How to use Near Chain Signatures in Flutterchain?

To use this feature, you need **NearBlockChainService** and a **BlockChainService** for the specific chain you want to use (e.g., **EthereumBlockChainService**, **BitcoinBlockChainService**, **XRPBlockChainService**). You can find the list of supported blockchains for Near CS in the **BlockChains** class from the variable **supportedBlockChainsForNearMPC**.

First, you need to generate an MPC account for the chosen blockchain. You can do this with the **NearBlockChainService** method called **getMPCAccount**.

```
final MPCAccountInfo mpcAccInfo = await
nearBlockChainService.getMPCAccount(
accountId: "flutterchain.testnet",
mpcPublicKey: "secp256k1:…",
chain: BlockChains.ethereum,
path: "flutterchain",
);
```

Here, “accountId” is your Near Account ID;
“chain” is the blockchain for which you want to generate an account (by default, **BlockChains.ethereum**);
“path” is the derivation path to generate the account (by default, 'flutterchain').
By changing the derivation path, you’ll get a new account for each one.
“mpcPublicKey” is the public key of the MPC account. By default, the library uses the MPC Contract with the address "v2.multichain-mpc.testnet" and its public key. However, if you want to change it, you

can get the public key of the chosen contract like this:

```
final mpcPublicKey =
  (await nearBlockChainService.callViewMethod(
     contractId: "mpcContractAddress",
     method: "public_key"))
 .data["response"];
```

MPCAccountInfo provides the address and public key of the created account.

The next step is to create a transaction for the chosen blockchain. For example, if it is Ethereum, we need to get transaction details such as gas, etc. To accomplish this, we use **getTransactionInfo**:

```
final transactionInfo =
   await ethereumBlockChainService.getTransactionInfo(
from: mpcAccountInfo.adress,
to: "reciever adress", //optional for smart contract
call
data: smartContractData, //optional for smart contract
call
amountInEth: 0.1, //optional for smart contract call
);
```

For EVM chains, it is also possible to call smart contracts, so we need to provide additional information to **getTransactionInfo** for this purpose. How to obtain **smartContractData** will be described later.

After we get **transactionInfo**, we can create a transaction to sign it with Near MPC. To do this, we call **createPayloadForNearMPC**:

```
final unsignedTx =
    await
ethereumBlockChainService.createPayloadForNearMPC(
  receiverAddress: "receiverAddress",
  amount: 0.1,
  transactionInfo: transactionInfo, // we already got
it in the previous step
  smartContractCallEncoded: smartContractData, //
optional if we want to call a smart contract
);
```

Now we have to sign it using the **NearBlockChainService** method **signEVMTransactionWithMPC**

```
final signedTX =
await nearBlockChainService.signEVMTransationWithMPC(
accountId: "Near account ID",
publicKey: "Near public key",
privateKey: "Flutterchain private key",
mpcTransactionInfo: unsignedTx,
senderAdress: mpcAccountInfo.adress,
path: "flutterchain", // derivation path of the MPC
account
mpcContract: "v2.multichain-mpc.testnet", // address of
the MPC contract
);
```

For each type of blockchain, there is a specific method to sign the transaction. If you changed the MPC Contract for account generation, you need to change it here as well. The function returns an already signed transaction, which we can then send:

```
final txSendInfo =
await
ethereumBlockChainService.sendTransaction(signedTX);
```

The result of executing it is a **BlockchainResponse** which includes the "status" of the execution and "data" that contains the transaction hash.

Smart Contract Call

To execute Smart Contract Calls for EVM blockchains, you need to create encoded smart contract call data using the method **encodeSmartContractCall**. All EVM blockchains have this method.

```
    final encodedRequest =
ethereumBlockChainService.encodeSmartContractCall(
      functionSignature: "function1(address, uint256)",
      parameters: ["0xsdasd...", 100],
    );
```

"functionSignature" is a function name of the smart contract and the type of arguments that it takes. If there are no arguments, then we leave it in the format "function1()".

"parameters" is an array arguments for this function. If there are no arguments, we leave it in the format "[]".