Rubix Cold-Hot wallet Installation Instructions

1. Setting up Cold Wallet

a. To set up a Rubix Cold Wallet, download the Rubix executable JAR from Rubix Github releases along with the file **didcreateimag.png** and save it to a portable drive such as a USB.
   Link to Releases
   Link to **didcreateimag.png**
b. Connect the portable drive to the Machine dedicated for cold wallet setup and copy the 2 files to a folder.
c. Open a command prompt window and navigate to the folder where the Rubix executable JAR and the png file are saved.
d. Once the above step is finished execute the below command.

```
java -jar <jar_name>.jar
```

e. Since the jar is now running, next step would be to create the shares namely, **DID.png, PrivateShare.png, PublicShare.png**
f. For creating the above 3 files open a new command prompt tab and execute the following curl command.

```
curl --location --request POST
'http://localhost:1898/newColdWallet' --form
'passPhrase="<passPhrase>"' --form
'image=@"<imagepath>"'
```

Please note to input PassPhrase and imagepath - imagepath would be the path where the didimagecreate.png is saved in the machine.

g. On Execution of the command in the previous step, if successful following would be the output received for the curl command.

h. In the Home folder of the Machine a Folder named **Rubix** would be created and the 3 file **DID.png, PrivateShare.png, PublicShare.png,** will be generated and stored in the Path **Rubix/DATA/RubixShares**

i. With this the Cold wallet is set up.

2. Setting up Hot Wallet

a. After the successful setup of the Cold wallet, moving on to the setup of the Hot wallet.

b. Using the ***one-click-setup*** script available for mac, windows and linux os from Rubix github page, install the pre-requisites, such as OpenJDK Java and ipfs. Follow the instructions given in github.

c. To check if ipfs is installed, in a cmd/terminal, type and execute the command ipfs id.

d. Now in the Hot wallet also download the Rubix Executable JAR and run it as in **Part 1.d**

e. Copy the files **DID.png and PublicShare.png** from the cold wallet in the path **Rubix/DATA/RubixShares,** to the system dedicated to be set up as a Hot Wallet.

f. Once above step is completed, execute the below curl command

```
curl --location --request POST
'http://localhost:1898/hotWallet' --form
```

```
'did=@"<imagepath>"' --form
'publicShare=@"<imagepath>"'
```

g. In the Previous step please provide the path where the DID.png and Publicshare.png are copied and stored in the machine designated for HotWallet.

h. On the Successful execution of the above curl command following will be the output.

i. The Hot wallet has been successfully set up

j. Copy the value of DID from the above response and in the cold wallet create a folder in the path **Rubix/DATA/** where folder name is the value of DID and move the DID.png , PrivateShare.png and PublicShare.png

k. Also copy the file named **DID.json** in the Path **Rubix/DATA/** from the Hot Wallet to the path **Rubix/DATA/** in the Cold Wallet.

l. After the completion of the above steps, please proceed to create the Private and Public Key pairs for the rubix node.

m. To Create the Private and Public Key pairs execute the below curl command. Enter the private key password according to user

```
curl --header "Content-Type: application/json"
--request POST
'http://localhost:1898/generateEcDSAKeys'
--data '{ "pvtKeyPass" : "<password>" ,
"returnKey" : 0}'
```

Please do note, the user will have sole responsibility to remember the password for the Private Key.

3. Transactions on Cold - Hot Wallet

Transactions from a Hot wallet to any other type of wallet can be performed by

a. Initiating a Transaction

i. To initiate a transaction from Rubix node open a new cmd/terminal tab on the Hot wallet system, and execute the below curl script.

```
curl --header "Content-Type: application/json"
--request POST
http://localhost:1898/initiateTransaction
--data '{ "receiver": "<Receiver DID>",
"tokenCount":2, "comment":"<any comment>",
"type":1, "pvtKeyPass":"<Sending node Pvt key
Password>"}'
```

ii. Since the Hot Wallet does not contain the PrivateShare.png that is required to sign transactions, the output of the above curl execution will be a file which holds the txn data to be signed.

iii. On the command line tab where the Rubix executable is running, after the successful execution of the above curl command, the below message will be displayed.

```
Challenge Payload for txnId
7e71777133c87d36c46c22f9ae3b67e5225e684760395c0ff06e8
5a4b99919b9
generated and saved to path
/home/nodeadmin/Rubix/Wallet/WALLET_DATA//ChallengePa
yload7e71777133c87d36c46c22f9ae3b67e5225e684760395c0f
f06e85a4b99919b9.json
```

iv. The ChallengePayload file for the corresponding txn contains the data that needs to be signed for txn to continue.

b. Signing a Transaction

i. To Sign an Offline transaction copy the Challenge Payload file, for the transaction that was initiated, to the Cold Wallet.

ii. Save the Challenge Payload file in the path **Rubix/Wallet/WALLET_DATA** in the cold wallet

iii. Once this is done, check whether the Rubix jar is started or not

iv. To Sign the challenge payload execute the below curl command, by supplying the txn ID of the Challenge Payload file.

```
curl --header "Content-Type:
application/json" --request POST
http://localhost:1898/signChallenge --data
'{"transactionID": ""}'
```

v. The result of the above curl command will be the Signed payload. Written into a file with name in the format **signedPayload<txnId>.json** saved in the path **Rubix/Wallet/WALLET_DATA**

c. Continuing a Transaction

i. Now after getting the Signed payload from Cold wallet, move the signed payload file from the cold wallet to the Path **Rubix/Wallet/WALLET_DATA** in the Hot Wallet.

ii. The next step will be to execute the below curl command to continue the txn and send the token to receiver

```
curl --header "Content-Type:
application/json" --request POST
http://localhost:1898/transactionFinality
--data '{"transactionID": ""}'
```

iii. In the above curl command the transaction Id field is mandatory and the value should be the ID that was generated when calling the curl command in **3.a.i**