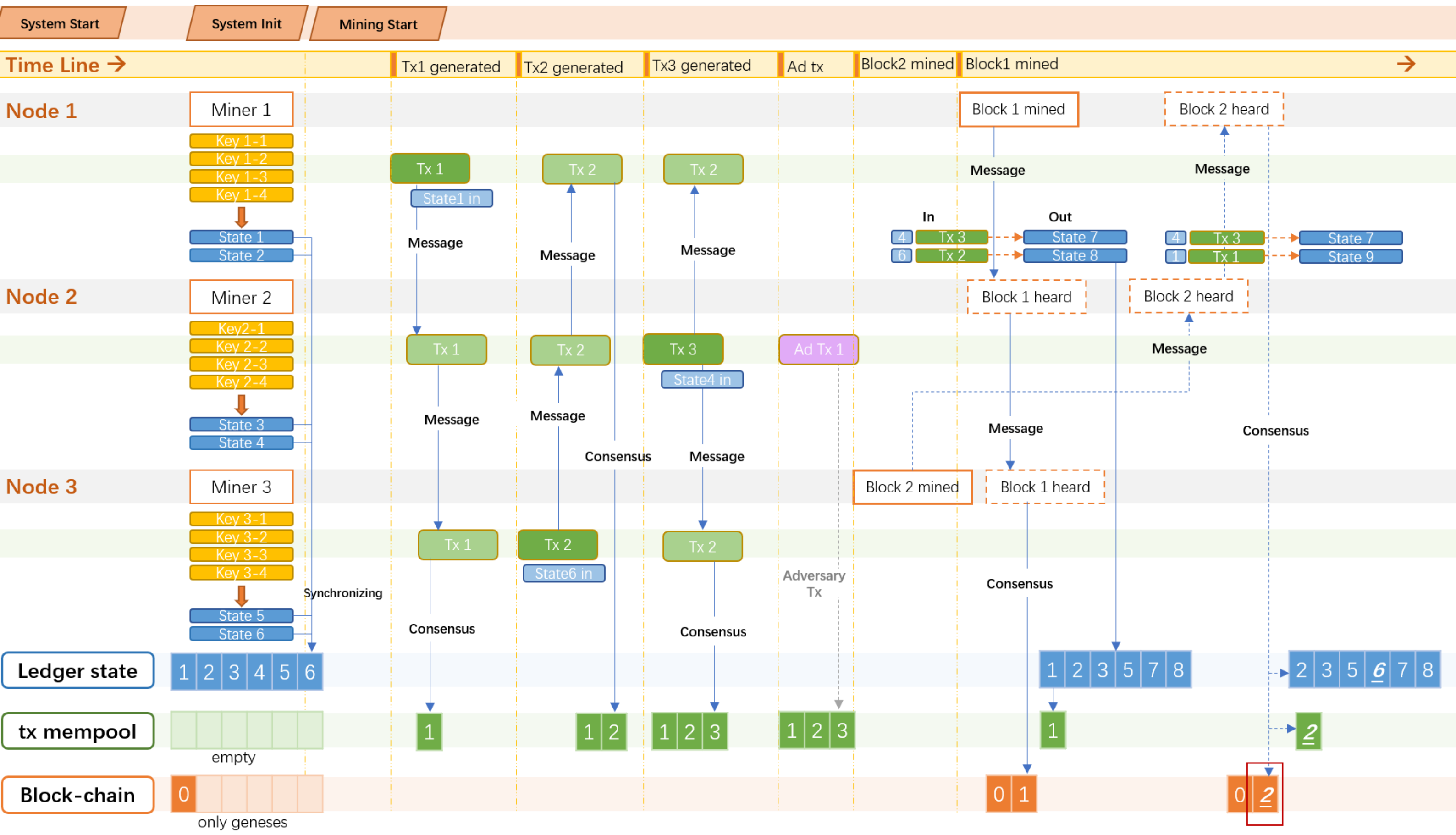
Part 5 & 6 Report

Name: Zhaojie Li   ***NetID***: ZLV8102

Design Document:

➢ **Design diagram:**

Figure below are a design diagram of my code. The diagram was pictured in timeline sequence. At the top is the stage system at: system started with initializing. Then is mining. Timeline shows the main event happening. Node1, node2, and node3 are things happened for each miner. Ledger state explains the consensus state change (UTXO change) when timeline going. Similarly, tx mempool and blockchain shows their consensus changing as time goes by. Details in diagram will explain in paragraphs later.



➢ **Details**

Words below state design thoughts (in deep black font) and details showed in diagram (in gray blue font).

My code realized a bitcoin system in UTXO model. The system was initialed with three nodes. Each nodes control 4 keypairs (randomly in different runs). When the system initialed, every miner will add 2 states in their ledger state value in block-chain. When mining starts, three nodes will synchronize their states with each other's periodically in whole mining process. This was done by sending message of their own states and receive new states from others. As in diagram, at initial stage, each node generated four keys. Node1 generated key 1-1 ~ key 1-4. Node2 generated key 2-1 ~ key 2-4. Node3 generated key 3-1 ~ key 3-4. Meanwhile, they generated state 1 to state 6 separately (each generated 2). When mining starts, in the figure, the first thing nodes doing will be synchronizing their states which same as in diagram. After synchronizing, the consensus ledger state turns to 1 to 6.

For transaction generating. I finished this in miner thread by setting probabilities about whether generating confirmed transactions or generating adversary transactions in a mining loop (adversary one is unqualified in different ways, basically randomly generated). That means, in each nonce trying loop, there's a probability that a transaction will be generated (Confirmed or adversary). After one confirmed transaction generated, the miner will send messages to peers and make the consensus in each node. By adjusting the mutex value properly, the code will not turn in dead lock and runs smoothly. In diagram, the most obviously part is the green blocks. Deep green blocks meaning generated transactions, light green blocks meaning transactions heard from other nodes, and purple blocks meaning adversarial transaction generated. Transaction generating is processing in mining loop as figure shows.

After each transaction generated (confirmed or adversary), the miner will check whether the transaction is qualified for ledger state which kept in block-chain. If it passes the check, this transaction will be added in to the transaction memory pool in chain controlled by this miner. Meanwhile, other miners get the new transaction message will check the validation in their own chain too, making sure everything goes right. In figure, with timeline going, the first transaction was generated by node1, this transaction was using UTXO1 in ledger state with input (UTXO1 = State1 in diagram). Other two nodes heard from node1 and having consensus with this confirmed transaction. After that, the transaction mempool adds transaction 1 – Tx1, which shows nodes are consensus on it. Same happens for Tx2 and Tx3 with UTXO6 and UTXO4 as input. When Ad Tx 1 generated, miner checked it and decide not telling other nodes for it. Because of that, after Ad tx 1 generated the consensus tx mempool is not changing.

When in the block mining processes, miners will select several transactions in block-chain's transaction memory pool. Before adding to content, miners will check the qualification again to make sure that this transaction is valid at this time. In every mining loop, after selecting proper number of transactions, an assumed block was generated with a random nonce. If the hash of the block was smaller than difficulty (in code), the block will be insert into block-chain. In figure above, block1 mined at some time. It contains two transactions – Tx2 and Tx3. Used 2 UTXO (state4 and state6) and generated 2 UTXO (state7 and state8). The tx mempool, ledger state, and block-chain update.

In my realization, the insert function has the ability of check whether this block should be the tip one. If it is the tip one and there's no confliction with other blocks, the state in new block's transaction input part will be deleted from ledger state. And the out put of transaction will be added in ledger state. Meanwhile, transactions included by block will be deleted from transaction memory pool. However, if the block has a height same as the tip, the function will check the timestamp in their header for decision. If the later insert one does have an earlier timestamp, the function will return the inserted old block's transaction and states. Then, dealing with the real tip. As in diagram, block2 was mined earlier than block one, but delayed sending information. So, when nodes heard block2, the block1 has already been insert into chain. UTXOs and transactions are handled already. By checking timestamp, Tx3 and Tx2 are returned in tx mempool. Same as state4 and state6. Then, Tx3 and Tx1 are selected for block2, things reset correctly for ledger state and tx mempool.

For transaction confirm, four check will be done in each confirmation. First, coin existing check. The input should be a valid UTXO (whether input is in ledger state). Second, owner check. The transaction's public key should consensus with the input state's address. Third, signature check. Checking whether the signature was signed by the public key in transaction. Third, value check. The output value should be smaller than input one.

I also realized transaction fee when mining block. At the beginning of initializing system, I give each miner a keypair as their own key. When a new block mined, there will be a 2 BTC fee for its miner and a UTXO was generated for their address. Though these UTXO are not included in generating transactions. (Maybe I'm the miner and want to save money instead of spend them). Transactions fees has a previous transaction hash 0 to differentiated with other normal one.

Division of Work

I did everything by myself. Don't have partner.