BlockChain

Сатановский Артем, 331

6 декабря 2020 г.

1 Вступление

В рамках курса была реализована программа blockchain на языке C++. Также рассмотрены основные моменты работы и особенности технологии: сама цепь, синхронизация цепей, умные контракты и тд.

1.1 class Block

В классе блока есть 5 основных полей, они составляют необходимый минимум: имя, сообщение, чило, хэш, время создания блока.

```
struct Block {
private:
    string name;
    string message;
    uint64_t nonce{};
    uint64_t hash{};
    typedef boost::posix_time::ptime time;
    time time_stamp{};
//...
```

Далее – интерфейс работы: получение сообщения, имени, числа, хэша, времени создания, вставка хэша. Необходимые функции, конструкторы и операторы. Считаем, что хэш следующего блока есть сумма числа, имени, времени создания и хэша от предыдущего блока. Функция $getHash$ реализована при помощи методов библиотеки boost.

```
//...
public:
    /*
     * Конструкторы и Деструкторы
     */
    Block();
    ~Block() = default;
    explicit Block(const string& name, const string& message, uint64_t hash, uint64_t nonce);
    explicit Block(const string& name, const string& message, uint64_t nonce, time time_stamp);
    /*
     * Интерфейс блока
     */
    string get_message();
    string get_name();
    uint64_t get_nonce();
    uint64_t get_hash();
```

```
    time get_time_stamp();

    void insert_hash(int block_hash);
    //Вернуть всю информацию этого блока
    tuple<string,string,uint64_t,uint64_t,time> get_all(){return make_tuple(message,name,nonce,hash,time_sta

    /*
     * Оператор вывода всей информации блока
     */
    friend ostream& operator<<(ostream &out,const Block &block);


};
/*
 * Получение хэша
 */
size_t getHash( Block& block);
```

1.2 class BlockChain

РАссмотрим класс цепи. Было решено, что оптимальным способом хранения блоков является list. Его сложно перезаписывать, если он вырождается в длинную цепочку. Присоединять в хвост новые блоки и менять таким образом длину цепи достаточно легко. Таким образом, имеем класс:

```
class BlockChain {
private:
    list<Block> block_chain;
//...
```

И, соответственно, интерфейс работы. Необхоимые конструкторы, работа с файлами и вспомогательные функции.

```
    /*
     *Конструкторы и Деструкторы
    */
    BlockChain() = default;
    ~BlockChain() = default;
    explicit BlockChain(Block& block);
    /*
     * Интерфейс
    */
    list<Block> push(Block& newTail); //Добавление блока в конец "этой" цепи
    Block& get_block(int index); // Получение блока по индексу
    Block& pop(); //Удаление последнего блока из "этой" цепи, возвращает остаток цепочки
    unsigned int print_last_messages(int n); //Печать последних n сообщений "этой" цепи
    int get_chain_length(); // Длина "этой" цепи
    /*
     * Работа с файлами
     */
    unsigned int save_to_file(const string &file_name); // Сохранение в файл цепи(цепей)
    unsigned int load_from_file(const string &file_name); // Загрузка цепи(цепей) из файла
    /*
     * Операторы
     */
```

```
    Block& operator[](int index);  //Доступ по индексу в цепи
    friend ostream& operator<<(ostream &out, const BlockChain &chain);  // Вывод всей цепи

    /*
     * Синхронизация блоков
     */
    BlockChain sendData(int index);

    void loadData(BlockChain chain, int index);

    int is_block_in_chain(Block &block);
};
bool  sync(BlockChain &chain1, BlockChain &chain2); // Синхронизация двух цепей с учетом правильности
```

1.3 Синхронизация двух цепей функцией *sync*

Достаточно важная задача – синхронизация двух цепей. Функция sync выполняет эту задачу следующим образом: Предположим, что *Alice* и *Bob* хотят синхронизировать свои цепи. Для начала нужно сравнить длины. Не умаляя общности, у *Alice* длина больше, тогда нужно записать 'хвост' , который есть у Alice в цепь $Bob'a$. Нам нужно понять, с какого момента начинается этот хвост. При помощи метода $is\_block\_in\_chain$ находим эту позицию. Далее – тривиально. 'Цепляем хвост' *Alice*, начиная с этой позиции, к концу цепи $Bob'a$. Такимм образом, получаем, что цепи синхронизированы. Второй случай аналогичный.

2 Умные контракты

В рамках курса были рассмотрены смарт-контракты.

2.1 Как это работает?

Допустим кто-то хочет произвести транзакцию. Для начала эта транзакция пересылается в систему, состоящую из равноправных компьютерных узлов (участников сети), для ее подтверждения. После подтверждения эта транзация объединяется с другими транзациями, создается блок. Затем этот блок записывается в блокчейн на уникальное место, которое нельзя изменить. Происходит синхронизация цепей между участниками сети. Транзакция завершена.
Например, транзакция может быть переводом средств с одного кошелька на другой.

2.2 Приемущества

Из приемуществ смарт-контрактов можно выделить следующее:

- Скорость
- Независимость
- Надежность
- Нет посредников
- Нет ошибок

2.3 Недостатки

Из недостатков смарт-контрактов можно выделить следующее:

- Сложность в реализации
- Отсутствие регулирования
- Невозможность изменения

2.4 Где используется?

Смарт-контракты могут использоваться или уже используются здесь:

- Финансы
- Выборы
- Аудит и налогообложение
- Страхование

3 Немного о электронных цифровых подписях (ЭЦП)

Цифровые подписи играют важную роль в системе. С ее помощью можно утверждать, что информация внутри блока является точной, не поддельной. По сути, цифровая подпись – это последовательность байтов, формируемая путем преобразования подписываемой информации по криптографическому алгоритму и предназначенная для проверки авторства электронного документа.
ЭЦП основывается на методах асимметричного шифрования и хэш-функциях. Одним из таких может быть алгоритм RSA. Участники сети могут проверить с помощью открытого ключа, что именно владелец блока внес изменения и подписал блок своей подписью. Соответственно, закрытый ключ находится у владельца.

3.1 Ключи

Как мы знаем, есть открытые и закрытые ключи.
Они связаны между собой. Открытй ключ генерируется из закрытого.
Закрытый ключ находится у пользователя и используется для доступа в блок (к кошельку, например).

3.2 Как происходит подписание документа

Для этого потребуется алгоритм шифрования (RSA), хэш-функция ($sha256/sha512$) и информация для подписи. При помощи RSA гинерируется публичный и закрытый ключи. Далее от подписываемой информации берется хэш. Затем полученный хэш и закрытый ключ подставляем в функцию асимметричного шифрования RSA получаем строку ЭЦП. Вставляем новую транзацию с подписью и данными в созданный блок.

github