Report for the individual coursework

November 19, 2023

1 Instruction

This report is divided into three main sections:

- Bitcoin Testnet Transaction
- Smart contract for cash in DAML
- Deploy a constant product AMM contract in Solidity

2 Bitcoin Testnet Transaction

First, import the necessary modules.
Python code:

```
from ecc import PrivateKey
from helper import hash256, little_endian_to_int, decode_base58, SIGHASH_ALL
from script import p2pkh_script, Script
from tx import TxIn, TxOut, Tx
```

2.1 Create Bitcoin Testnet address

Create four Bitcoin Testnet addresses. Add below the addresses and the corresponding secrets, namely address1, address2, address3, and address4.
Python code:

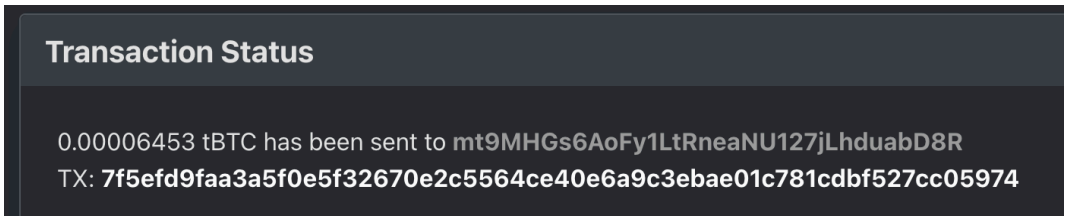
```python
# Q1: Create 4 Bitcoin Testnet addresses
# Create a Bitcoin Testnet address for address1
secret1 = little_endian_to_int(hash256(b'address1'))
private_key1 = PrivateKey(secret1)
address1 = private_key1.point.address(testnet=True)
print("Address1: "+address1)
print("Secret1: ",secret1)

# Create a Bitcoin Testnet address for address2
secret2 = little_endian_to_int(hash256(b'address2'))
private_key2 = PrivateKey(secret2)
address2 = private_key2.point.address(testnet=True)
print("Address2: "+address2)
print("Secret2: ",secret2)

# Create a Bitcoin Testnet address for address3
secret3 = little_endian_to_int(hash256(b'address3'))
private_key3 = PrivateKey(secret3)
address3 = private_key3.point.address(testnet=True)
print("Address3: "+address3)
print("Secret3: ",secret3)

# Create a Bitcoin Testnet address for address4
secret4 = little_endian_to_int(hash256(b'address4'))
private_key4 = PrivateKey(secret4)
address4 = private_key4.point.address(testnet=True)
print("Address4: "+address4)
print("Secret4: ",secret4)
```

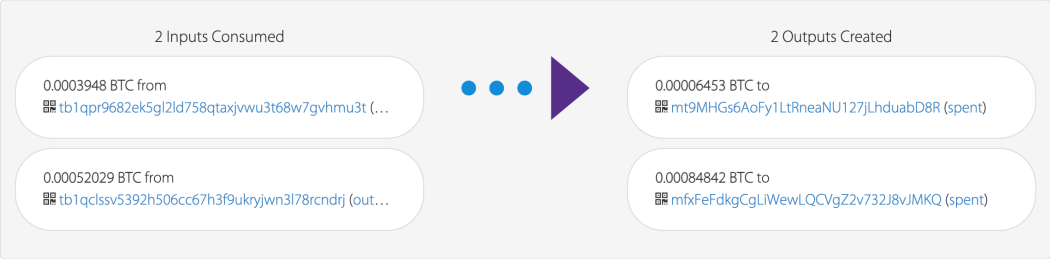
Results:

```
Address1: mt9MHGs6AoFy1LtRneaNU127jLhduabD8R
Secret1:  92796367777975090783698567212022215639238203613608596400466403599635546939269
Address2: mmhnuqdP8MsWP5pTBaga3s5Dax9KZvo4BL
Secret2:  56862173944063545459702801039276135097742049485800819803903611163252760829990
Address3: mwqRTEDBFNWo3ZAFcYAWovxtq6Muao69f9
Secret3:  10607853043198916084451497030701541794746030915917803470479264104914593293219O
Address4: mtS5MKhUgiGTqqeQzx6ux8kdegBawYwMsk
Secret4:  10590553989766538575374316431472311718236395476837620218244296531585794640541 7
```

2.2 Send testnet bitcoin to address 1

From a BTC testnet faucet, 0.00006453 BTC was sent to address 1.
Transaction screen shots:





Transaction ID: 7f5efd9faa3a5f0e5f32670e2c5564ce40e6a9c3ebae01c781cdbf527cc05974

2.3 Create a transaction

Create a transaction with 1 input and 3 outputs, transferring 50%, 30%, and 15% of the amount into address2, address3, and address4, respectively.

Python code:

```python
# Q3: Create a 1-input 3-outputs transactions
# 1 input
prev_tx = bytes.fromhex('7f5efd9faa3a5f0e5f32670e2c5564ce40e6a9c3ebae01c781cdbf527cc05974')
prev_index = 0
tx_in1 = TxIn(prev_tx, prev_index)

# 3 outputs
tx_outs = []
# Transfer 0.000032265 BTC (50%) to address2 - Output1
target_amount1 = int(3226.5)
target_h160_1 = decode_base58('mmhnuqdP8MsWP5pTBaga3s5Dax9KZvo4BL')
target_script1 = p2pkh_script(target_h160_1)
target_output1 = TxOut(amount=target_amount1, script_pubkey=target_script1)
# Transfer 0.000019359 BTC (30%) to address3 - Output2
target_amount2 = int(1935.9)
target_h160_2 = decode_base58('mwqRTEDBFNWo3ZAFcYAWovxtq6Muao69f9')
target_script2 = p2pkh_script(target_h160_2)
target_output2 = TxOut(amount=target_amount2, script_pubkey=target_script2)
# Transfer 0.0000096795 BTC (15%) to address4 - Output3
target_amount3 = int(967.95)
target_h160_3 = decode_base58('mtS5MKhUgiGTqqeQzx6ux8kdegBawYwMsk')
target_script3 = p2pkh_script(target_h160_3)
target_output3 = TxOut(amount=target_amount3, script_pubkey=target_script3)

# Define the transaction
tx_obj = Tx(1, [tx_in1], [target_output1, target_output2, target_output3], 0, True)
print(tx_obj)
```

Results:

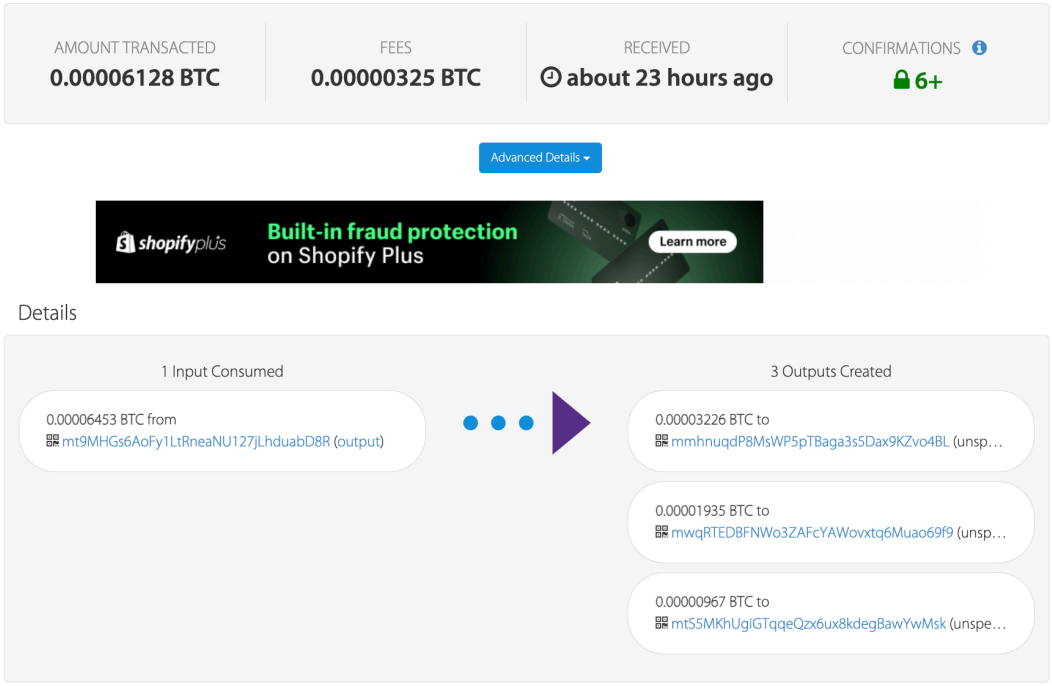
```
tx: 74d68669bced47b21f20c1a2fdcb8607d267d7cc870b6f774dde6e4d5b8e1e29
version: 1
tx_ins:
7f5efd9faa3a5f0e5f32670e2c5564ce40e6a9c3ebae01c781cdbf527cc05974:0
tx_outs:
3226:OP_DUP OP_HASH160 43dd7f6bae541939fd5a3febdfe313fbef10874f OP_EQUALVERIFY OP_CHECKSIG
1935:OP_DUP OP_HASH160 b3002b9ccbba468e39882907326bc4eb0db06c8f OP_EQUALVERIFY OP_CHECKSIG
967:OP_DUP OP_HASH160 8dad721bd31e65178f095b23c027d135a7a567bd OP_EQUALVERIFY OP_CHECKSIG
locktime: 0
```

2.4 Sign and submit the transaction

Sign the transaction and submit it to the Bitcoin testnet.
Python code:

```
z = tx_obj.sig_hash(0)
# use address1's secret
private_key = PrivateKey(secret=92796367777975090783698567212022215639238203613608596400466403599635546939269)
der = private_key.sign(z).der()
sig = der + SIGHASH_ALL.to_bytes(1, 'big')
sec = private_key.point.sec()
script_sig = Script([sig, sec])
tx_obj.tx_ins[0].script_sig = script_sig
print(tx_obj.serialize().hex())
```

Result:
01000000017459c07c52bfcd81c701aeebc3a9e640ce64552c0e67325f0e5f3aaa9ffd5e7f00000000
6a47304402200f83188ba51da1042f713299509600987e358eff3e5f0e01d39c00e5acbda0c402200f
ae62399d9cce94cbd8d1c77e97f1b675f87fbbdf7879cbe7032aa9cd2f0b8601210230a6b9404032c7
fe97c995ef43c800a646dd743d7d2536ba78d1319e5384781effffffff039a0c0000000000001976a9
1443dd7f6bae541939fd5a3febdfe313fbef10874f88ac8f070000000000001976a914b3002b9ccbba
468e39882907326bc4eb0db06c8f88acc703000000000001976a9148dad721bd31e65178f095b23c0
27d135a7a567bd88ac00000000
1-input 3-outputs transaction screen shot:



2.5 Transaction fee calculation

The transaction fee is calculated from the difference between the sum of the inputs and the sum of the outputs. In this case, the values of inputs and outputs are as follows:
Input1 = 0.00006453 BTC
Output1 = 0.000032265 BTC (50%)

Output2 = 0.000019359 BTC (30%)
Output3 = 0.0000096795 BTC (15%)
Therefore, the transaction fee is calculated as:

$$Transaction fee = input - (Output1 + Output2 + Output3) = 0.0000032265 BTC$$

As as result, the transaction fee is 0.0000032265 BTC.

2.6 Full python code

```
from ecc import PrivateKey
from helper import hash256, little_endian_to_int, decode_base58, SIGHASH_ALL
from script import p2pkh_script, Script
from tx import TxIn, TxOut, Tx

# Q1: Create 4 Bitcoin Testnet addresses
# Create a Bitcoin Testnet address for address1
secret1 = little_endian_to_int(hash256(b'address1'))
private_key1 = PrivateKey(secret1)
address1 = private_key1.point.address(testnet=True)
print("Address1: "+address1)
print("Secret1: ",secret1)

# Create a Bitcoin Testnet address for address2
secret2 = little_endian_to_int(hash256(b'address2'))
private_key2 = PrivateKey(secret2)
address2 = private_key2.point.address(testnet=True)
print("Address2: "+address2)
print("Secret2: ",secret2)

# Create a Bitcoin Testnet address for address3
secret3 = little_endian_to_int(hash256(b'address3'))
private_key3 = PrivateKey(secret3)
address3 = private_key3.point.address(testnet=True)
print("Address3: "+address3)
print("Secret3: ",secret3)

# Create a Bitcoin Testnet address for address4
secret4 = little_endian_to_int(hash256(b'address4'))
private_key4 = PrivateKey(secret4)
address4 = private_key4.point.address(testnet=True)
print("Address4: "+address4)
print("Secret4: ",secret4)
```

```
# Q2: from previous transaction the address1 received 0.00006453 BTC from BTC testnet faucet
# i.e. 6453 Satoshis
# Q3: Create a 1-input 3-outputs transactions
# 1 input
prev_tx = bytes.fromhex('7f5efd9faa3a5f0e5f32670e2c5564ce40e6a9c3ebae01c781cdbf527cc05974')
prev_index = 0
tx_in1 = TxIn(prev_tx, prev_index)

# 3 outputs
tx_outs = []
# Transfer 0.000032265 BTC (50%) to address2 - Output1
target_amount1 = int(3226.5)
target_h160_1 = decode_base58('mmhnuqdP8MsWP5pTBaga3s5Dax9KZvo4BL')
target_script1 = p2pkh_script(target_h160_1)
target_output1 = TxOut(amount=target_amount1, script_pubkey=target_script1)
# Transfer 0.000019359 BTC (30%) to address3 - Output2
target_amount2 = int(1935.9)
target_h160_2 = decode_base58('mwqRTEDBFNWo3ZAFcYAWovxtq6Muao69f9')
target_script2 = p2pkh_script(target_h160_2)
target_output2 = TxOut(amount=target_amount2, script_pubkey=target_script2)
# Transfer 0.0000096795 BTC (15%) to address4 - Output3
target_amount3 = int(967.95)
target_h160_3 = decode_base58('mtS5MKhUgiGTqqeQzx6ux8kdegBawYwMsk')
target_script3 = p2pkh_script(target_h160_3)
target_output3 = TxOut(amount=target_amount3, script_pubkey=target_script3)

# Define the transaction
tx_obj = Tx(1, [tx_in1], [target_output1, target_output2, target_output3], 0, True)
print(tx_obj)

z = tx_obj.sig_hash(0)
# use address1's secret
private_key = PrivateKey(secret=92796367777975090783698567212022215639238203613608596400466403599635546939269)
der = private_key.sign(z).der()
sig = der + SIGHASH_ALL.to_bytes(1, 'big')
sec = private_key.point.sec()
script_sig = Script([sig, sec])
tx_obj.tx_ins[0].script_sig = script_sig
print(tx_obj.serialize().hex())
```

3 Smart contract for cash in DAML

3.1 Contract template

3.1.1 Define template and variable type.

```
module Cash where

import Daml.Script

-- Define template and variable type
template Cash
  with -- 6 variables
    amount : Decimal
    currency : Text
    issuer : Party -- A bank
    holder : Party
    exchange : Party
    exchangeRate : Decimal
```

3.1.2 Define the roles of the parties and the condition.

```
-- Define the role of the parties
where
  signatory issuer
  ensure(  -- add the condition on the amount
    amount >= 0.0
  )
```

The issuer should be a signatory, and the exchange should be the controller.

3.1.3 Add functions for holder

```
controller holder can
  Transfer: (ContractId Cash
    ) -- Add a function to transfer the cash to the new holder
    with
      newHolder: Party
    do
      create this with
        holder=newHolder

  UpdateExchangeRate: (ContractId Cash
    ) -- Add a function that sets a new exchange rate
    with
      newExchangeRate: Decimal
    do
      create this with
        exchangeRate=newExchangeRate
```

3.1.4 Add function for exchange

```
    controller exchange can
      Swap:(ContractId Cash
        )
        with
          newCurrency: Text
        do
          assert( --exchangeRate should greater than 1.0
            exchangeRate>1.0
            )
          create this with
            currency=newCurrency
            amount=amount/exchangeRate
```

3.2 Scenario testing

3.2.1 Testing code

```
-- Scenario testing
cashTests: Script ()
Script results
cashTests = script do

-- Add parties
party1 <- allocateParty "the issuer"
party2 <- allocateParty "the holder"
party3 <- allocateParty "the exchange"


-- Create contract
cash1 <- submit party1 do
  createCmd Cash with
    amount = 100.0
    issuer = party1
    holder = party1
    exchange = party3
    exchangeRate = 0.0
    currency = "USD"

--- Transfer the cash
payCash1 <- submit party1 do
  exerciseCmd cash1 Transfer with
    newHolder = party2

--- Update the exchange rate
newRate1 <- submit party2 do
  exerciseCmd payCash1 UpdateExchangeRate with
    newExchangeRate = 1.2

--- Swap the currency
swap1 <- submit party3 do
  exerciseCmd newRate1 Swap with
    newCurrency = "GBP"

return()
```

3.2.2 Script results

Script: cashTests

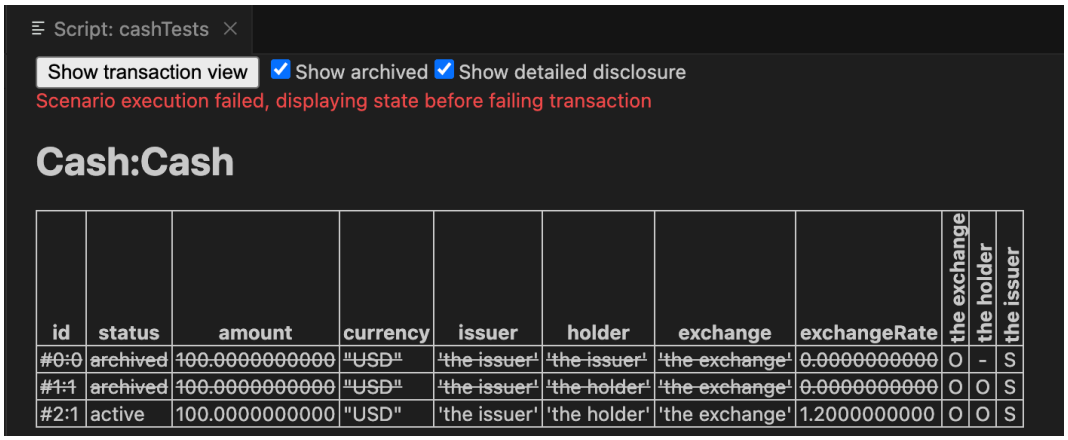
Show transaction view | Show archived | Show detailed disclosure

Cash:Cash

id	status	amount	currency	issuer	holder	exchange	exchangeRate	the exchange	the holder	the issuer
~~#0:0~~	~~archived~~	~~100.0000000000~~	~~"USD"~~	~~'the issuer'~~	~~'the issuer'~~	~~'the exchange'~~	~~0.0000000000~~	O	-	S
~~#1:1~~	~~archived~~	~~100.0000000000~~	~~"USD"~~	~~'the issuer'~~	~~'the holder'~~	~~'the exchange'~~	~~0.0000000000~~	O	O	S
~~#2:1~~	~~archived~~	~~100.0000000000~~	~~"USD"~~	~~'the issuer'~~	~~'the holder'~~	~~'the exchange'~~	~~1.2000000000~~	O	O	S
#3:1	active	83.3333333333	"GBP"	'the issuer'	'the holder'	'the exchange'	1.2000000000	O	O	S

3.2.3 Try to let the holder do the swap

Substitute 'party3' with 'party2' in the last segment of the scenario testing (swap the currency part). As a consequence, an error occurred, indicating that the scenario execution failed. The reason behind this failure is attributed to the holder lacking control over the 'swap' function, the only controller is the exchange, hence causing the script results to terminate prematurely. The final step remains unavailable due to this issue.



id	status	amount	currency	issuer	holder	exchange	exchangeRate	the exchange	the holder	the issuer
~~#0:0~~	~~archived~~	~~100.0000000000~~	~~"USD"~~	~~'the issuer'~~	~~'the issuer'~~	~~'the exchange'~~	~~0.0000000000~~	O	-	S
~~#1:1~~	~~archived~~	~~100.0000000000~~	~~"USD"~~	~~'the issuer'~~	~~'the holder'~~	~~'the exchange'~~	~~0.0000000000~~	O	O	S
#2:1	active	100.0000000000	"USD"	'the issuer'	'the holder'	'the exchange'	1.2000000000	O	O	S

3.3 Full DAML code

```
module Cash where

import Daml.Script

-- Define template and variable type
template Cash
  with -- 6 variables
    amount : Decimal
    currency : Text
    issuer : Party -- A bank
    holder : Party
    exchange : Party
    exchangeRate : Decimal

  -- Define the role of the parties
  where
    signatory issuer
    ensure(  -- add the condition on the amount
        amount >= 0.0
      )

    controller holder can
      Transfer: (ContractId Cash
        ) -- Add a function to transfer the cash to the new holder
        with
          newHolder: Party
        do
          create this with
            holder=newHolder

      UpdateExchangeRate: (ContractId Cash
        ) -- Add a function that sets a new exchange rate
        with
          newExchangeRate: Decimal
        do
          create this with
            exchangeRate=newExchangeRate

    controller exchange can
      Swap:(ContractId Cash
        )
        with
          newCurrency: Text
        do
          assert( --exchangeRate should greater than 1.0
            exchangeRate>1.0
            )
          create this with
            currency=newCurrency
            amount=amount/exchangeRate
```

```
-- Scenario testing
cashTests: Script ()
Script results
cashTests = script do

-- Add parties
party1 <- allocateParty "the issuer"
party2 <- allocateParty "the holder"
party3 <- allocateParty "the exchange"

-- Create contract
cash1 <- submit party1 do
  createCmd Cash with
    amount = 100.0
    issuer = party1
    holder = party1
    exchange = party3
    exchangeRate = 0.0
    currency = "USD"

--- Transfer the cash
payCash1 <- submit party1 do
  exerciseCmd cash1 Transfer with
    newHolder = party2

--- Update the exchange rate
newRate1 <- submit party2 do
  exerciseCmd payCash1 UpdateExchangeRate with
    newExchangeRate = 1.2

--- Swap the currency
swap1 <- submit party3 do
  exerciseCmd newRate1 Swap with
    newCurrency = "GBP"

return()
```

4 Deploy a constant product AMM contract in Solidity

4.1 Full contract code in solidity

```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

interface IERC20 {
    function totalSupply() external view returns (uint);

    function balanceOf(address account) external view returns (uint);

    function transfer(address recipient, uint amount) external returns (bool);

    function allowance(address owner, address spender) external view returns (uint);

    function approve(address spender, uint amount) external returns (bool);

    function transferFrom(
        address sender,
        address recipient,
        uint amount
    ) external returns (bool);

    event Transfer(address indexed from, address indexed to, uint value);
    event Approval(address indexed owner, address indexed spender, uint value);
}

// token 1 with name comp163_1 and symbol comp1
contract token1 is IERC20 {
    // Specify the total supply of tokens
    uint public totalSupply = 100;

    // Create a mapping to keep track of balances
    mapping (address => uint) public balanceOf;

    // Create a mapping to keep track of allowances
    mapping (address => mapping(address => uint)) public allowance;

    // Specify the name of the token
    string public name = "comp163_1";

    // Specify the symbol of the token
    string public symbol = "comp1";

    // Specify the number of decimals for the token
    uint public decimals = 18;
```

```
// Initialize the total supply and allocate all tokens to the contract creator
constructor(){
    balanceOf[msg.sender] = totalSupply;
}

// Transfer tokens from the sender to the recipient, and emit the Transfer event
function transfer(address recipient, uint amount) external returns (bool) {
    require(balanceOf[msg.sender] >= amount);
    balanceOf[msg.sender] -= amount;
    balanceOf[recipient] += amount;
    emit Transfer(msg.sender, recipient, amount);
    return true;

}

// Approve the spender to spend the specified amount of tokens on behalf of the owner, and emit the Approval event
function approve(address spender, uint amount) external returns (bool) {
    allowance[msg.sender][spender] = amount;
    emit Approval(msg.sender, spender, amount);
    return true;

}

// Transfer tokens from the sender to the recipient, and emit the Transfer event
function transferFrom(
    address sender,
    address recipient,
    uint amount
) external returns (bool) {
    require(balanceOf[sender] >= amount);
    require(allowance[sender][msg.sender] >= amount);
    balanceOf[sender] -= amount;
    allowance[sender][msg.sender] -= amount;
    balanceOf[recipient] += amount;
    emit Transfer(sender,recipient,amount);
    return true;
}
```

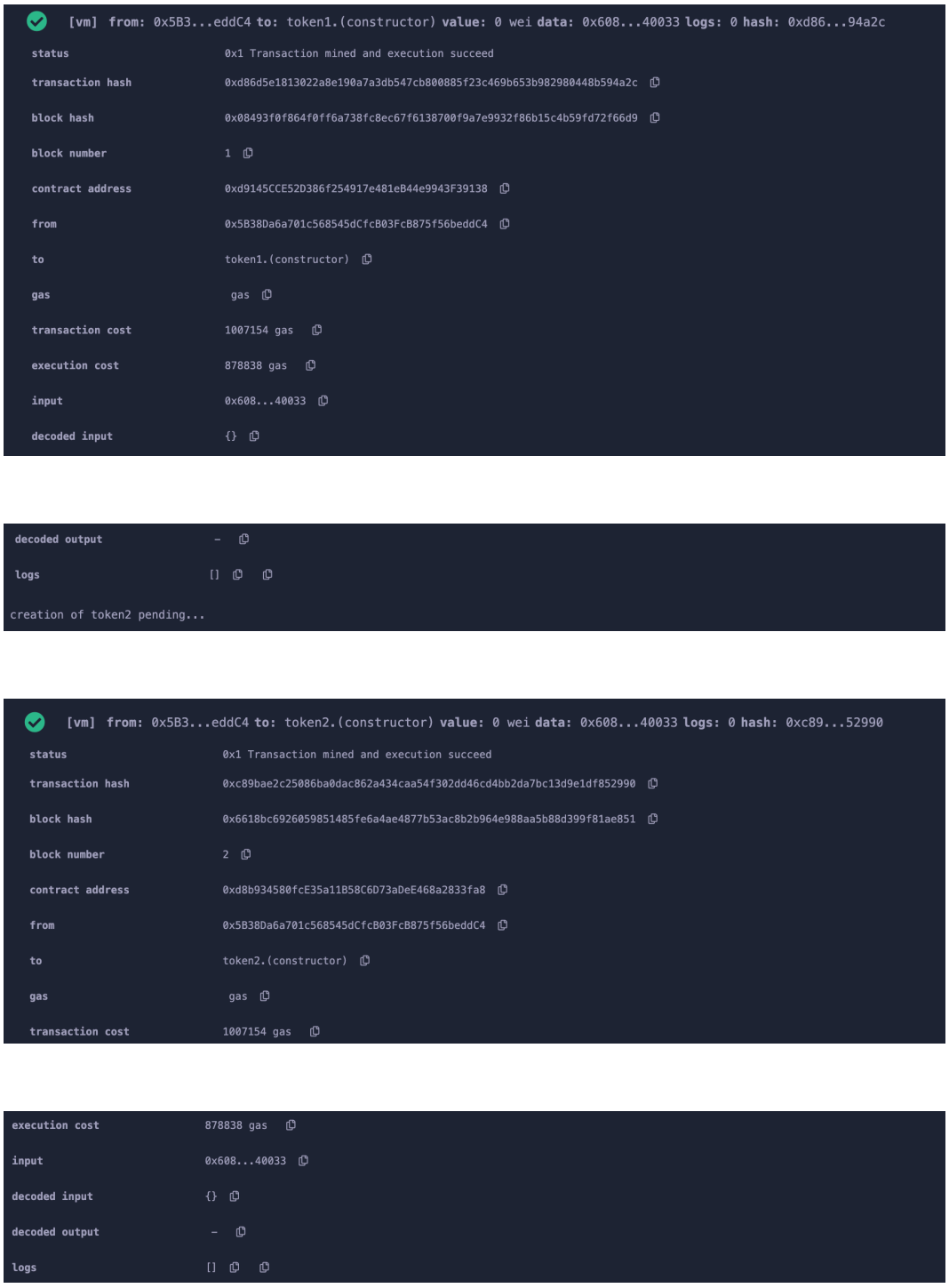
```
    // Mint new tokens and allocate them to the message sender
    function mint(uint amount) external {
        balanceOf[msg.sender] += amount;
        totalSupply += amount;
        emit Transfer(address(0),msg.sender, amount);

    }

    // Burn tokens from the message sender
    function burn(uint amount) external {
        uint accountBalance = balanceOf[msg.sender];
        require(accountBalance >= amount,"ERC20: burn amount exceeds balance");
        balanceOf[msg.sender] -= amount ;
        totalSupply -= amount;
        emit Transfer(msg.sender,address(0), amount);
    }

}
```

```
// token 2 with name comp163_2 and symbol comp2
contract token2 is IERC20 {
    // Specify the total supply of tokens
    uint public totalSupply = 100;

    // Create a mapping to keep track of balances
    mapping (address => uint) public balanceOf;

    // Create a mapping to keep track of allowances
    mapping (address => mapping(address => uint)) public allowance;

    // Specify the name of the token
    string public name = "comp163_2";

    // Specify the symbol of the token
    string public symbol = "comp2";

    // Specify the number of decimals for the token
    uint public decimals = 18;

    // Initialize the total supply and allocate all tokens to the contract creator
    constructor(){
        balanceOf[msg.sender] = totalSupply;
    }

    // Transfer tokens from the sender to the recipient, and emit the Transfer event
    function transfer(address recipient, uint amount) external returns (bool) {
        require(balanceOf[msg.sender] >= amount);
        balanceOf[msg.sender] -= amount;
        balanceOf[recipient] += amount;
        emit Transfer(msg.sender, recipient, amount);
        return true;

    }
```

```
    // Transfer tokens from the sender to the recipient, and emit the Transfer event
    function transferFrom(
        address sender,
        address recipient,
        uint amount
    ) external returns (bool) {
        require(balanceOf[sender] >= amount);
        require(allowance[sender][msg.sender] >= amount);
        balanceOf[sender] -= amount;
        allowance[sender][msg.sender] -= amount;
        balanceOf[recipient] += amount;
        emit Transfer(sender,recipient,amount);
        return true;
    }


    // Mint new tokens and allocate them to the message sender
    function mint(uint amount) external {
        balanceOf[msg.sender] += amount;
        totalSupply += amount;
        emit Transfer(address(0),msg.sender, amount);

    }

    // Burn tokens from the message sender
    function burn(uint amount) external {
        uint accountBalance = balanceOf[msg.sender];
        require(accountBalance >= amount,"ERC20: burn amount exceeds balance");
        balanceOf[msg.sender] -= amount ;
        totalSupply -= amount;
        emit Transfer(msg.sender,address(0), amount);
    }

}
```

4.2 Test ERC20 contracts by deploying it on the Remix VM

The two contracts of comp1 and comp2 are deployed on the Remix VM with the transaction overview below.



Checking the balance of comp1 and comp2 for account using ”balanceOf”. The results are shown below such that there are 100 comp1 and 100 comp2 tokens in my account.

```
CALL  [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: token1.balanceOf(address) data: 0x70a...eddc4

from                        0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                          token1.balanceOf(address) 0xd9145CCE52D386f254917e481eB44e9943F39138
execution cost              2824 gas (Cost only applies when called by a contract)
input                       0x70a...eddc4
decoded input               {
                                    "address ": "0x5B38Da6a701c568545dCfcB03FcB875f56beddC4"
                            }
decoded output              {
                                    "0": "uint256: 100"
                            }
logs                        []
```

```
CALL  [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: token2.balanceOf(address) data: 0x70a...eddc4

from                        0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                          token2.balanceOf(address) 0xd8b934580fcE35a11B58C6D73aDeE468a2833fa8
execution cost              2824 gas (Cost only applies when called by a contract)
input                       0x70a...eddc4
decoded input               {
                                    "address ": "0x5B38Da6a701c568545dCfcB03FcB875f56beddC4"
                            }
decoded output              {
                                    "0": "uint256: 100"
                            }
logs                        []
```

4.3 Full AMM contract code

```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

interface IERC20 {
    function totalSupply() external view returns (uint);

    function balanceOf(address account) external view returns (uint);

    function transfer(address recipient, uint amount) external returns (bool);

    function allowance(address owner, address spender) external view returns (uint);

    function approve(address spender, uint amount) external returns (bool);

    function transferFrom(
        address sender,
        address recipient,
        uint amount
    ) external returns (bool);

    event Transfer(address indexed from, address indexed to, uint amount);
    event Approval(address indexed owner, address indexed spender, uint amount);
}

contract CPAMM {
    // Create immutable variables for token0 and token1
    IERC20 public immutable token0;
    IERC20 public immutable token1;

    // Create variables to keep track of reserves
    uint public reserve0;
    uint public reserve1;

    // Create variables to keep track of total supply and balances
    uint public totalSupply;
    mapping(address => uint) public balanceOf;

    // Initialize the contract with token0 and token1
    constructor(address _token0, address _token1) {
        token0 = IERC20(_token0);
        token1 = IERC20(_token1);
    }
```

```
// Mint tokens to the specified address
function _mint(address _to, uint _amount) private {
    balanceOf[_to] += _amount;
    totalSupply += _amount;
}

// Burn tokens from the specified address
function _burn(address _from, uint _amount) private {
    balanceOf[_from] -= _amount;
    totalSupply -= _amount;
}

// Update the reserves
function _update(uint _reserve0, uint _reserve1) private {
    reserve0 = _reserve0;
    reserve1 = _reserve1;
}


// Swap tokens for the other token
function swap(address _tokenIn, uint _amountIn) external returns (uint amountOut) {
    // Check if the token is token0 or token1
    require(
        _tokenIn == address(token0) || _tokenIn == address(token1),
        "invalid token"
    );
    // Check if the amount in is greater than 0
    require(_amountIn> 0, "amount in = 0");

    // Check if the token is token0
    bool isToken0 = _tokenIn == address(token0);
    // If the token is token0, then set tokenIn to token0 and tokenOut to token1
    // Otherwise, set tokenIn to token1 and tokenOut to token0
    (IERC20 tokenIn, IERC20 tokenOut, uint reserveIn, uint reserveOut) = isToken0
        ? (token0, token1, reserve0, reserve1)
        : (token1, token0, reserve1, reserve0);

    // Transfer the tokens from the sender to the contract
    tokenIn.transferFrom(msg.sender, address(this),_amountIn);
```

```
    /*
    How much dy for dx?

    xy = k
    (x + dx)(y - dy) = k
    y - dy = k / (x + dx)
    y - k / (x + dx) = dy
    y - xy / (x + dx) = dy
    (yx + ydx - xy) / (x + dx) = dy
    ydx / (x + dx) = dy
    */
    // 0.3% fee

    // Calculate the amount of tokens to be sent back
    uint amountInWithFee = (_amountIn * 997) / 1000;

    amountOut = (reserveOut * amountInWithFee) / (reserveIn + amountInWithFee);

    // Transfer the tokens to the sender
    tokenOut.transfer(msg.sender,amountOut);

    // Update the reserves
    _update(token0.balanceOf(address(this)), token1.balanceOf(address(this)));
}

// Add liquidity to the pool
function addLiquidity(uint _amount0, uint _amount1) external returns (uint shares) {
    // Transfer the tokens from the sender to the contract
    token0.transferFrom(msg.sender, address(this),_amount0);
    token1.transferFrom(msg.sender, address(this),_amount1);
```

```
/*
How much dx, dy to add?

xy = k
(x + dx)(y + dy) = k'

No price change, before and after adding liquidity
x / y = (x + dx) / (y + dy)

x(y + dy) = y(x + dx)
x * dy = y * dx

x / y = dx / dy
dy = y / x * dx
*/

// Check if the reserves are greater than 0
if (reserve0 > 0 || reserve1 > 0) {
    require(reserve0 * _amount1 == reserve1 * _amount0, "x / y != dx / dy");
}
```

```
/*
How much shares to mint?

f(x, y) = value of liquidity
We will define f(x, y) = sqrt(xy)

L0 = f(x, y)
L1 = f(x + dx, y + dy)
T = total shares
s = shares to mint

Total shares should increase proportional to increase in liquidity
L1 / L0 = (T + s) / T

L1 * T = L0 * (T + s)

(L1 - L0) * T / L0 = s
*/

/*
Claim
(L1 - L0) / L0 = dx / x = dy / y

Proof
--- Equation 1 ---
(L1 - L0) / L0 = (sqrt((x + dx)(y + dy)) - sqrt(xy)) / sqrt(xy)

dx / dy = x / y so replace dy = dx * y / x

--- Equation 2 ---
Equation 1 = (sqrt(xy + 2ydx + dx^2 * y / x) - sqrt(xy)) / sqrt(xy)

Multiply by sqrt(x) / sqrt(x)
Equation 2 = (sqrt(x^2y + 2xydx + dx^2 * y) - sqrt(x^2y)) / sqrt(x^2y)
           = (sqrt(y)(sqrt(x^2 + 2xdx + dx^2) - sqrt(x^2)) / (sqrt(y)sqrt(x^2))

sqrt(y) on top and bottom cancels out
```

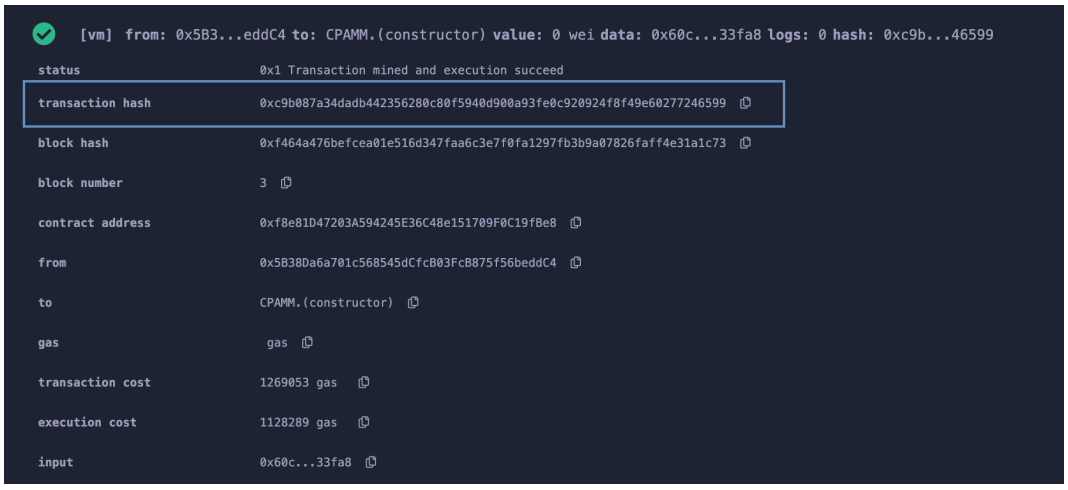
```
    --- Equation 3 ---
    Equation 2 = (sqrt(x^2 + 2xdx + dx^2) - sqrt(x^2)) / (sqrt(x^2)
    = (sqrt((x + dx)^2) - sqrt(x^2)) / sqrt(x^2)
    = ((x + dx) - x) / x
    = dx / x

    Since dx / dy = x / y,
    dx / x = dy / y

    Finally
    (L1 - L0) / L0 = dx / x = dy / y
    */

    // Calculate the number of shares to be minted
    // Hint: Use the _sqrt function on _amount0 * _amount1
    if (totalSupply == 0) {
        shares = _sqrt(_amount0*_amount1);
    } else {
        // Hint: Use the _min function
        shares = _min(
            (_amount0 * totalSupply) / reserve0,
            (_amount1 * totalSupply) / reserve1
        );
    }
    // Check if the number of shares is greater than 0
    require(shares > 0, "shares = 0");
    // Mint the shares to the sender
    _mint(msg.sender, shares);
    // Update the reserves
    _update(token0.balanceOf(address(this)), token1.balanceOf(address(this)));
}
```

```
// Remove liquidity from the pool and return the tokens to the sender
function removeLiquidity(
    uint _shares
) external returns (uint amount0, uint amount1) {
    /*
    Claim
    dx, dy = amount of liquidity to remove
    dx = s / T * x
    dy = s / T * y

    Proof
    Let's find dx, dy such that
    v / L = s / T

    where
    v = f(dx, dy) = sqrt(dxdy)
    L = total liquidity = sqrt(xy)
    s = shares
    T = total supply

    --- Equation 1 ---
    v = s / T * L
    sqrt(dxdy) = s / T * sqrt(xy)

    Amount of liquidity to remove must not change price so
    dx / dy = x / y

    replace dy = dx * y / x
    sqrt(dxdy) = sqrt(dx * dx * y / x) = dx * sqrt(y / x)

    Divide both sides of Equation 1 with sqrt(y / x)
    dx = s / T * sqrt(xy) / sqrt(y / x)
       = s / T * sqrt(x^2) = s / T * x

    Likewise
    dy = s / T * y
    */
```

```
        // Calculate the amount of tokens to be sent back
        uint bal0 = token0.balanceOf(address(this));
        uint bal1 = token1.balanceOf(address(this));


        amount0 = (_shares * bal0) / totalSupply;
        amount1 = (_shares * bal1) / totalSupply;
        require(amount0 > 0 && amount1 > 0, "amount0 or amount1 = 0");

        // Burn the shares from the sender
        _burn(msg.sender, _shares);
        // Update the reserves
        _update(bal0-amount0, bal1-amount1);

        // Transfer the tokens to the sender
        token0.transfer(msg.sender, amount0);
        token1.transfer(msg.sender, amount1);
    }

    // Calculate the square root of a number
    function _sqrt(uint y) private pure returns (uint z) {
        if (y > 3) {
            z = y;
            uint x = y / 2 + 1;
            while (x < z) {
                z = x;
                x = (y / x + x) / 2;
            }
        } else if (y != 0) {
            z = 1;
        }
    }

    // Return the smaller of two numbers
    function _min(uint x, uint y) private pure returns (uint) {
        return x <= y ? x : y;
    }
}
```

4.4 Deploying AMM on the Remix VM

Still using address1 to deploy the contract, part of the information is shown through the below figure, and it is easy to see the transaction hash of creation is ”0xc9b087a34dadb442356280c80f5940d900a93fe0c920924f8f49e60277246599”



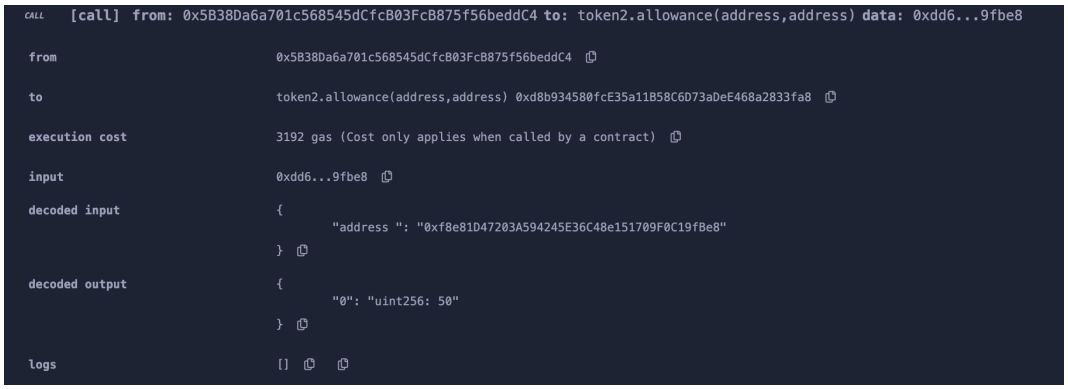
4.5 Liquidity and Swap

Approve 50 comp1

Check the allowance

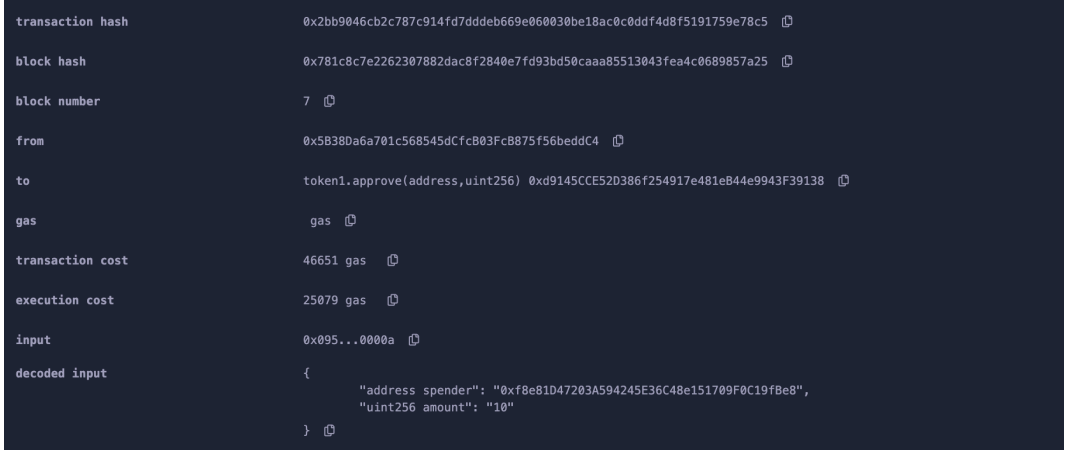
Approve 50 comp2

Check the allowance



Add liquidity

Approve another 10 comp1


transaction hash 0x2bb9046cb2c787c914fd7dddeb669e060030be18ac0c0ddf4d8f5191759e78c5
block hash 0x781c8c7e2262307882dac8f2840e7fd93bd50caaa85513043fea4c0689857a25
block number 7
from 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to token1.approve(address,uint256) 0xd9145CCE52D386f254917e481eB44e9943F39138
gas gas
transaction cost 46651 gas
execution cost 25079 gas
input 0x095...0000a
decoded input
{
"address spender": "0xf8e81D47203A594245E36C48e151709F0C19fBe8",
"uint256 amount": "10"
}


decoded output
{
"0": "bool: true"
}
logs
[
{
"from": "0xd9145CCE52D386f254917e481eB44e9943F39138",
"topic": "0x8c5be1e5ebec7d5bd14f71427d1e84f3dd0314c0f7b2291e5b200ac8c7c3b925",
"event": "Approval",
"args": {
"0": "0x5B38Da6a701c568545dCfcB03FcB875f56beddC4",
"1": "0xf8e81D47203A594245E36C48e151709F0C19fBe8",
"2": "10",
"owner": "0x5B38Da6a701c568545dCfcB03FcB875f56beddC4",
"spender": "0xf8e81D47203A594245E36C48e151709F0C19fBe8",
"value": "10"
}
}
]

Swap

Remove the liquidity

4.6 What other traditional financial agent could be replaced by the smart contract?

The AMM smart contract serves as a decentralized alternative to traditional banks, removing the need for intermediaries in borrower-lender relationships. Leveraging decentralized protocols, the AMM enables direct peer-to-peer transactions, enhancing transparency, security, and convenience. With blockchain's public and tamper-proof nature, users can engage in transparent and verifiable transactions without relying on a trustworthy third party. This decentralized approach is not only applicable to conventional lending but also extends to financial derivatives like options trading. By operating on a transparent and trustless blockchain, smart contracts provide users with direct control over their assets and transactions, fostering a more efficient and accessible financial ecosystem.