Advanced Programming - HW2

Homework 2 - Spring 2022 Semester
Deadline: Tuesday Esfand 24st - 11:59 pm

Outline

In this homework we are going to implement a simple program to simulate what is happening in **cryptocurrencies**. In this homework we are going to implement 2 classes one of them is called `Server` and the other one is the `Client`. Unlike famous cryptocurrencies we are going to use a centralized server to keep track of our clients and transactions, the clients on the other hand can use the server to transfer money to each other and most importantly mine transactions to receive rewards. Remember for this homework you will need to understand the concepts of *hashing* and *digital signatures*, for this you can use the functions provided in `crypto.cpp/h`file. **note.** You are only allowed to alter `server.cpp/h` , `client.cpp/h`, and only the debug section of `main.cpp`.

Server Class

Use the code fraction bellow to implement this class. **note.** you may need to add some keywords to these functions if necessary. other than these keywords you are not allowed to change the functions or add other functions in this class.

```
class Server
{
public:
    Server();
    std::shared_ptr<Client> add_client(std::string id);
    std::shared_ptr<Client> get_client(std::string id);
    double get_wallet(std::string id);
    bool parse_trx(std::string trx, std::string sender, std::string
receiver, double value);
    bool add_pending_trx(std::string trx, std::string signature);
    size_t mine();
private:
    std::map<std::shared_ptr<Client>, double> clients;
};
```

- **clients** This member variable will map each client to its wallet. The wallet is the amount of money the client has.
- **add_client** This function will create a new *Client* with the specified `id`. If this `id` already exists, the server should add a random 4 digit number at the end of it automatically.
  ::UPDATE:: each client should be assigned with 5 coins at the begining.
  note. do not use *srand* for your random numbers.

- **get_client** Using this function you can get a pointer to a Client using its `id`.
- **get_wallet** Using this function will return the wallet value of the client with username `id`.
- **parse_trx** Each transaction has 3 properties: **i)** id of the sender **ii)** id of the receiver **iii)** value of money to transfer. We will show each transaction with a string, concatenating each of these properties with a `-`. For example if *ali* sends *1.5* coins to *hamed* the transaction will be shown by a string `"ali-hamed-1.5"`. This function will parse this string format and outputting each property separately, if the string is not standard you should throw a runtime error.
- **add_pending_trx** Each Client can add a pending transaction using the transaction format described in the above section. Only accept a pending transaction by authenticating the sender's signature and if he has enough money in his wallet. **note.** define the below variable outside the `Server` class and save the pending transactions in it.

  ```
  std::vector<std::string> pending_trxs;
  ```

- **mine** As mentioned in the TA class each transaction has a pending state until it has been mined and to mine transactions you first need to put your pending transactions together: For example if you have 3 transactions like `"ali-hamed-1.5"`, `"mhmd-maryam-2.25"`, and `"mahi-navid-0.5"`; You will get one final string as: `"ali-hamed-1.5mhmd-maryam-2.25mahi-navid-0.5"`. We call this string the *mempool*. You will also add a number called *nonce* at the end of this string. To mine the transactions the server will generate the mempool and asks each Client for a nonce and calculates the *sha256* of the final string. For each nonce if the generated *sha256* has 4 zeros in a row in the first 10 numbers, then the mine is successful and the client who called the correct nonce will be awarded with 6.25 coins. after a successful mine of the pending transactions, all the transactions will be removed from pending and the effect of them will be applied on the clients.
  ::UPDATE:: instead of 4 zeros use 3 zeros in a row so it wont take time for your runs.
  note. after a successful mine, print the id of the miner and return the associate nonce.

Client Class

Use the code fraction bellow to implement this class. **note.** you may need to add some keywords to these functions if necessary. you are not allowed to change the functions or add other functions in this class.

```
class Client
{
public:
    Client(std::string id, const Server& server);
    std::string get_id();
    std::string get_publickey();
    double get_wallet();
    std::string sign(std::string txt);
    bool transfer_money(std::string receiver, double value);
    size_t generate_nonce();
private:
```

```
    Server const* const server;
    const std::string id;
    std::string public_key;
    std::string private_key;
};
```

- **Constructor** Creates an object of Client and assigning the specified variables using the inputs. Also generate RSA keys for the client (public and private keys).
- **get_id** Returns the Client's id.
- **get_publickey** Returns the Client's public key.
- **get_wallet** Returns the amount of money the client has.
- **:UPDATE:: sign** signs the input with the private key and returns the signature.
- **transfer_money** Creates a transaction in the server according to its inputs. To create a transaction use the specified string format described in above sections and sign the final transaction string with your private key. use both your signature and your transaction signature to create a pending transaction in the Server using *add_pending_trx* function.
- **generate_nonce** Returns a random number as a nonce so the server uses it for mining.

Questions

- As mentioned above, you will need to define `std::vector<std::string> pending_trxs;` outside of the Server class in order to keep track of your pending transactions. Add the following function in main.cpp and make arrangements so that it will print the pending transactions correctly.

```
void  show_pending_transactions()
{
    std::cout  <<  std::string(20, '*') <<  std::endl;
    for(const  auto& trx : pending_trxs)
        std::cout << trx <<  std::endl;
    std::cout  <<  std::string(20, '*') <<  std::endl;
}
```

- The variable `std::map<std::shared_ptr<Client>, double> clients;` is a private member of Server class. define a function outside of Server class like bellow and make the function work properly.

```
void  show_wallets(const  Server& server)
{
    std::cout << std::string(20, '*') << std::endl;
    for(const auto& client: server.clients)
```

```
        std::cout << client.first->get_id() <<  " : "  <<
client.second << std::endl;
    std::cout << std::string(20, '*') << std::endl;
}
```

Hint

Instructions to use functions in `crypto.cpp/h` file.

- To calculate the *sha256* of a string you can use the `crypto.cpp/h` function as:

```
std::string hash{crypto::sha256("hi")}
```

- To generate RSA key pairs do:

```
std::string public_key{}, private_key{};
crypto::generate_key(public_key, private_key);
```

- To sign and authenticate a string with your RSA keys, use:

```
std::string signature = crypto::signMessage(private_key, "my data");
bool authentic = crypto::verifySignature(public_key, "my data",
signature);
```

Finally

As mentioned before, do not alter other files already populated except otherwise indicated. In case you want to test your code you may use the `debug` section of the `main.cpp`.

```
if (true) // make false to run unit tests
{
    // debug section
}
else
{
    ::testing::InitGoogleTest(&argc, argv);
    std::cout << "RUNNING TESTS ..." << std::endl;
    int ret{RUN_ALL_TESTS()};
    if (!ret)
        std::cout << "<<<SUCCESS>>>" << std::endl;
    else
      std::cout << "FAILED" << std::endl;
```

```
    }
    return 0;
```

GOOD LUCK