Problem

Use biconomy sdk for following tasks:

1. Deploy a biconomy smart account on multiple chains with the same address. It should allow EOA to sign multiple user operations (one per chain) on multiple chains with a single ECDSA signature. After the deployment, share the address of the deployed contracts.
2. Write a biconomy module (or alternatively, in *pseudo-code*) which has the ability to:
    a. Abort the transaction if a user spending amount crosses the limit value set by the user.
    b. Limit the number of transactions per day for a user.

Solution

Problem 1

Biconomy smart Account is deployed on the following chains using same ECDSA signature. Used Biconomy MultiChainValidationModule for the same.

i. Polygon Mumbai testnet,
ii. Arbitrum Sepolia testnet,
iii. Optimism Goerli,
iv. Linea Goerli testnet and
v. Base Goerli testnet

Deployed contract Address : 0xED0Bcf64F297427DEaDc7dA7CFa5b7A9D899dDeC

Github Repository : https://github.com/ArpanManna/Lightcurve/tree/main/coding2

Code :
https://github.com/ArpanManna/Lightcurve/blob/main/coding2/deployMultichainAccount.js

Contract Creation transactions

- Polygon Mumbai:
  https://mumbai.polygonscan.com/address/0xED0Bcf64F297427DEaDc7dA7CFa5b7A9D899dDeC#internaltx
- Arbitrum Sepolia:
  https://sepolia.arbiscan.io/address/0xED0Bcf64F297427DEaDc7dA7CFa5b7A9D899dDeC#internaltx

- Optimism Goerli:
  https://goerli-optimism.etherscan.io/address/0xED0Bcf64F297427DEaDc7dA7CFa5b7A9D899dDeC#internaltx
- Linea Goerli:
  https://explorer.goerli.linea.build/address/0xED0Bcf64F297427DEaDc7dA7CFa5b7A9D899dDeC/internal-transactions#address-tabs
- Base Goerli:
  https://goerli.basescan.org/address/0xED0Bcf64F297427DEaDc7dA7CFa5b7A9D899dDeC#internaltx

NFT Minting Transactions

Smart account is deployed with the first transaction only. As the first transaction minted a NFT (c**ontract address** : *0x1758f42Af7026fBbB559Dc60EcE0De3ef81f665e*, **chain -** *sepolia*) on each chain.

- Polygon Mumbai:
  https://mumbai.polygonscan.com/tx/0x83a2e1c8f094e6c42eb037c7560f66639a1ed98e6c634f1e6655c8386048b1e6
- Arbitrum Sepolia:
  https://sepolia.arbiscan.io/tx/0x4d5b8eba242e8a7d11d496e880a8d86a1fb252f0358963205dfa1e57072b4bc3
- Optimism Goerli:
  https://goerli-optimism.etherscan.io/tx/0xe166f4f00ffa050bfd1ddbca2c368df3c7b917d1091cfb37b98307220ca4564e
- Linea Goerli:
  https://explorer.goerli.linea.build/tx/0xd59b1496abcff9157ad76d30b766e2c181821c1f7a7d472756bcbea35ccb9600
- Base Goerli:
  https://goerli.basescan.org/tx/0x1a7f5caa094b9ea46b65ccfb09c61fc465ca0bc580d65813969647fefeabb8f8

Problem 2

Written a Biconomy module named ValidationModule, that performs the following validation

- Smart Account Initialization validation
- Revert transaction if spending limit reached
  - Validation added for the spending limit for individual token transactions also.
- Transaction limit per day

Code : https://github.com/ArpanManna/Lightcurve/blob/main/coding2/ValidationModule.sol

Spend limit validation

- For every userOp, spend limit validation is done through corresponding token limit associated with the transaction. Corresponding function - validateSpendLimit()
- Smart accounts can enable spend limit for individual token address by setSpendLimit() function
- Smart accounts can update spend limits by _updateLimit() function. Integer overflow and underflow is handled by solidity. (compiler version >= 0.8.2)

```
  function validateSpendLimit(address smartAccount, address tokenAddress, uint256 _amount)
public {
      _spendingLimits memory spendLimit = _spendingLimits[smartAccount][tokenAddress];
      // return if spending limit hasn't been enabled yet
      if (!spendLimit.isEnabled) return;
      // reverts if the amount exceeds the remaining available amount.
      require(spendLimit.availableToSpend >= _amount, "Exceed Spend limit");
      // decrement `available`
      spendLimit.availableToSpend -= _amount;
  }
```

Transaction limit validation

- Transaction limit is set while smart accounts initialization.

```
function initForSmartAccount(uint transactionLimit) {
    if (_transactionLimits[msg.sender].isEnabled){
        revert AlreadyInitedForSmartAccount(msg.sender);
    }
    _transactionLimits[msg.sender].isEnabled = true;
    _transactionLimits[msg.sender].Limit = transactionLimit;
    _transactionLimits[msg.sender].txAvailable = transactionLimit;
    return address(this);
}
```

- Transaction limit validation happens through validateTransactionCounts() function.
- For first userOp resetLimit is set to 1 day from current block.timestamp
- For subsequent userOps, if block.timestamp is within current resetLimit, validate based on available transaction to be allowed
- If block.timestamp exceeds resetLimit, transaction limit and resetLimit is updated.

```
  function validateTransactionCounts(address smartAccount) public {
    _transactionLimits memory txLimit = li_transactionLimitsmits[smartAccount];
    if (!txLimit.isEnabled) return;
    uint256 timestamp = block.timestamp; // L1 batch timestamp
    // Renew resetTime and available amount, which is only performed
    // if a day has already passed since the last update: timestamp > resetTime
    if (txLimit.limit != txLimit.txAvailable && timestamp > txLimit.resetTime) {
        txLimit.resetTime = timestamp + ONE_DAY;
        txLimit.txAvailable = txLimit.limit;

        // Or only resetTime is updated if it's the first spending after enabling limit
    } else if (txLimit.limit == txLimit.txAvailable) {
        txLimit.resetTime = timestamp + ONE_DAY;
    }
    require(txLimit.txAvailable <= txLimit.limit, "Exceed daily limit");
    // decrement `available transaction count`
    txLimit.txAvailable -= 1;
}
```