of the request
error	error message if system or chain error occured
payload	detail of transaction payload submitted to the chain

```
{
    "status":"success",
    "txid":"X84ReKTmp+yfgmMCbbokVqeFFF.............",
    "request-id": "72635909-294e-4b0a-8776-6ee1ef26fbc0",
    "payload": {"txn": {
        "amt": 5000000,
        "fee": 1000,
        "fv": 6000000,
        "gen": "mainnet-v1.0",
        "gh": "wGHE2Pwdvd7S12BL5FaOP20EGYesN73ktiC1qzkkit8=",
        "lv": 6001000,
        "note": "SGVsbG8gV29ybGQ=",
        "rcv": "GD64YIY3TWGDMCNPP553DZPPR6LDUSFQOIJVFDPPXWEG3FVOJCCDBBHU5A",
        "snd": "EW64GC6F24M7NDSC5R3ES4YUVE3ZXXNMARJHDCCCLIHZU6TBEOC7XRSBG4",
        "type": "pay"
    }}
}
```

4.2 - Asset transfer request

4.2.1 request fields:

path: /api/transaction/request

Field Name	***Description***
type	asset-transfer type
receiver	receiver address (rcv)
asset-id	asset ID from user account to transfer
amount	unit amount of asset to transfer
request-id	unique id associate with this request
note	optional note to add to the transaction
callback	url to callback after transaction result

```
{
    "type": "asset-transfer"
    "reciever": "GD64YIY3TWGDMCNPP553DZPPR6LDUSFQOIJVFDPPXWEG3FVOJCCDBBHU5A"
    "asset-id": 168103
    "amount": 1000
    "request-id": "72635909-294e-4b0a-8776-6ee1ef26fbc0"
    "callback":"https://example.com/my-store/"
}
```

4.2.2 response fields:

Field Name	***Description***
txid	transaction ID of the request
request-id	return request id associate with this transaction
status	success or error status of the request
error	error message if system or chain error occured
payload	detail of transaction payload submitted to the chain

```
{
    "status":"success"
    "txid":"X84ReKTmp+yfgmMCbbokVqeFFF.............."
    "request-id": "72635909-294e-4b0a-8776-6ee1ef26fbc0"
    "payload": {"txn":
        {
            "aamt": 1000,
            "arcv": "QC7XT7QU7X6IHNRJZBR67RBMKCAPH67PCSX4LYH4QKVSQ7DQZ32PG5HSVQ",
            "fee": 3000,
            "fv": 7631196,
            "gh": "SGO1GKSzyE7IEPItTxCByw9x8FmnrCDexi9/cOUJOiI=",
            "lv": 7632196,
            "snd": "EW64GC6F24M7NDSC5R3ES4YUVE3ZXXNMARJHDCCCLIHZU6TBEOC7XRSBG4",
            "type": "axfer",
            "xaid": 168103
        }
    }
}
```


4.3 - dApp opt-in request

4.3.1 request fields:

path: /api/transaction/request

Field Name	*Description*
type	dapp-register
app-id	app ID for user to opt-in
request-id	unique id associate with this request
note	optional note to add to the transaction
callback	url to callback after transaction result


```
    {
        "type": "dapp-register"
        "app-id": 1337
        "request-id": "72635909-294e-4b0a-8776-6ee1ef26fbc0"
        "callback":"https://example.com/my-store/"
    }
```


4.3.2 response fields:

Field Name	*Description*
txid	transaction ID of the request
request-id	return request id associate with this transaction
status	success or error status of the request
error	error message if system or chain error occured
payload	detail of transaction payload submitted to the chain


```
{
    "status":"success"
    "txid":"X84ReKTmp+yfgmMCbbokVqeFFF.............."
    "request-id": "72635909-294e-4b0a-8776-6ee1ef26fbc0"
    "payload": {"txn":
        {
            "apan": 1,
            "apid": 1337,
            "fee": 1000,
            "fv": 13010,
            "gh": "ALXYc8IX90hlq7olIdloOUZjWfbnA3Ix1N5vLn81zI8=",
            "lv": 14010,
            "note": "SEQpWAYkzoU=",
            "snd": "LNTMAFSF43V7RQ7FBBRAWPXYZPVEBGKPNUELHHRFMCAWSARPFUYD2A623I",
            "type": "appl"
        }
    }
}
```


4.4 - dApp call

4.4.1 request fields:

path: /api/transaction/request

Field Name	*Description*
type	dapp-call
app-id	application ID
app-args	application arguments
app-account	application account address

Field Name	*Description*
app-fasset	application foreign asset
app-fapp	application foreign app
request-id	unique ID associate with this request
note	optional note to add to the transaction
callback	url to callback after transaction result

```
{
    "type": "app-call"
    "app-id": 1337
    "app-args": [
        "ZG9jcw==",
        "AAAAAAAAAAE="
        ]
    "request-id": "72635909-294e-4b0a-8776-6ee1ef26fbc0"
    "callback":"https://example.com/my-store/"
}
```

4.4.2 response fields:

Field Name	*Description*
txid	transaction ID of the request
request-id	return request id associate with this transaction
status	success or error status of the request
error	error message if system or chain error occured
payload	detail of transaction payload submitted to the chain

```
{
    "status":"success"
    "txid":"X84ReKTmp+yfgmMCbbokVqeFFF.............."
    "request-id": "72635909-294e-4b0a-8776-6ee1ef26fbc0"
    "payload": {"txn":
        {
            "apaa": [
            "ZG9jcw==",
            "AAAAAAAAAAE="
            ],
            "apid": 1337,
            "fee": 1000,
            "fv": 13376,
            "gh": "ALXYc8IX90hlq7olIdloOUZjWfbnA3Ix1N5vLn81zI8=",
            "lv": 14376,
            "note": "vQXvgqySYPY=",
            "snd": "LNTMAFSF43V7RQ7FBBRAWPXYZPVEBGKPNUELHHRFMCAWSARPFUYD2A623I",
            "type": "appl"
        }
    }
}
```


Design Overview

The Vault design is broken into two architectural components.

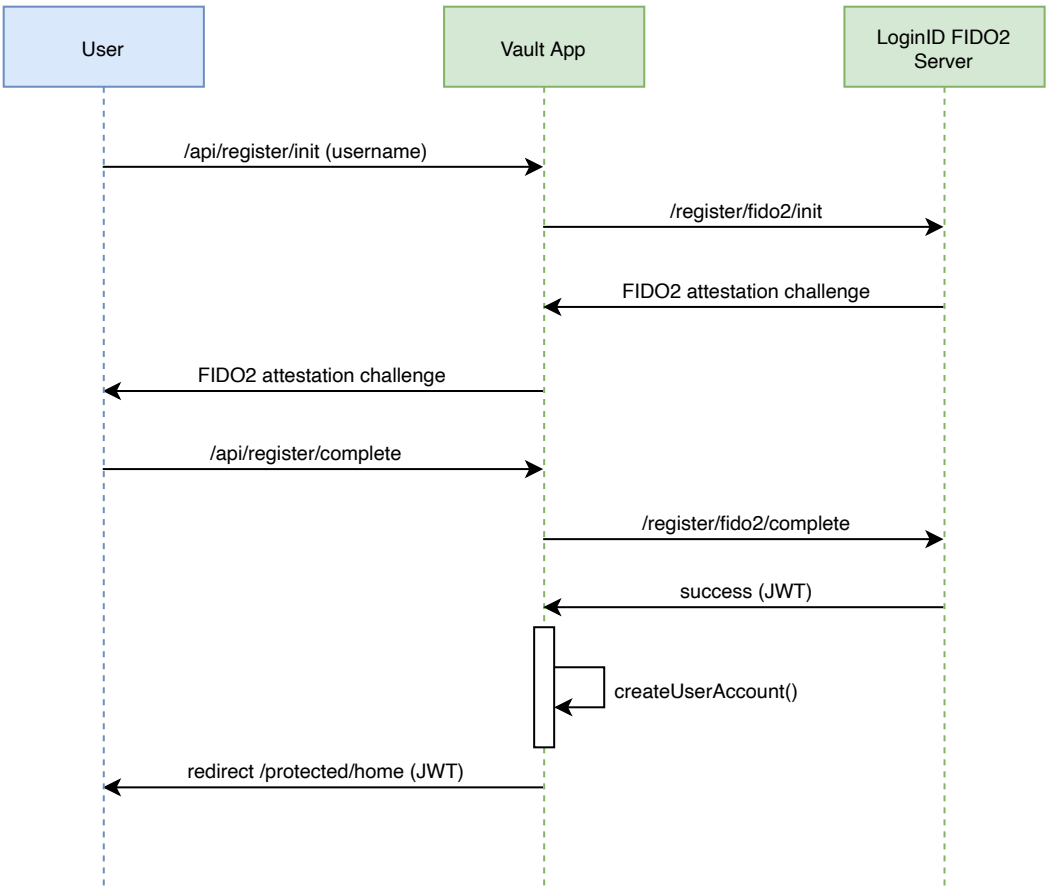
- The front-end component is built using React framework
- The back-end component is built using Golang with PostgreSQL as its data store.

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 user devices and Vault web app.

Here is the sequence diagram flow of user registration page:



For the most part the Vault acts a proxy between the user and the LoginID service. At the 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 the FIDO credentials and one recovery code. These selections will be then substituted into the variables of the following PyTEAL script.

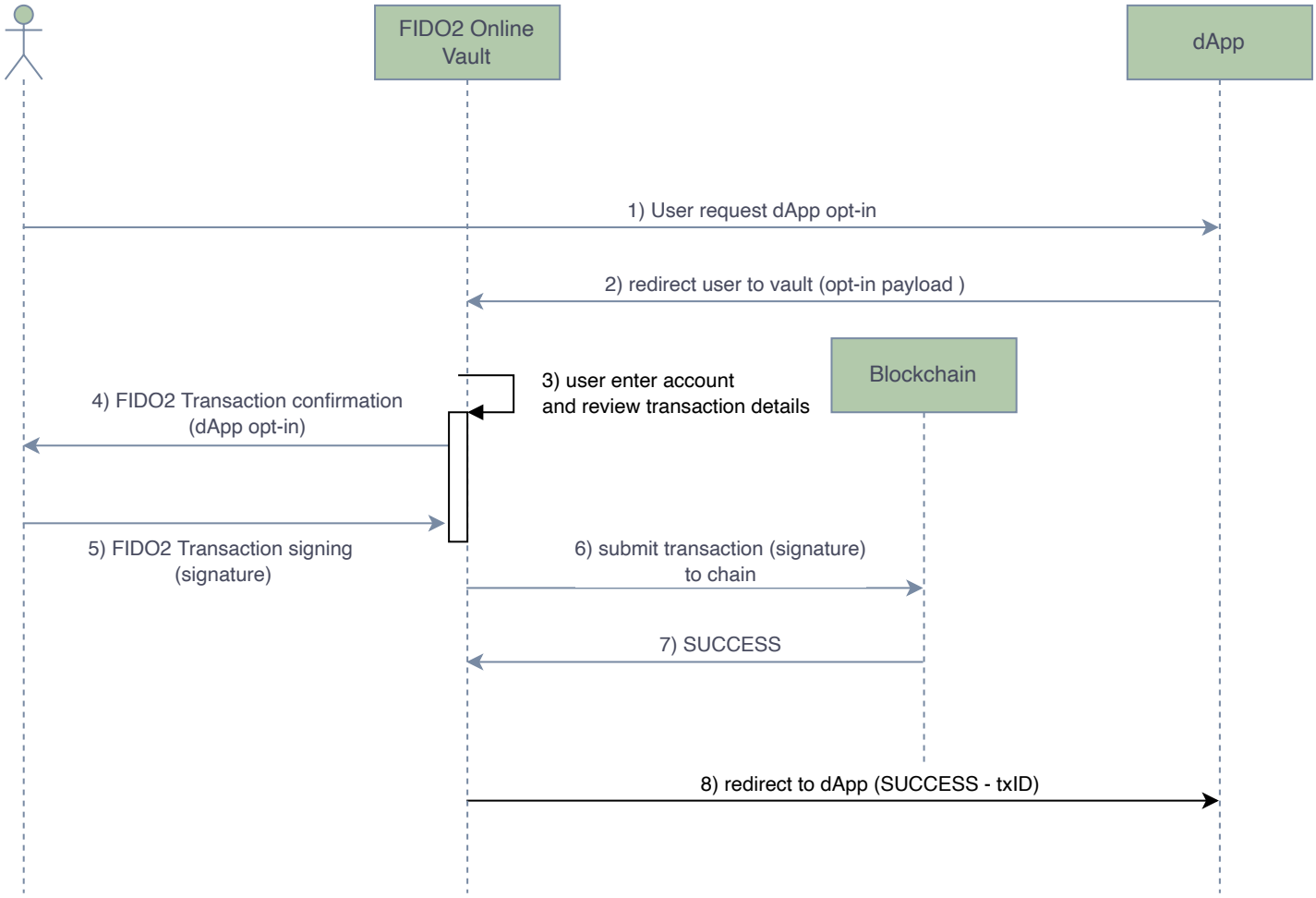
```
from pyteal import *

def verify_fido(pk,signature,clientData,authData,server_challenge):
    challenge = Concat(Txn.tx_id(),Txn.lease(),server_challenge)
    compute_challenge = base64url(challenge)
    extract_challenge = json_extract(clientData,"challenge"
    message = Concat(authData, Sha256(clientData))
    return And(compute_challenge == extract_challenge, Ecc_Verify(message,signature,Addr

def verify_recovery(public_key, signature):
    return And(Txn.type_enum() == TxnType.KeyRegistration , Ed25519Verify(Txn.tx_id(), s

def fido_signature(fido2_pk1,fido2_pk2,recovery_pk):

    signature = Arg(0)
    clientData = Arg(1)
    authData = Arg(2)
    server_challenge = Arg(3)


    return (
        If(verify_fido(fido2_pk1,signature, clientData, authData, server_challenge))
        .Then(Int(1)) # exit success if fido2_pk1 successful
        .ElseIf(verify_fido(fido2_pk2,signature,clientData, authData, server_challenge))
        .Then(Int(1)) # exit success if fido2_pk2 successful
        .ElseIf(verify_recovery(recovery_pk,signature))
        .Then(Int(1)) # exit success if recovery successful
        .Else(Int(0)) # exit fail
    )



if __name__ == "__main__":
    fido_1_template   = "AAAAA55555AAAAA55555AAAAA55555AAAAA55555AAAAA55555222244AO"
    fido_2_template   = "BBBBB55555BBBBB55555BBBBB55555BBBBB55555BBBBB5555522224444"
    recovery_template = "RRRRR55555RRRRR55555RRRRR55555RRRRR55555RRRRR5555522224444"

    program = fido_signature(
        fido_1_template, fido_2_template, recovery_template
    )
    print(compileTeal(program, mode=Mode.Signature, version=3))
```

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

> **Some opcodes are still in development. Hence, these arguments may change.**

Transaction Confirmation

Here are the sequence flow of a transaction confirmation. The dApp required to redirect user to vault using the SDK provided under the use cases section 4.



The dApp sends the user to the Vault with the request payload according to the SDK. At the transaction confirmation page, certain key fields will be highlighted to make sure users agree to the transaction. For example in a payment request, the vault will highlight the amount and the receiver's address where it will highlight the appID and callback origin for the opt-in transaction.