区块链期末 project 项目合约部署报告

陈铭涛

16340024

合约简述

部署的智能合约为一个区块链加密聊天的智能合约。合约部分的主要作用为为当前的以太坊账户创建应用用户，向其他已创建的用户发送信息，向其他用户发送以太币等。由于应用涉及计算量较大的加密与签名部分执行的位置在浏览器前端，合约部分所涉及的代码逻辑就较为简单。

合约结构简述

合约中包含了一个结构体，定义了每个用户所包含的数据：

```
struct User {
    string username;
    address addr;
    string[] recv_messages;
    string[] sent_messages;
    string pubKey;
    bool exists;
    string signature;
}
```

各数据字段作用如下：

- Username：字符串，为用户的用户名，便于标记各用户的地址，也可便于其他用户快速查找其他用户。

- addr：用户的以太坊地址
- recv_messages：用户收到的消息数据，每个消息为一个经过 RSA 加密并签名的 JSON 字符串，包含了消息的 Hex 编码数据和发送者对该数据通过 web3 API 完成的数字签名。（由于区块链上存储数据 gas 较高，后续可能会改为使用 ipfs 等其他方式存储此类消息数据）
- sent_messages：用户发送的消息数据，结构与收到的消息相同，包含发送的消息数据和用户对数据的数字签名。需要分别保存发送与接收数据的原因是发送数据时使用接收方公钥加密，发送方无法重新读取发送的数据明文，因此需要
- pubKey 为用户的 RSA 公钥，其他用户向用户发送数据时使用该公钥进行加密，用户接收数据后使用自行保存的 RSA 私钥进行解密。
- exists 为代表用户存在的布尔型变量，用与从用户地址的 mapping 查找时检查用户是否存在，当用户创建时设为 true。
- signature 为创建用户时对用户名，地址和公钥使用 Web3 API 获得的数字签名，用于验证用户信息的正确性。

合约中的数据成员如下：

```
mapping(address => User) users;
```

以太坊地址到用户的映射.

```
mapping(string => address) name_addr;
```

用户名到对应用户的以太坊地址的映射。

合约中的函数如下：

```
function checkNewUser(string name, string pubKey) public view
returns (uint)
```

用于在创建用户前检查用户名和公钥是否符合要求，根据情况可能返回用户名过长、公钥过长、用户名存在重复、该地址已存在用户的 4 中错误码，若可以创建用户则错误码为 0.

```
function addUser(string name, string pubKey, string sign) public
```

创建用户，需传入用户名，公钥以及用户信息的数字签名。

```
function getAddrFromName(string username) public view
returns (address)
```

根据传入的用户名返回对应的以太坊地址。

```
function getUsername(address target) public view returns
(string memory)
```

根据传入的以太坊地址返回对应的用户名。

```
function sendEther(address target) public payable
```

用户调用该函数时向合约转入一定 value 的以太币，合约将会将币转至传入的以太坊地址中。

```
function hasUser(address target) public view returns (bool)
```

检查传入的地址是否存在对应的已注册用户

```
function getCurrentAddr() public view returns (address)
```

返回当前用户的以太坊地址

```
function sendMessage(address targetUser, string memory
recv_msg, string memory send_msg) public
```

传入发送消息的接收用户地址，对方接收到的消息密文和本方用户记录下的发送的消息密文。

```
function getSentMsgCount() public view returns (uint)
```

返回用户发送的消息总数。

```
function getUserSentMsg(uint index) public view returns
(string)
```

根据输入的索引返回对应的用户发送的信息。

```
function getRecvMsgCount(address userAddr) public view
returns (uint)
```

返回传入的地址的对应用户接收的信息总数。

```
function getUserRecvMsg(address userAddr, uint index) public
view returns (string)
```

根据输入的地址和索引返回用户接收的对应的消息数据。

合约部署测试效果

合约在 Remix 下进行编写并部署在私链上，部署后从 geth 访问测试的步骤如下：

1. 从 remix 中复制出 abi 信息，转为 JSON 字符串后拷贝到 geth 中获得 abi 对象用于在 geth 中调用合约：

```
> var abi = '[{"constant":false,"inputs":[{"name":"name","type":"string"},{"name":"pubKey","type":"str
ing"},{"name":"sign","type":"string"}],"name":"addUser","outputs":[],"payable":false,"stateMutability"
:"nonpayable","type":"function"},{"constant":false,"inputs":[{"name":"target","type":"address"}],"name
":"sendEther","outputs":[],"payable":true,"stateMutability":"payable","type":"function"},{"constant":f
alse,"inputs":[{"name":"targetUser","type":"address"},{"name":"recv_msg","type":"string"},{"name":"sen
d_msg","type":"string"}],"name":"sendMessage","outputs":[],"payable":false,"stateMutability":"nonpayab
le","type":"function"},{"constant":true,"inputs":[{"name":"name","type":"string"},{"name":"pubKey","ty
pe":"string"}],"name":"checkNewUser","outputs":[{"name":"","type":"uint256"}],"payable":false,"stateMu
tability":"view","type":"function"},{"constant":true,"inputs":[{"name":"username","type":"string"}],"n
ame":"getAddrFromName","outputs":[{"name":"","type":"address"}],"payable":false,"stateMutability":"vie
w","type":"function"},{"constant":true,"inputs":[],"name":"getCurrentAddr","outputs":[{"name":"","type
":"address"}],"payable":false,"stateMutability":"view","type":"function"},{"constant":true,"inputs":[{
"name":"userAddr","type":"address"}],"name":"getRecvMsgCount","outputs":[{"name":"","type":"uint256"}]
,"payable":false,"stateMutability":"view","type":"function"},{"constant":true,"inputs":[],"name":"getS
entMsgCount","outputs":[{"name":"","type":"uint256"}],"payable":false,"stateMutability":"view","type":
"function"},{"constant":true,"inputs":[{"name":"target","type":"address"}],"name":"getUsername","outpu
ts":[{"name":"","type":"string"}],"payable":false,"stateMutability":"view","type":"function"},{"consta
nt":true,"inputs":[{"name":"userAddr","type":"address"},{"name":"index","type":"uint256"}],"name":"get
UserRecvMsg","outputs":[{"name":"","type":"string"}],"payable":false,"stateMutability":"view","type":"
function"},{"constant":true,"inputs":[{"name":"index","type":"uint256"}],"name":"getUserSentMsg","outp
uts":[{"name":"","type":"string"}],"payable":false,"stateMutability":"view","type":"function"},{"const
ant":true,"inputs":[{"name":"target","type":"address"}],"name":"hasUser","outputs":[{"name":"","type":
"bool"}],"payable":false,"stateMutability":"view","type":"function"}]'
undefined
> abi = JSON.parse(abi)
[{
    constant: false,
```

2. 从 abi 对象和在 remix 中获得的合约部署地址构建可调用的合约实例：

```
> contract = web3.eth.contract(abi).at('0x752d298fe0757777489e02f8816187a1bd307e63')
{
  abi: [{
      constant: false,
      inputs: [{...}, {...}, {...}],
      name: "addUser",
      outputs: [],
      payable: false,
      stateMutability: "nonpayable",
      type: "function"
  }, {
      constant: false,
      inputs: [{...}],
      name: "sendEther",
      outputs: [],
      payable: true,
```

3. 用于测试的账户信息如下：

```
> var acc1 = eth.accounts[0]
undefined
> var acc2 = eth.accounts[1]
undefined
```

```
> eth.getBalance(acc1)
549999370000000000
> eth.getBalance(acc2)
102000000000000000000
```

4. 添加用户使用的公钥与用户名如下：

```
> var pubKey1="MIGeMA0GCSqGSIb3DQEBAQUAA4GMADCBiAKBgFZiFTAgVxXAhmqCGqMS4SpkHZX07X80bfcIlR2cZEmFBqTB6Kg
G/scsqJRw/nB1+7Ot2QaktQP+5eKPw/Vsvb5YOt9pP8rdfassV4BS+hup+3hLxvSWOEQDN19/PIu2rNhp9oSO3V8njm7iU1kyyMQWG
nWwiTJWbLkHCBJCBDPnAgMBAAE="
undefined
> var pubKey2="MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCMmXDrv4JRTlCCJMejx7Vkc1G1eQ4vyYS1IRVMYWdfgZEWwC1
Xz01/ak8fWEwFRREL01ubvomo3YFhHeDBIXS+WjAf2Mr6g+8lE9lIJUkQ9QGVZDlS7E0/WsZSYAN9LivuNOWfPTuMWKaswzCdFvzdn
w9vuiFawGgcU886mHaq9QIDAQAB"
undefined
> var username1 = "aaa"
undefined
> var username2 = "bbb"
undefined
>
```

5. 测试检查并添加用户 1：

```
> contract.checkNewUser(username1, pubKey1)
0
> sign1 = web3.eth.sign(acc1, username1 + pubKey1)
"0xf3a2d54180923dca67866e1d02257d436265e89edb98d5fcd3033f6eb19db3591ccd1beb6292238a5343ffb388488303acf5c9720a52b12900d53
585e9e1311d01"
> contract.addUser(username1, pubKey1, sign1, {from: acc1, gas: 1000000})
"0xfcd4133d1ea8e5cf8c02abc0ce6073135e7d85bdd0b5f166452a217165689c75"
>
```

6. 测试检查并添加用户 2：

```
> contract.checkNewUser(username1, pubKey1)
1
> contract.checkNewUser(username1, pubKey1, {from: acc2})
4
> sign2 = web3.eth.sign(acc1, username2 + pubKey2)
"0x03a8a3c0baa396cdd37e417bedc522d45ccd7251c676e0e01f189548046e88545672c58a47f4bc99e080b5361b657169c1eb023ca0fd4f103e3ba
3c5aba9291d01"
> contract.checkNewUser(username2, pubKey2, {from: acc2})
0
> contract.addUser(username2, pubKey2, sign2, {from: acc2, gas: 1000000})
"0xc88a5c8f0631e7f57feb7235e62dbe135f4cca87f3b6198984acc423be1ee3a1"
>
```

7. 测试从用户 1 向用户 2 发送以太币：

```
> eth.getBalance(acc1)
51997781872000000000
> eth.getBalance(acc2)
10499915292800000000
> contract.sendEther(acc2, {from: acc1, value: web3.toWei(3, "ether")})
"0x0c32cb0da9130fe83588d88ea80c48e15a85f6879c3657882783ebf9bf70bad9"
> eth.getBalance(acc1)
48997719584000000000
> eth.getBalance(acc2)
10799915292800000000
```

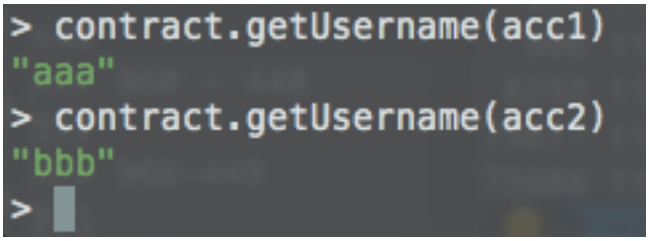
8. 测试从用户名获取以太坊地址：

```
> contract.getAddrFromName('aaa') === acc1
true
> contract.getAddrFromName('bbb') === acc2
true
>
```

9. 测试从用户 2 发送消息至用户 1 并查看消息：

```
> sent_msg = "0x0000123123"
"0x0000123123"
> recv_msg = "0x1231230000"
"0x1231230000"
> contract.sendMessage(acc1, recv_msg, sent_msg, {from: acc2, gas: 500000})
"0x1194de0a34b3ce0fd3ba9de773c77065bb83fb2409594f9da9ca0e1ffde78d14"
> contract.getSentMsgCount({from: acc1})
0
> contract.getSentMsgCount({from: acc2})
1
> contract.getUserSentMsg(0, {from:acc2})
"0x0000123123"
> contract.getRecvMsgCount(acc1)
1
> contract.getRecvMsgCount(acc2)
0
> contract.getRecvMsgCount(acc1, 0)
0
> contract.getUserRecvMsg(acc1, 0)
"0x1231230000"
>
```

10. 测试从以太坊地址获取用户名：



智能合约源代码

CryptoChat.sol

```
pragma solidity ^0.4.24;

contract CryptoChat {

    struct User {
        string username;
        address addr;
        // Messages stores with a JSON format, containing the
RSA cipher
        // text and the digital sign from the sender.
        string[] recv_messages;
        // Since the cipher for receiver and sender is
different,
        // a message has to be stored in two ciphers
seperately.
        string[] sent_messages;
        string pubKey;
        bool exists;
        string signature;
    }
    mapping(address => User) users;
    mapping(string => address) name_addr;

    function getRecvMsgCount(address userAddr) public view
returns (uint){
        require(users[userAddr].exists);
        return users[userAddr].recv_messages.length;
    }
```

```
    function getUserRecvMsg(address userAddr, uint index)
public view returns (string) {
        require(users[userAddr].exists &&
users[userAddr].recv_messages.length > index);
        return users[userAddr].recv_messages[index];
    }

    function getSentMsgCount() public view returns (uint){
        require(users[msg.sender].exists);
        return users[msg.sender].sent_messages.length;
    }

    function getUserSentMsg(uint index) public view returns
(string) {
        require(users[msg.sender].exists &&
users[msg.sender].sent_messages.length > index);
        return users[msg.sender].sent_messages[index];
    }

    function sendMessage(address targetUser, string memory
recv_msg, string memory send_msg) public{
        require(users[targetUser].exists &&
users[msg.sender].exists);
        users[targetUser].recv_messages.push(recv_msg);
        users[msg.sender].sent_messages.push(send_msg);
    }

    function getCurrentAddr() public view returns (address) {
        return msg.sender;
    }

    function hasUser(address target) public view returns
(bool) {
        return users[target].exists;
    }

    // Only add user when this function returns 0.
    function checkNewUser(string name, string pubKey) public
view returns (uint) {
        if (users[msg.sender].exists) {
            //1 - represent user already exists.
            return 1;
        }
        if (bytes(name).length > 20) {
```

```
            //2 - represent username too long
            return 2;
        }
        if (bytes(pubKey).length > 300) {
            //3 - represent the pub
            return 3;
        }
        if (name_addr[name] != 0x0) {
            //4 - username already exists.
            return 4;
        }
        //OK
        return 0;
    }

    //Returns ERROR code to represent what kinds of error were
encountered.
    function addUser(string name, string pubKey, string sign)
public  {
        require(!users[msg.sender].exists &&
bytes(name).length <= 20 && bytes(pubKey).length <= 300 &&
name_addr[name] == 0x0);
        users[msg.sender] = User({
            username: name,
            addr: msg.sender,
            pubKey: pubKey,
            recv_messages: new string[](0),
            sent_messages: new string[](0),
            exists: true,
            signature: sign
        });
        name_addr[name] = msg.sender;
    }

    function getAddrFromName(string username) public view
returns (address) {
        return name_addr[username];
    }

    function sendEther(address target) public payable {
        require(users[target].exists &&
users[msg.sender].exists);
        target.transfer(msg.value);
    }
```

```
    function getUsername(address target) public view returns
(string memory) {
        require(users[target].exists);
        return users[target].username;
    }


}
```