Version	Description of Change	Author	Date
0.1	Draft	Richard	

CONTENTS

Summary

The smart contract design as specified at this document is based on following requirement:

Hi Richard,

We are sending a test out to our applicants to see if they have solid foundation in smart contracts. If you are interested in working with us please take a look atthe below problem.

"Create a smart contract that facilitates the storage of messages on-chain. The smart contract owner should be able to authorize new addresses to store messages on the chain. Each newly added user will have a certain credit amount that they are assigned on add. From there each time they add or remove a message they should lose a credit.
Users who are authorized should be able to purchase extra credits by sending funds to the smart contract. If a user is done they should be able to withdrawal any remaining credit funds.

There will be 2 smart contracts created based on this requirement, those two smart contracts are for “demo” only, there still have lots of space to improve (such as separate business logic from messaging contract to a library contract etc)

In order to build secure and robust contracts, Openzeppelin and best practices are followed at those two smart contracts (such as: safemath to avoid overflow, ReentrancyGuard to avoid the re-entrancy issue etc).

Credit token contract (ALAKEMI)

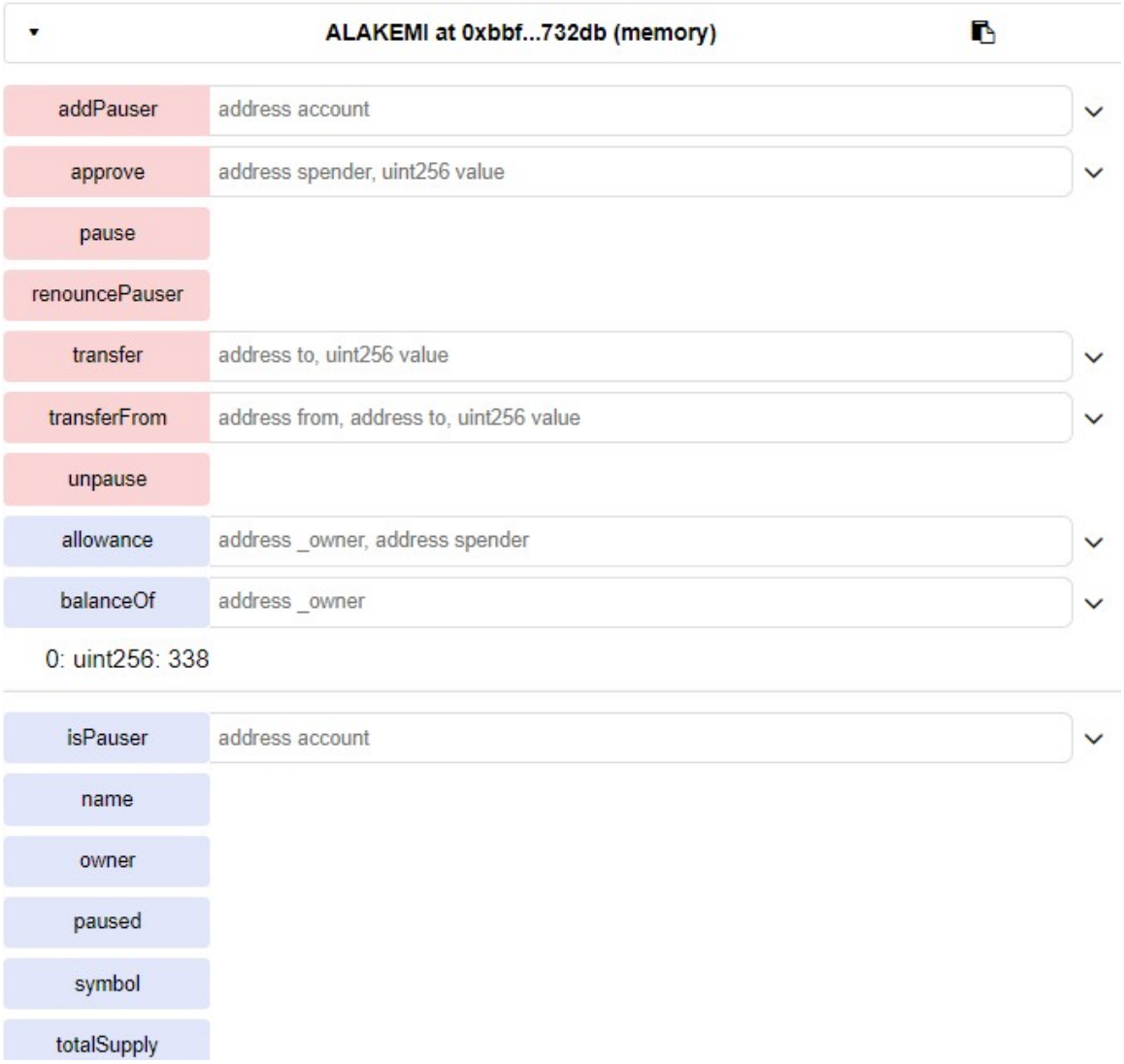
This is an ERC20 token with name as “ALAKEMI”. This smart contract will be used as “Credits” as specified at the requirement. And it implemented all interfaces as defined at ERC20 standard.

The specific wording of those function is followed by a clarification of what it does, in square brackets.

1. totalSupply [Get the total token supply]
2. balanceOf [Get the account balance of another account with address _owner]
3. transfer [Send _value amount of tokens to address _to]
4. transferFrom [Send _value amount of tokens from address _from to address _to]
5. approve [Allow _spender to withdraw from your account, multiple times, up to the _value amount. If this function is called again it overwrites the current allowance with _value]
6. allowance [Returns the amount which _spender is still allowed to withdraw from _owner]

Events format:

1. Transfer [Triggered when tokens are transferred.]
2. Approval


ALAKEMI at 0xbbf...732db (memory)
addPauser
address account
approve
address spender, uint256 value
pause
renouncePauser
transfer
address to, uint256 value
transferFrom
address from, address to, uint256 value
unpause
allowance
address _owner, address spender
balanceOf
address _owner
0: uint256: 338
isPauser
address account
name
owner
paused
symbol
totalSupply

Messaging contract (AlkemiMessaging)

The main functions of this contract are:

- Transfer credit tokens between this smart contract(messaging contract) and credit token (ERC20) contract
- (Owner Only) Authorized new player and assign credits to new player
- Create new message by authorized players, certain amount of tokens will be withdraw from current player account after the message adding
- Remove existing message. Only message owner can remove the message and certain amount of credits will be withdraw from current player account after the message removing
- Authorized players has the ability to buy credits using Ether
- Function to Check player details. Such as: player credit balance, is Authorized?
- Function to Check global credit status. Such as: total number of authorized player, total assigned credits, total raised credits, total raised Ether, total remaining credits etc.
- Check message details based on message ID. Such as: message creation date/time, message creator, message content and remove flag etc

Other Functions are shared by above two contracts:

- Pause - Only contract owner has the ability to pause the contract. After the contract is paused, all transactions will be denied (such as those smart contracts are in maintenance, we need pause them)
- Unpause – Contract owner has the ability to unpause the contract
- AddPausr – Contract owner has the ability to add another address as contract administrator so that those address also have the ability to pause/unpause contract

AlkemiMessaging at 0xdc5...79154 (memory)

(fallback)	
addPauser	address account
authorizeNewPlayer	address _newplayer
buyCredits	
createMessage	string _messagecontent
depositCredits	uint256 _amount
kill	
pause	
removeMessage	uint256 _messageID
renouncePauser	
unpause	
withdrawCredits	uint256 _amount
COST	
CREDITPRICE	
CREDITSONAUTHORIZED	
getCreditStatus	
getMessageInfo	uint256 _messageid
getPlayerInfo	address _player
isPauser	address account

Main functions at the messaging contract "ALKEMIMESSAGING"

Authorize New Player

```
authorizeNewPlayer(address _newplayer) external onlyAdministrators nonReentrant
isHuman whenNotPaused payable {
```

Input(s)

_newplayer

New player address

Return value

None

Function Details

- Modifier
  - Only contract owner can authorized a new player
  - Any call from any smart contract will be denied to avoid any attack from those smart contract
  - Use "ReentrancyGuard" and make sure Reentrancy not happens
  - If the contract is paused by owner, this function call will be denied

- Validation checking
  - New player address cannot be blank
  - The specified new player address is not authorized yet
  - Make sure there is enough credit tokens in order to assign credits to new player

- Transfer assigned credits from owner account to new player account
- Update global credit status: assignedCreditTokens , authorizedPlayers and remainingCreditTokens
- Update player authorization flag

```
//Authorized a new player (Oener only / Not allow Reentrancy / block contract call / Contract is not paused)
function authorizeNewPlayer(address _newplayer) external onlyAdministrators nonReentrant isHuman whenNotPaused payable {
      require(_newplayer != address(0),"Please specify new player address");
      require(player[_newplayer].isAuthorized == 0,"This player already was authorized");
      require(creditStatus.remainingCreditTokens > CREDITSONAUTHORIZED ,"Run out of credits, Please contact owner");
      require(player[owner].creditBalance >= CREDITSONAUTHORIZED , "Owner don't have enough balance");

      //Transfer credit tokens from owner account to new player account at this contract
      player[owner].creditBalance = player[owner].creditBalance.sub(CREDITSONAUTHORIZED);
      player[_newplayer].creditBalance = player[_newplayer].creditBalance.add(CREDITSONAUTHORIZED);
      //Update global credit status
      creditStatus.remainingCreditTokens = creditStatus.remainingCreditTokens.sub(CREDITSONAUTHORIZED);
      creditStatus.assignedCreditTokens = creditStatus.assignedCreditTokens.add(CREDITSONAUTHORIZED);
      creditStatus.authorizedPlayers++;

      player[_newplayer].isAuthorized = 1;   //Assign 1 to authorized new player.

      emit evt_AuthorizedNewPlayer(msg.sender,_newplayer);
}
```

Add new message

```
function createMessage(string _messagecontent) external whenNotPaused isHuman nonReentrant returns (uint256)
```

Input(s)

_messagecount

Message content

Return value

After new message is created, it generates a message ID for that new message. That New added message ID will be returned

Function Details

- Modifier
  - Any call from any smart contract will be denied to avoid any attack from those smart contract
  - Use “ReentrancyGuard” and make sure Reentrancy not happens
  - If the contract is paused by owner, this function call will be denied
- Validation checking
  - Only authorized player can add a new message
  - Have enough credit tokens in order to add a new message

- Withdraw certain amount of credits from current player account
- Update credit global status: arisedCredits
- Record new message to the chain: messageID, creation date/time, creator and message content

```
}

/* ========Messaging facilitates \ Create new Message (Not allow Reentrancy / block contract call / Contract is not paused)=====
function createMessage(string _messagecontent) external whenNotPaused isHuman nonReentrant returns (uint256) {
    require(player[msg.sender].isAuthorized == 1,"This player was not authorized yet");
    require(player[msg.sender].creditBalance >= COST  ,"You don't have enough credit tokens to add a new message");

    messageCount++;

    player[msg.sender].creditBalance = player[msg.sender].creditBalance.sub(COST);   //Pay the credit before adding the message

    creditStatus.arisedCredits = creditStatus.arisedCredits.add(COST);  //Update global credit status

    //Record added message info
    messageinfo[messageCount].CreatDate = now;
    messageinfo[messageCount].Creator = msg.sender;
    messageinfo[messageCount].MessageContent =_messagecontent;

    emit evt_MessageAction(msg.sender, messageCount, "Add");
    return messageCount;   //return added message ID
}
```

Remove message

```
removeMessage(uint256 _messageID) external whenNotPaused isHuman nonReentrant returns (bool)
```

Input(s)

_ messageID

Message ID

Return value

true or false

Function Details

- Modifier
  - Any call from any smart contract will be denied to avoid any attack from those smart contract
  - Use “ReentrancyGuard” and make sure Reentrancy not happens
  - If the contract is paused by owner, this function call will be denied
- Validation checking
  - Make sure the specified message is NOT removed yet
  - Make sure current player is the specified message creator (only message creator can remove their own message)
  - Make sure the input message ID is a valid ID
  - Make sure current player is authorized
  - Have enough credit tokens in order to add a new message

- Withdraw certain amount of credits from current player account
- Update credit global status: arisedCredits
- Update the specified message and leave the remove flag

```
/* ========Messaging facilitates \ Remove an existing Message(Not allow Reentrancy / block contract call / Contract is not paused
function removeMessage(uint256 _messageID) external whenNotPaused isHuman nonReentrant returns (bool) {
    require(_messageID <= messageCount,"Invalid Message ID");
    require(player[msg.sender].isAuthorized == 1,"This player was not authorized yet");
    require(messageinfo[_messageID].isRemoved == false,"This message already removed");
    require(player[msg.sender].creditBalance >= COST  ,"You don't have no enough credit tokens to remove a new message");
    require(messageinfo[messageCount].Creator == msg.sender ,"Only message creator can remove this message");

    player[msg.sender].creditBalance = player[msg.sender].creditBalance.sub(COST);     //Pay the credit before adding the message

    creditStatus.arisedCredits = creditStatus.arisedCredits.add(COST); //Update global credit status

    messageinfo[messageCount].isRemoved = true;   //Leave message remove flag

    emit evt_MessageAction(msg.sender, _messageID, "Remove");
    return true;
}
```

Deposit Credits

```
depositCredits(uint _amount) external whenNotPaused nonReentrant payable returns (bool)
```

Input(s)

_amount

Return value

true or false

Function Details

- Transfer specified credit amount for current address from ERC20 credit smart contract to current smart contract

```
/* ========Credit tokens Operation \ Deposit credit token from ERC20 contract to this contract========================== */
function depositCredits(uint _amount) external whenNotPaused nonReentrant payable returns (bool){
        require(creditTokens.transferFrom(msg.sender, address(this), _amount) == true); //Transfer credit tokens from current player a
        player[msg.sender].creditBalance = player[msg.sender].creditBalance.add(_amount);  //Record the transfered amounts to current
        emit  evt_TokenAction(msg.sender,_amount, "Deposit");
        return true;
}
```

Withdraw Credits

```
withdrawCredits(uint _amount) external whenNotPaused nonReentrant payable returns (bool)
```

Input(s)

_amount

Return value

true or false

Function Details

- Transfer specified credit amount for current address from messaging smart contract to ERC20 credit smart contract.

```
/* ========Credit tokens Operation \ Withdraw credit token from current contract to ERC20 contract========================== */
function withdrawCredits(uint _amount) external whenNotPaused nonReentrant payable returns (bool){
        require(player[msg.sender].creditBalance >= _amount,"You don't have enough credits");
        require(creditTokens.transfer(msg.sender, _amount) == true);  //Transfer credit tokens from current contract to this player a

        player[msg.sender].creditBalance = player[msg.sender].creditBalance.sub(_amount);  //remove transfered amount from current pl

        emit  evt_TokenAction(msg.sender,_amount, "Withdraw");
}
```

Buy Credits

```
buyCredits() external nonReentrant isHuman whenNotPaused payable
```

Input(s)

Ether amount

Return value

None

Function Details

- Modifier
  - Block the call from smart contract to avoid any attack from other contract
  - Use “ReentrancyGuard” and make sure Reentrancy not happens
  - If the contract is paused by owner, this function call will be denied
- Validation checking
  - Make sure current address is authorized
  - Make sure the Ether amount is greater than zero
  - Make sure there are still some remaining credits

- Calculate purchased credit amount based on Ether as well as base price and make sure there are enough remaining credits amount
- Transfer the Ether to the contract owner account
- Update Credits global status: arised Ether ad remainingCredits
- Transfer the purchased credits amount from owner account to current address at the Credit ERC20 Token contract

```
/* ========Credit tokens Operation \ Buy credit tokens using Ether (Wei)========================== */
function buyCredits() external nonReentrant isHuman whenNotPaused payable {
      require(player[msg.sender].isAuthorized == 1,"This player was not authorized yet");
      require(msg.value > 0,"Ether amount is zero");
      require(creditStatus.remainingCreditTokens > 0 ,"All credits sold out");


      uint256 weiAmount = msg.value;   //Ether in Wei amount
      uint256 credits = weiAmount.mul(CREDITPRICE); // Calculate credit tokens to sell based on base price

      require(credits <= creditStatus.remainingCreditTokens,"No enough remaining credit tokens");

      owner.transfer(weiAmount);    // Send Eth from buyer to contract owner before any status changes

      //Update Credit global status
      creditStatus.arisedEth = creditStatus.arisedEth.add(weiAmount); //
      creditStatus.remainingCreditTokens = creditStatus.remainingCreditTokens.sub(credits);

      require(creditTokens.transferFrom(owner, msg.sender, credits) == true);         //Process token purchase in th

      emit  evt_TokenAction(msg.sender,credits, "Buy");
}
```

Get Player Info

```
getPlayerInfo(address _player) external view returns (uint,uint,uint8)
```

Input(s)

Player address

Return value

- Player credit token balance
- Player Ether balance
- Player isAuthorized flag

```
}

function getPlayerInfo(address _player) external view returns (uint,uint,uint8) {
  return (player[_player].creditBalance,player[_player].ethBalance,player[_player].isAuthorized);
}

 function getCreditStatus() external view returns (uint256,uint256,uint256,uint256,uint256,uint256) {
  return (creditStatus.cap,creditStatus.authorizedPlayers,creditStatus.assignedCreditTokens,creditStatus.remainingCreditTokens,creditSta
}

 function getMessageInfo(uint _messageid) external view returns (uint,address,string,bool) {
  return (messageinfo[_messageid].CreatDate,messageinfo[_messageid].Creator,messageinfo[_messageid].MessageContent,messageinfo[_messagei
}


function kill() external onlyAdministrators { //onlyOwner can kill this contract
  selfdestruct(owner);  // `owner` is the owners address
}
```

Get Global Credit Status

```
getCreditStatus() external view returns
(uint256, uint256, uint256, uint256, uint256, uint256)
```

Input(s)

None

Return value

- Total credit supply (Cap)
- Total number of authorized players
- Total amount of assigned credits
- Total remaining credits
- Total amount of arised credits
- Total amount of arised Ether

Demo Script

ENVIRONAMNT DESCRIOPTION

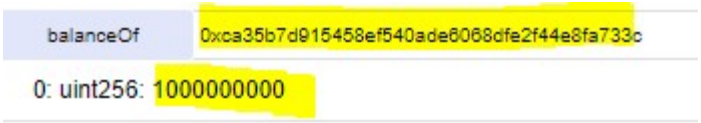
Two smart contracts are created and deployed under Remix

CREDIT TOKEN CONTRACT (ERC20)

Check the Contract Symbol Name

Check the contract Owner

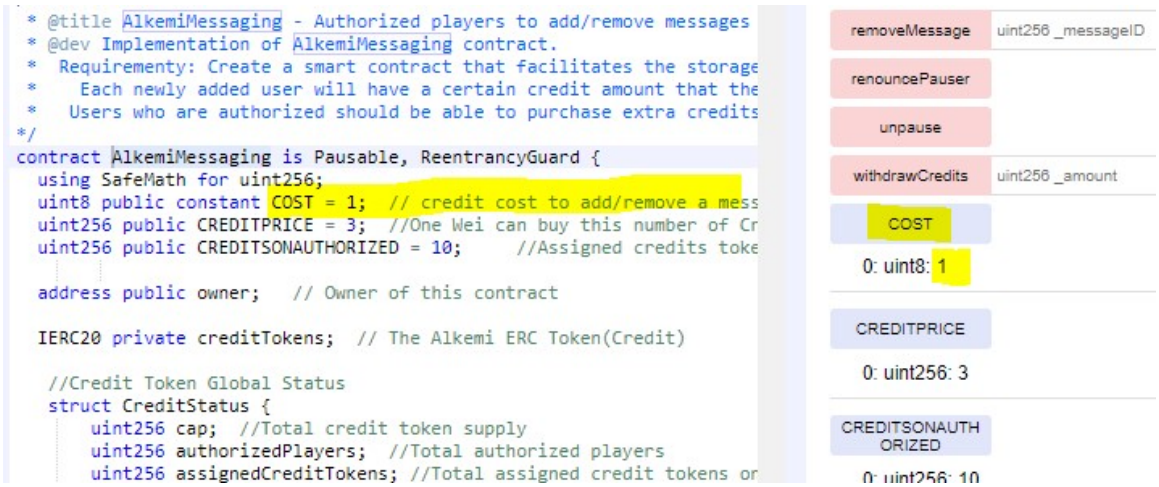
Check the contract owner token balance (By default, the owner own all 1 billion tokens)



MESSAGING CONTRACT (ALKEMIMESSAGING)

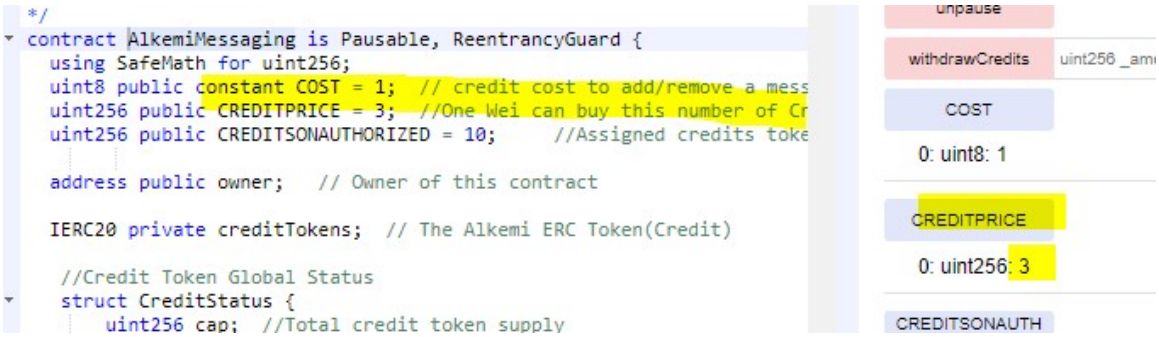
Check the "COST" as specified at the contract

This is the credit amount withdraw from user account after a message is added/removed



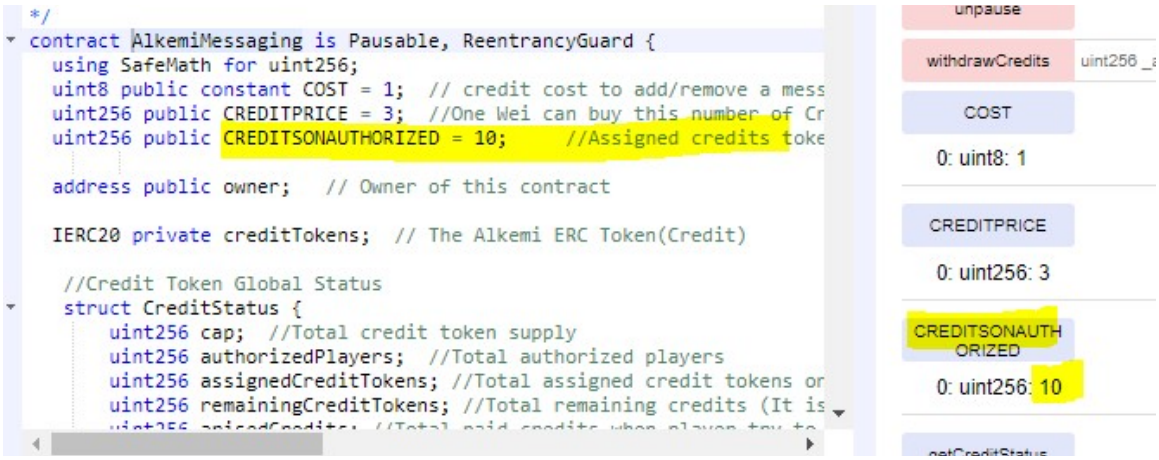
Check Credit purchase base price

This is the base price to purchase credit tokens (1 Wei can buy 3 credits at this demo)

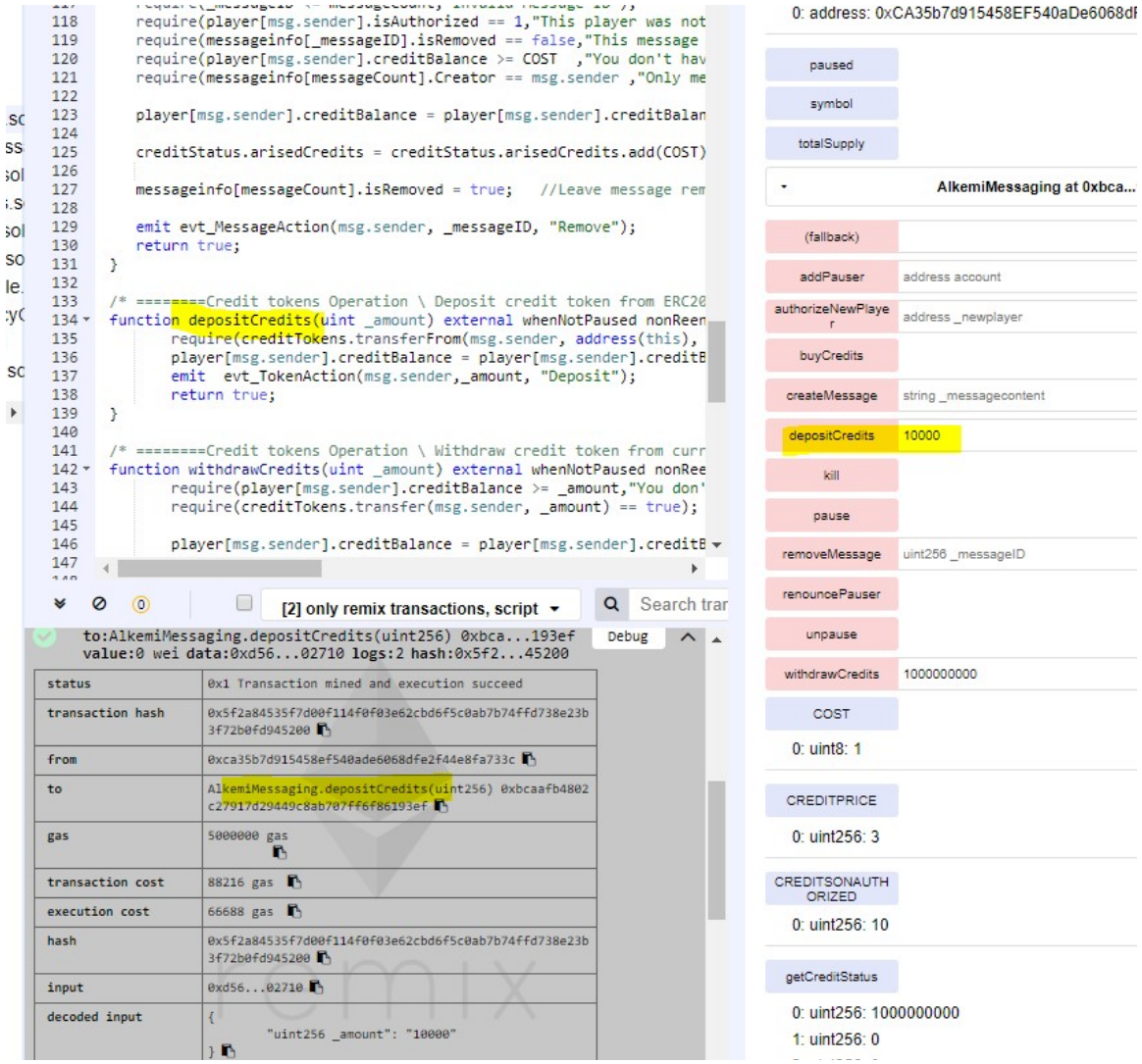


Check the specified credit amount assigned to new address upon they are authorized

10 credit tokens will be assigned to new address at this demo



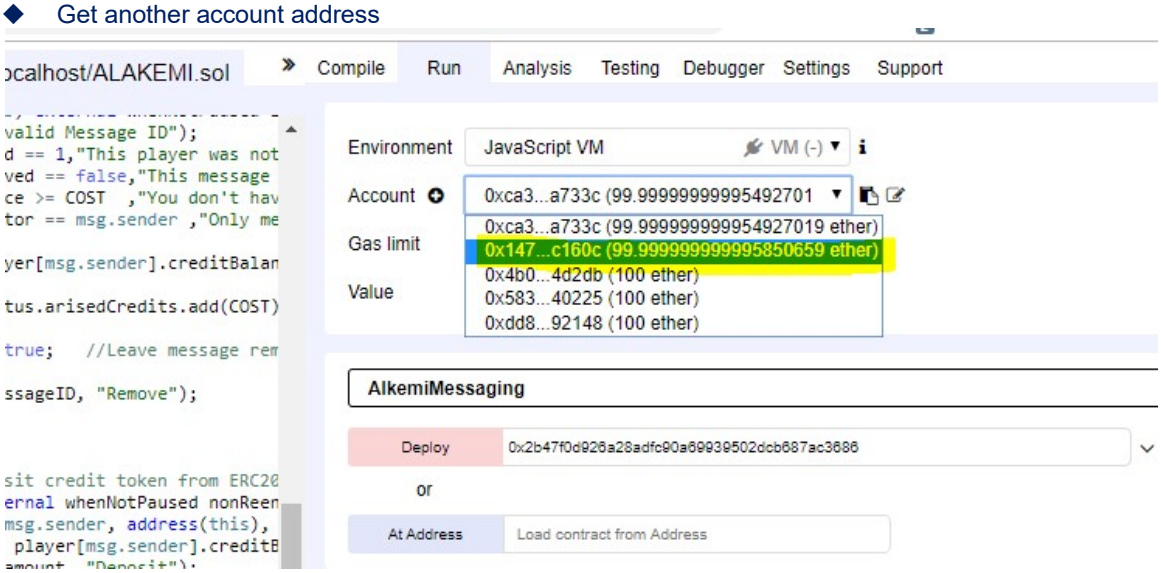
Switch to Owner account and deposit 10000 tokens from ERC20 credit token contract to messaging contract



Specified owner address and check 1000 tokens are deposited to owner account successfully via following function at the messaging contract

Authorize a new address by owner

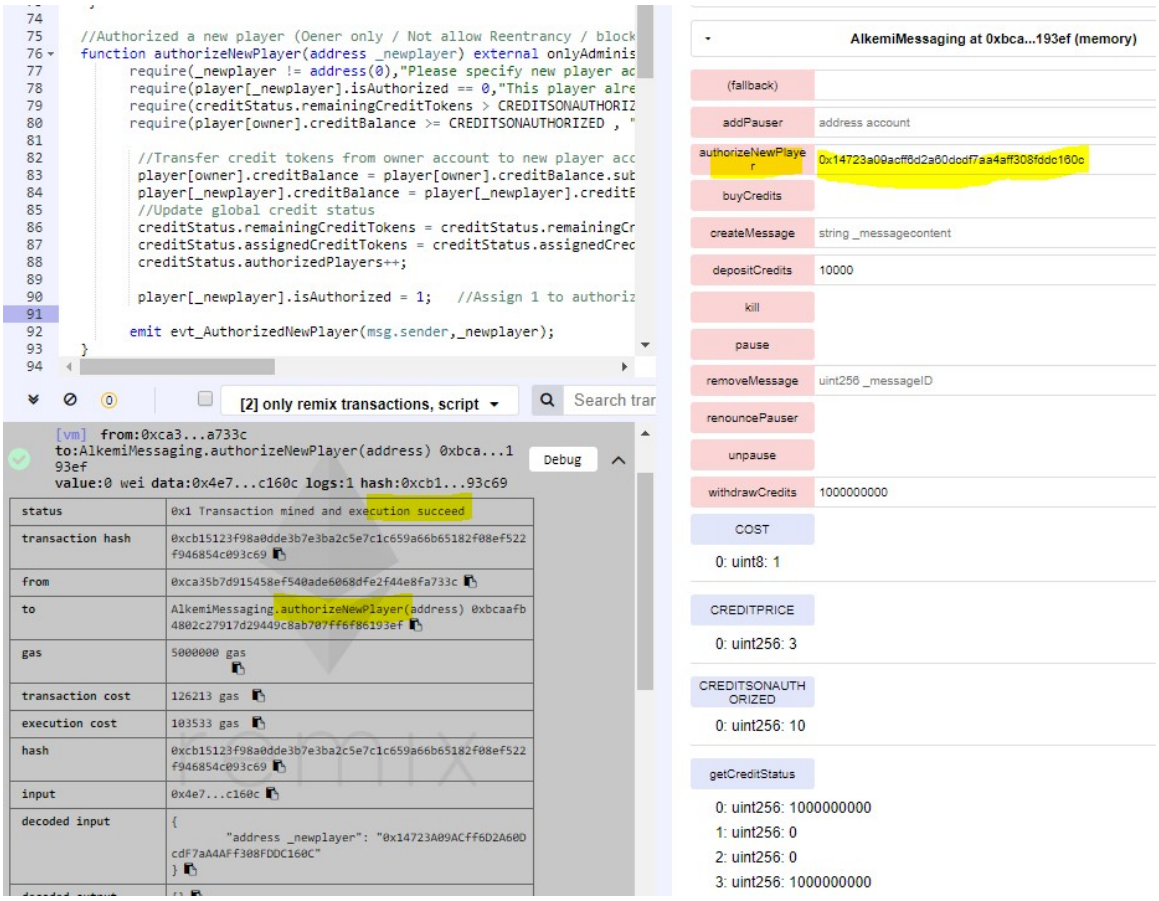
- Get another account address



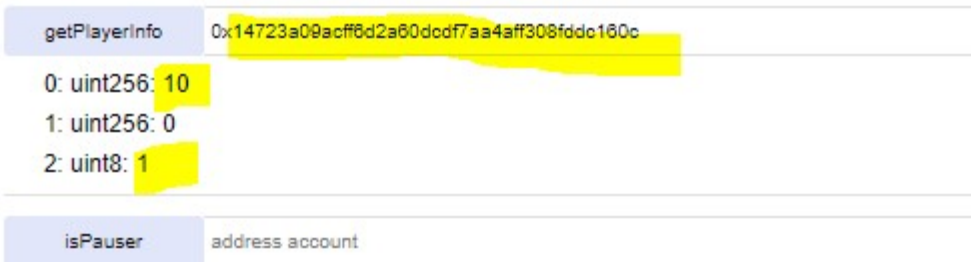
- Before the authorization, Get the new address credit balance.

  Paste the address and get the credit balance before the authorization (it is 0 at this demo)

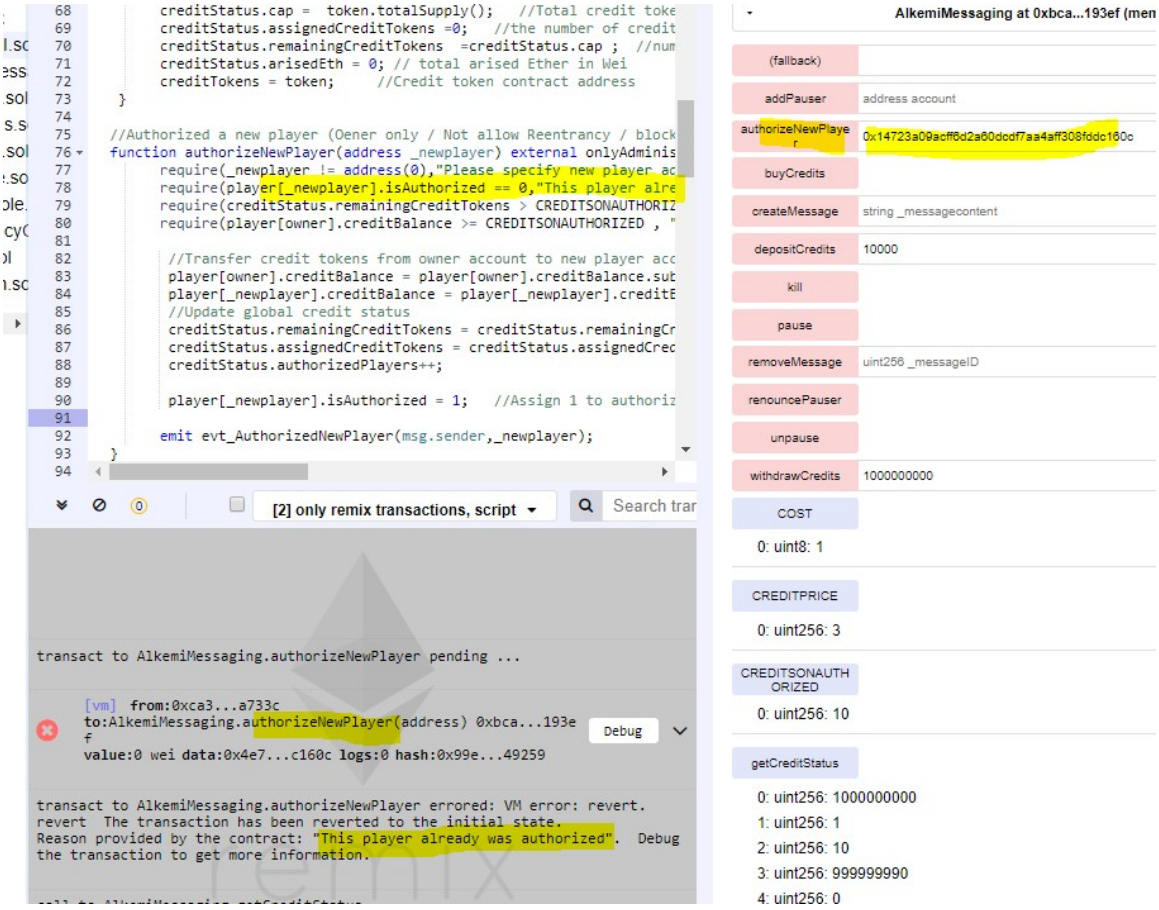
- Execute the new address authorization
  Make sure switch to owner account, Specify the new address and execute the new address authorization



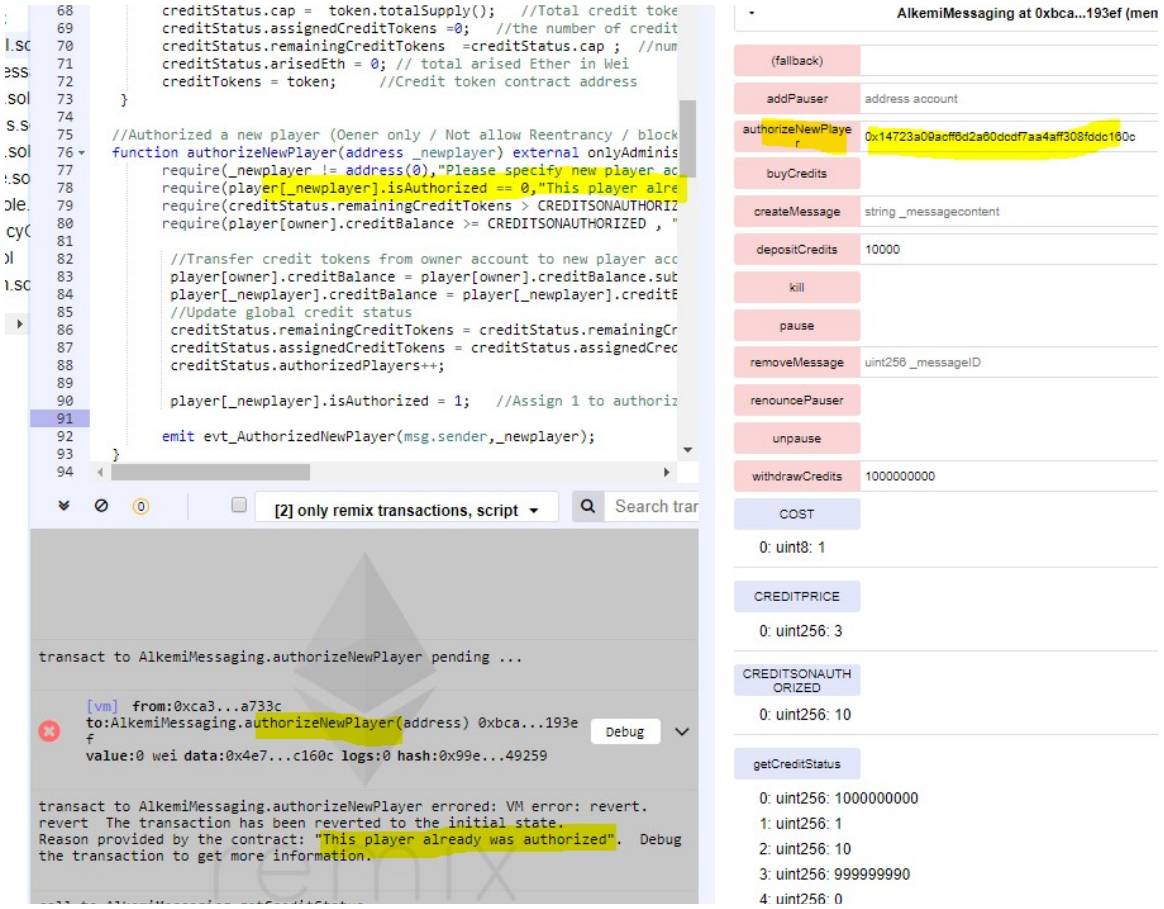
- Verify 10 tokens are assigned to the new address and the authorization flag is set as “1”



- Verify the global credit status is updated
  Such as, at following sample.
  - The total assigned player” is set as 1.
  - total assigned credits is set as “10”
  - 10 was reduced from the total remaining credits

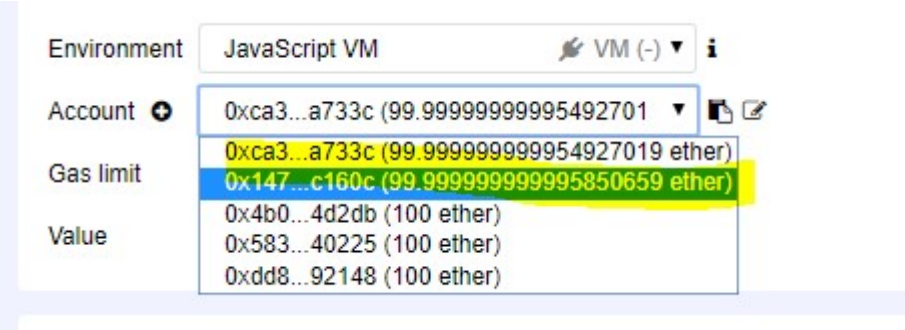


- Execute the new address authorization one more time with the same address
  The authorization will be declined because that address already was authorized.

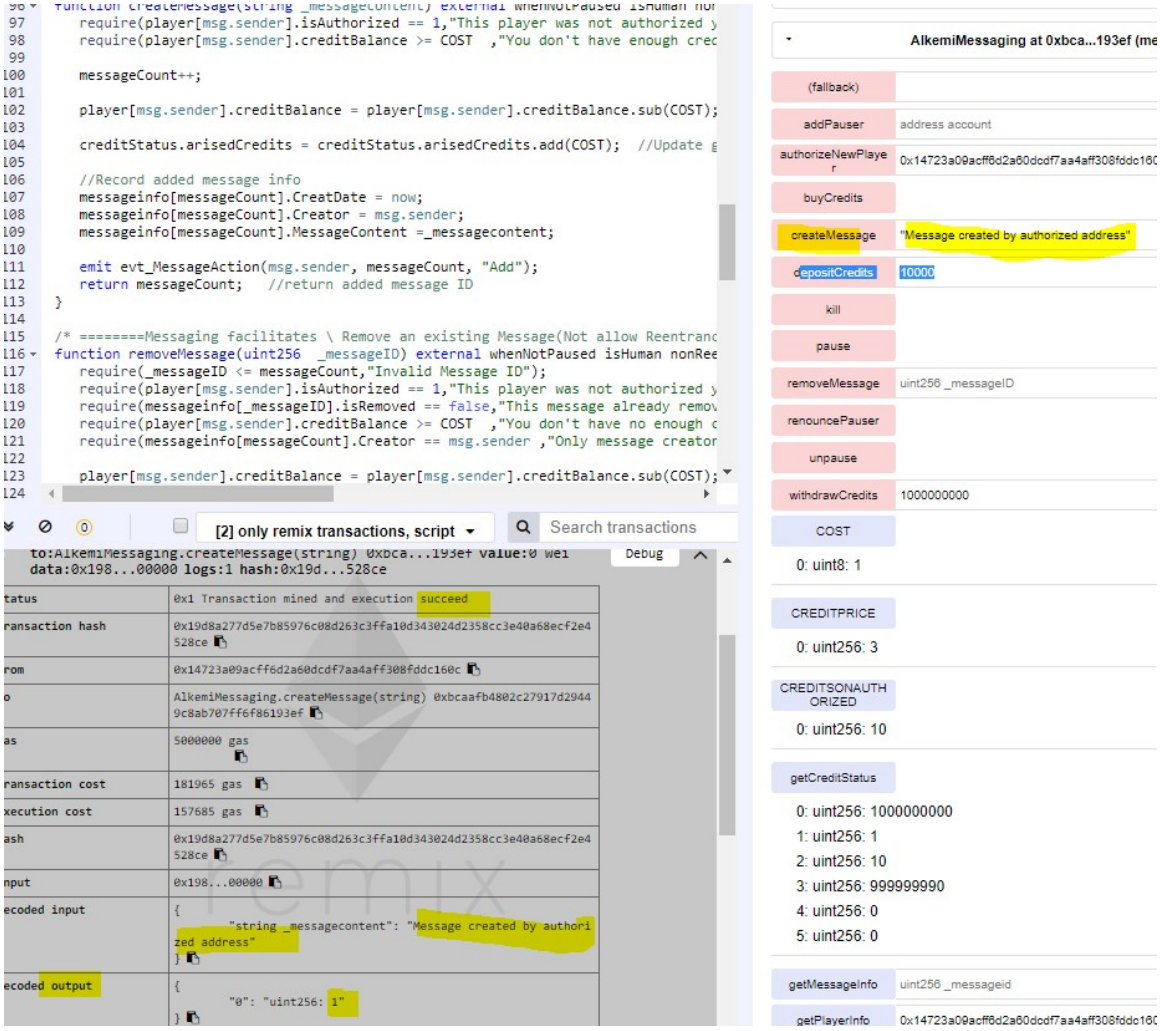

creditStatus.cap = token.totalSupply(); //Total credit toke
creditStatus.assignedCreditTokens =0; //the number of credit
creditStatus.remainingCreditTokens =creditStatus.cap ; //num
creditStatus.arisedEth = 0; // total arised Ether in Wei
creditTokens = token; //Credit token contract address
}
//Authorized a new player (Oener only / Not allow Reentrancy / block
function authorizeNewPlayer(address _newplayer) external onlyAdminis
require(_newplayer != address(0),"Please specify new player ac
require(player[_newplayer].isAuthorized == 0,"This player alre
require(creditStatus.remainingCreditTokens > CREDITSONAUTHORIZ
require(player[owner].creditBalance >= CREDITSONAUTHORIZED , "
//Transfer credit tokens from owner account to new player acc
player[owner].creditBalance = player[owner].creditBalance.sub
player[_newplayer].creditBalance = player[_newplayer].creditB
//Update global credit status
creditStatus.remainingCreditTokens = creditStatus.remainingCr
creditStatus.assignedCreditTokens = creditStatus.assignedCred
creditStatus.authorizedPlayers++;
player[_newplayer].isAuthorized = 1; //Assign 1 to authoriz
emit evt_AuthorizedNewPlayer(msg.sender,_newplayer);
}
[2] only remix transactions, script
Search trar
transact to AlkemiMessaging.authorizeNewPlayer pending ...
[vm] from:0xca3...a733c
to:AlkemiMessaging.authorizeNewPlayer(address) 0xbca...193ef
value:0 wei data:0x4e7...c160c logs:0 hash:0x99e...49259
Debug
transact to AlkemiMessaging.authorizeNewPlayer errored: VM error: revert.
revert The transaction has been reverted to the initial state.
Reason provided by the contract: "This player already was authorized". Debug
the transaction to get more information.
AlkemiMessaging at 0xbca...193ef (mem
(fallback)
addPauser
address account
authorizeNewPlayer
0x14723a09acff6d2a60dcdf7aa4aff308fddc160c
buyCredits
createMessage
string _messagecontent
depositCredits
10000
kill
pause
removeMessage
uint256 _messageID
renouncePauser
unpause
withdrawCredits
1000000000
COST
0: uint8: 1
CREDITPRICE
0: uint256: 3
CREDITSONAUTHORIZED
0: uint256: 10
getCreditStatus
0: uint256: 1000000000
1: uint256: 1
2: uint256: 10
3: uint256: 999999990
4: uint256: 0

Add a new message / Remove a message

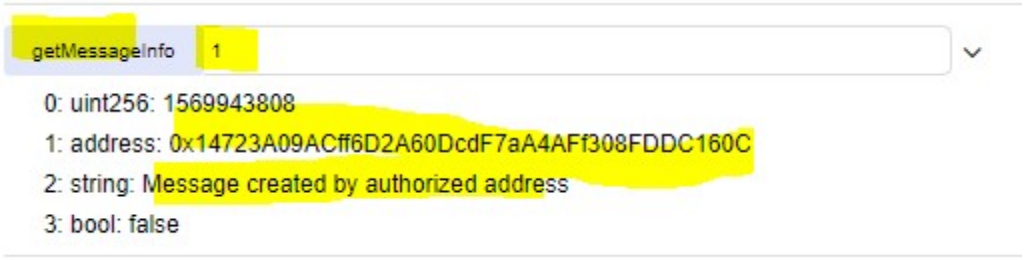
- Switch to above authorized address

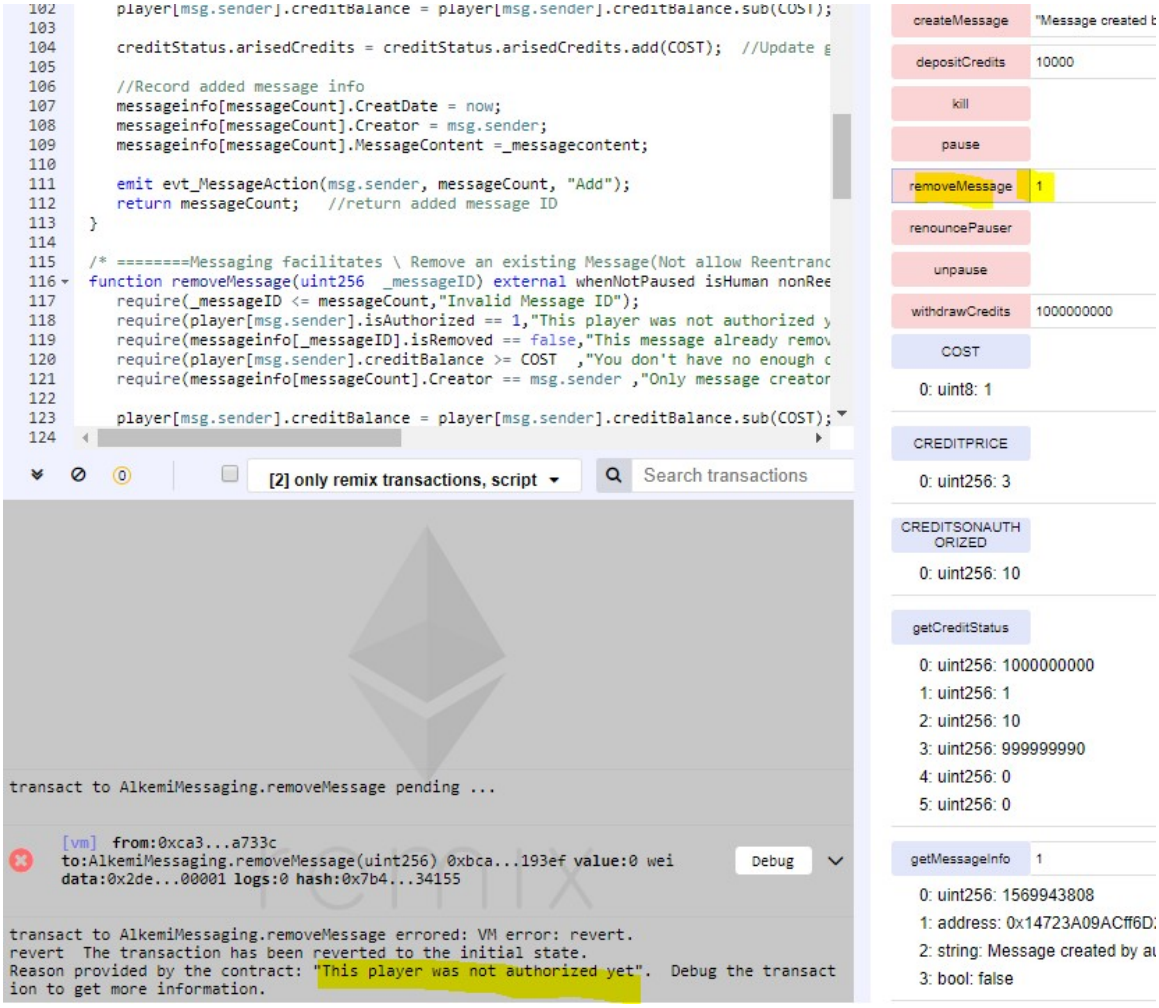


- Specify any message content and execute “createmessage” function
  The function is executed successfully and returns the message ID “1” at following sample



- Specify the message ID “1” and execute “getMessageInfo”
  It returns the message details (such as message creation date/time, message creator address, and message content and isremoved flag)

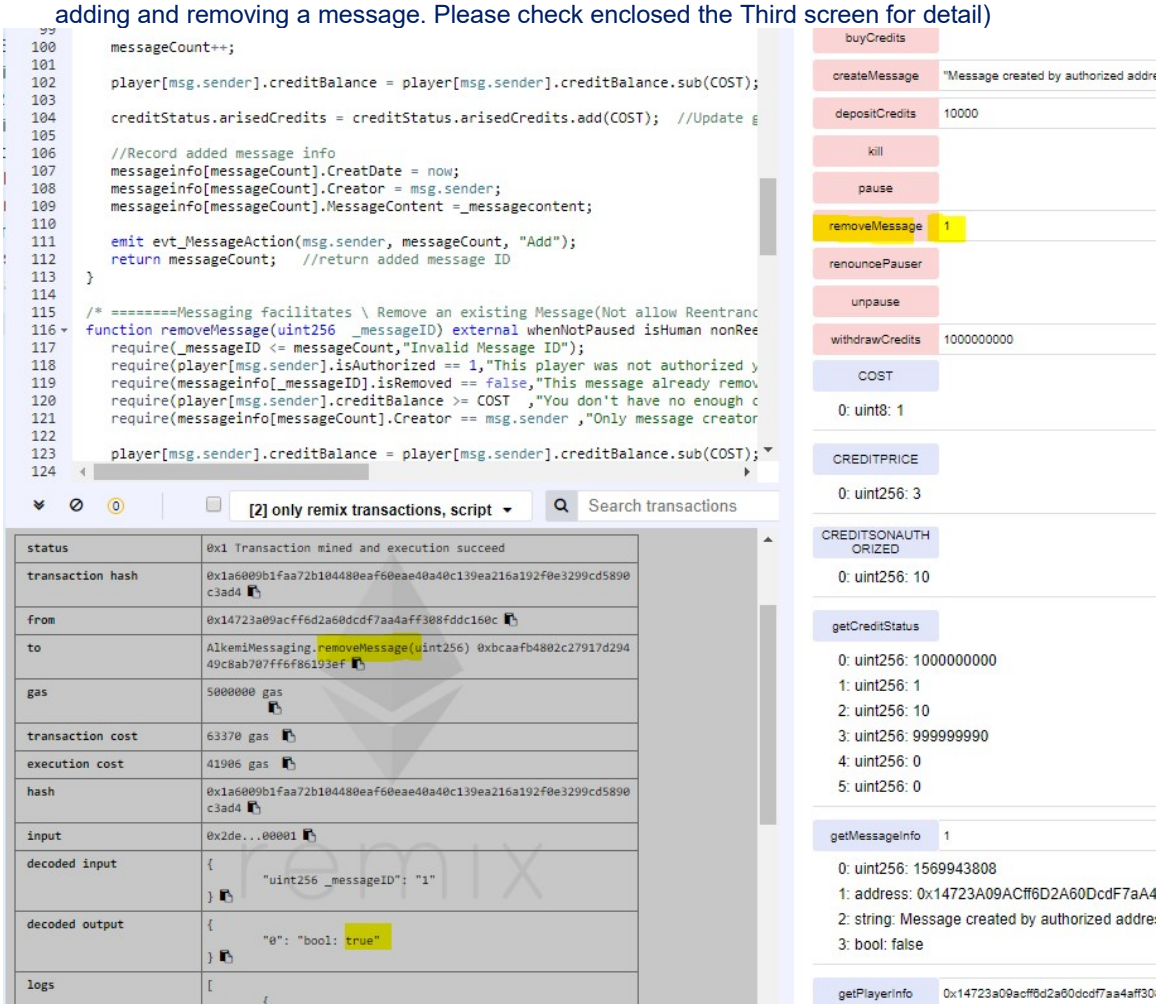


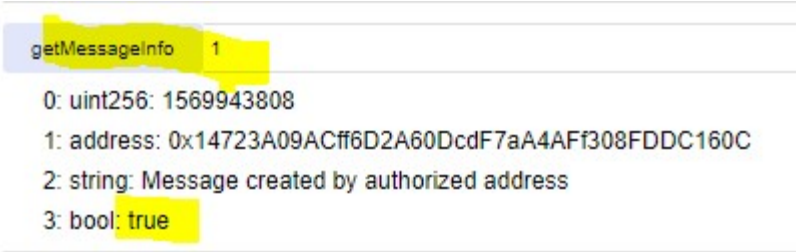


- Stay at the same address and execute the “removemessage” with the messageID as “1”

  The message flag “isRemioved” is changed from “false” to “true” (see the second screen below)

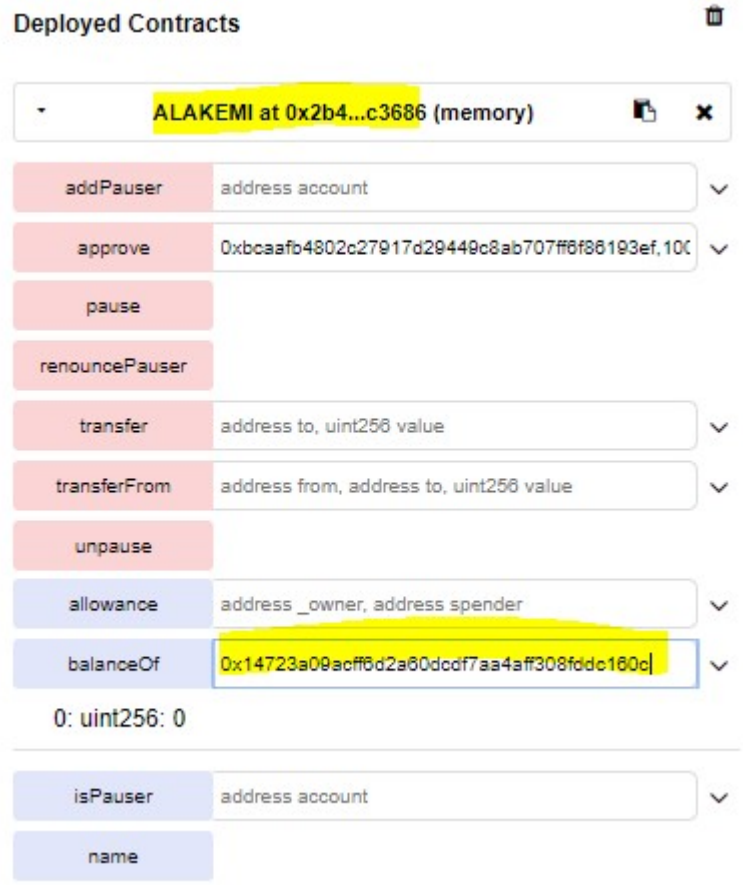
One more credits are withdraw from current player account (When the address is authorized, there are 10 credits are assigned at his account. It reduces to 8 after adding and removing a message. Please check enclosed the Third screen for detail)



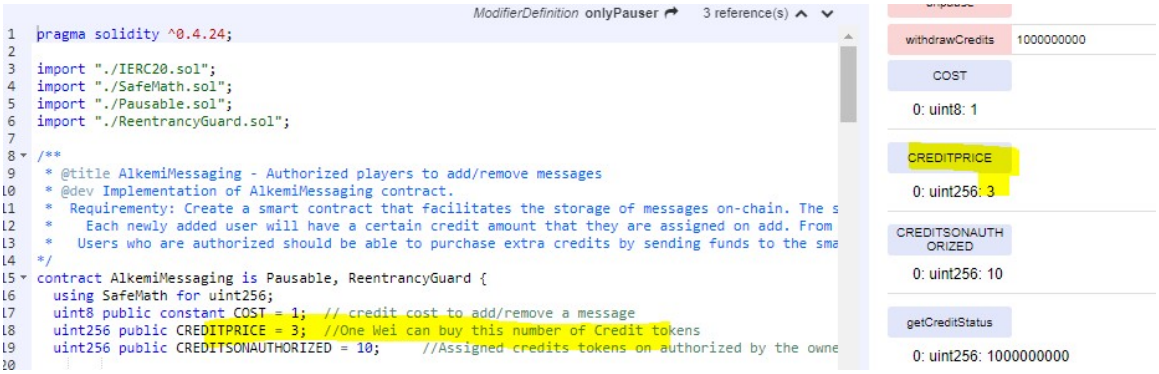


Buy Credits

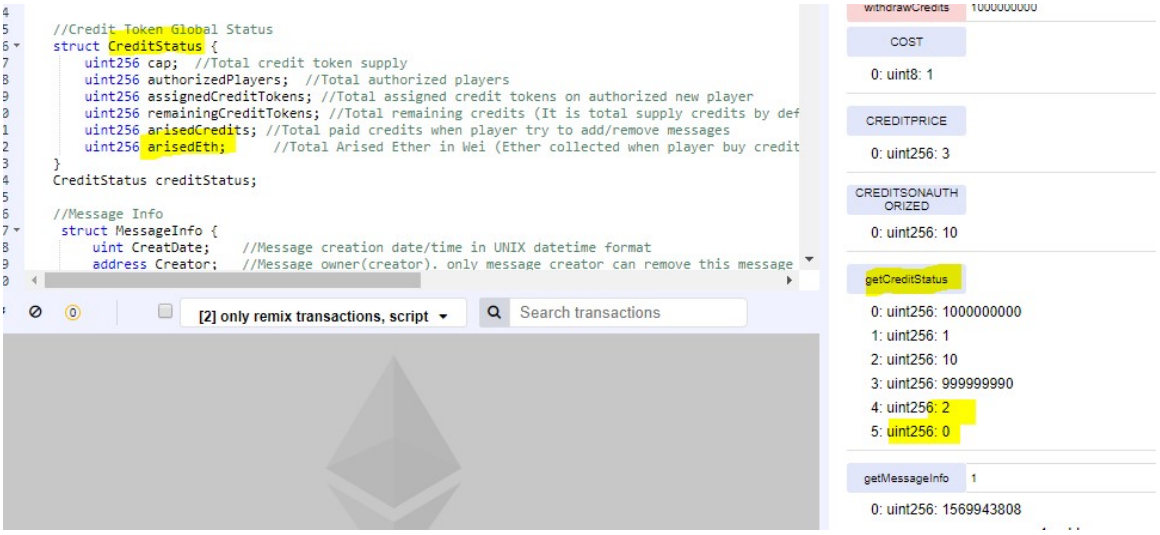
- Switch to any authorized address
- Get the credit balance at the ERC20 token contract before the buying. Such as: the balance is 0 for that address at the ERC20 contract



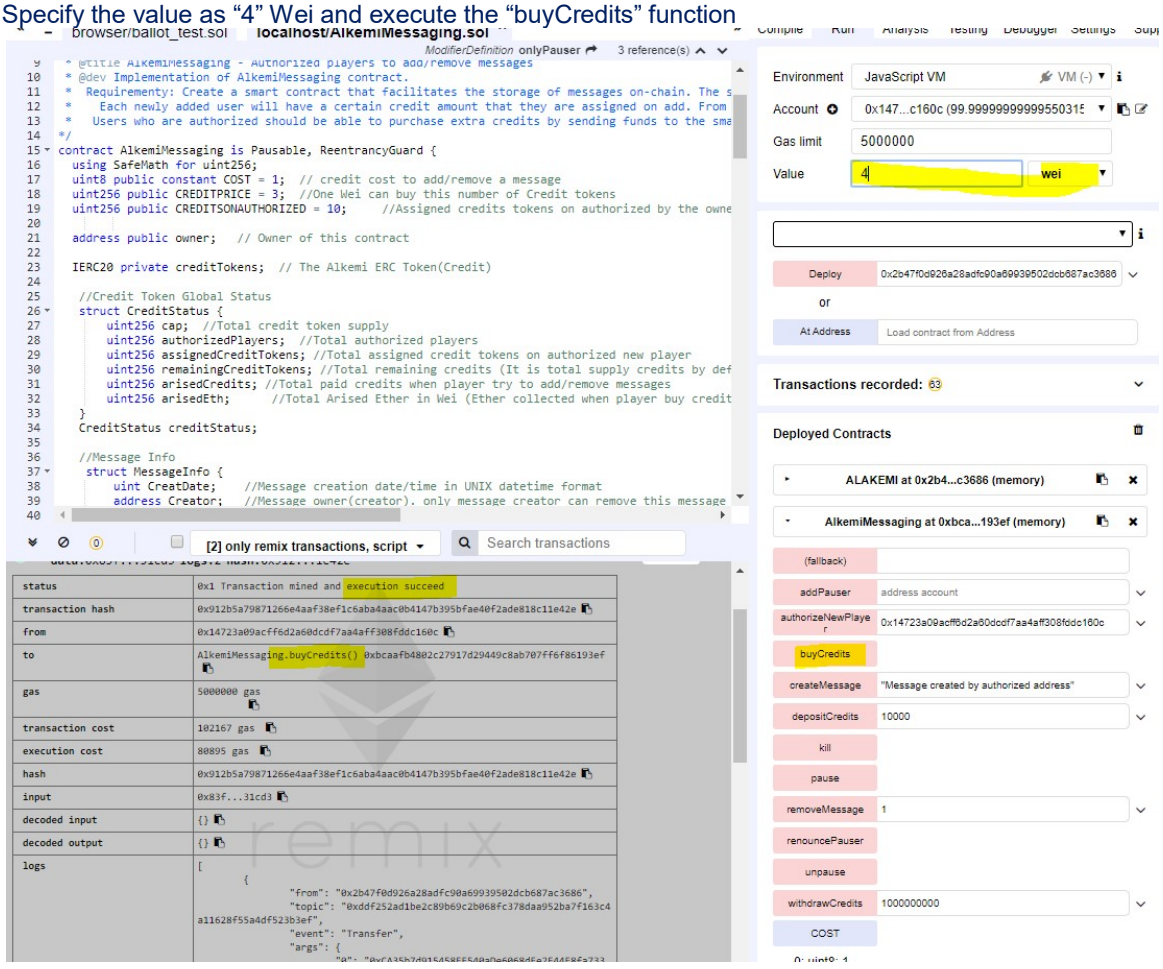
- Verify the credit price before the buying. It is set as “1 Wei can buy 3 credits” at following sample. Such as if current address paid 4 Wei, there should be 12 credits deposited to his balance



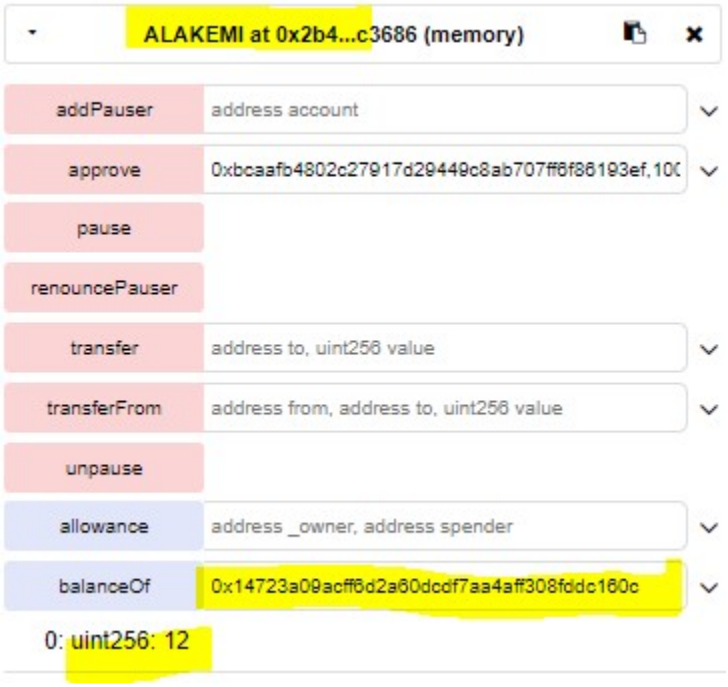
- Verify the global credit status. After the buying, the ether amount should be added to the arisedEther (Before the buying, the arisedEther is 0 at following sample)



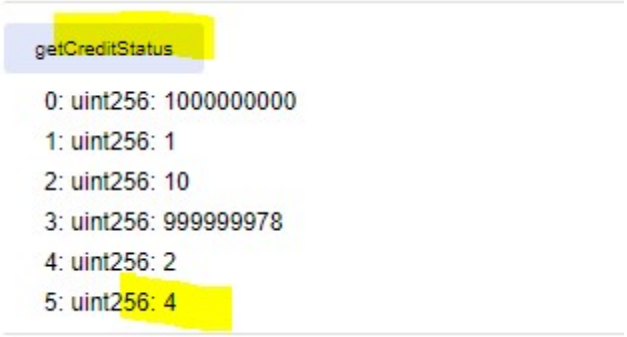
- Specify the value as “4” Wei and execute the “buyCredits” function



- Verify 12 credit tokens is transferred to the address successfully at the ERC20 token contract

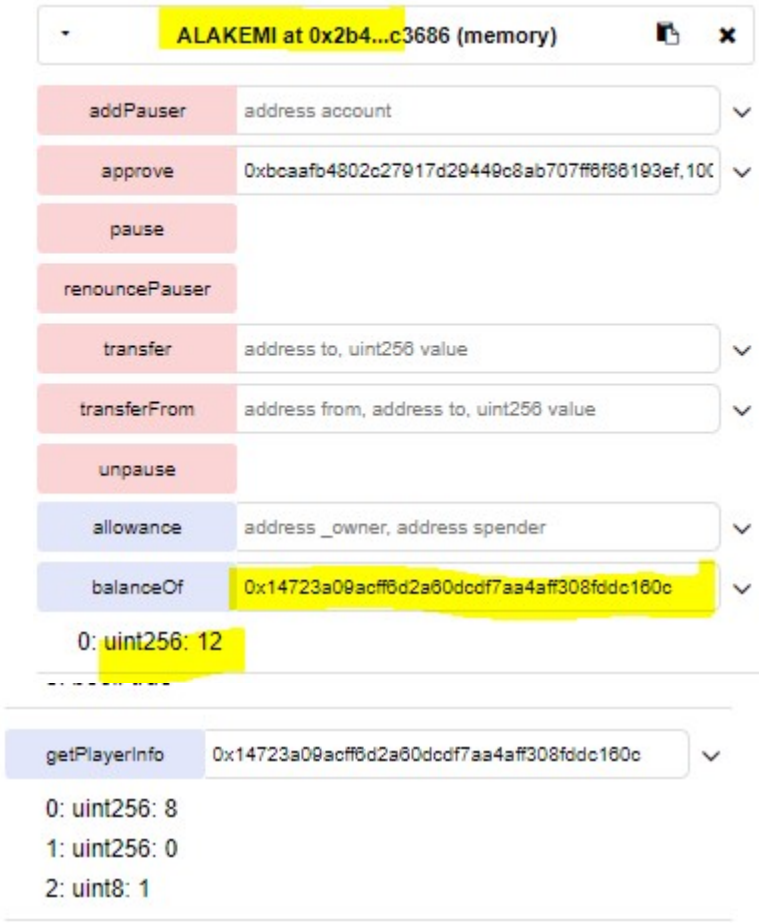


- Verify 4 Wei is added to the global credit status successfully.

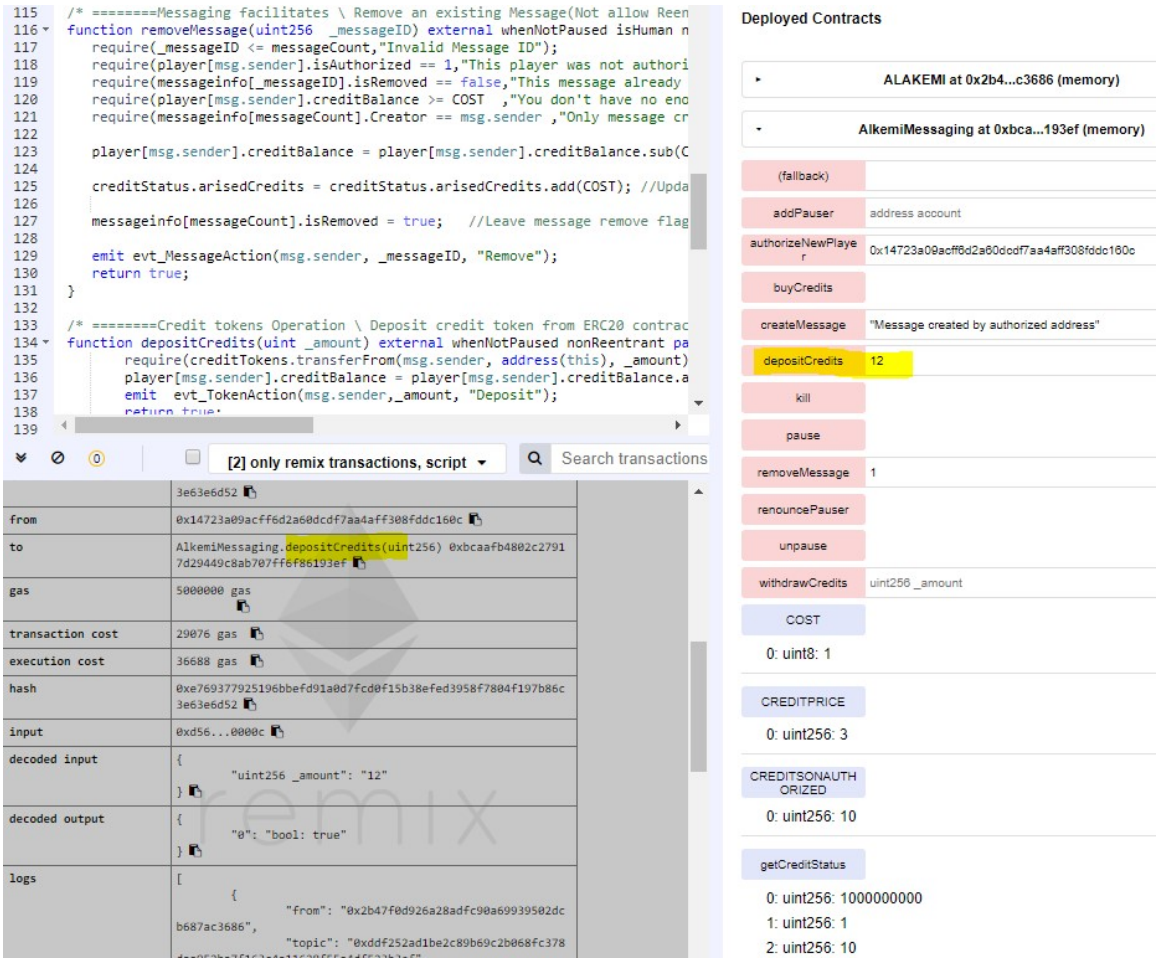


Deposit credits from “ALAKEMI” ERC20 smart contract to messaing contract “AlkemiMessaging”

- Continue above steps and there are 12 tokens at “ALAKEMI” ERC20 contract and 8 tokens at the messaging contract before the “deposit”

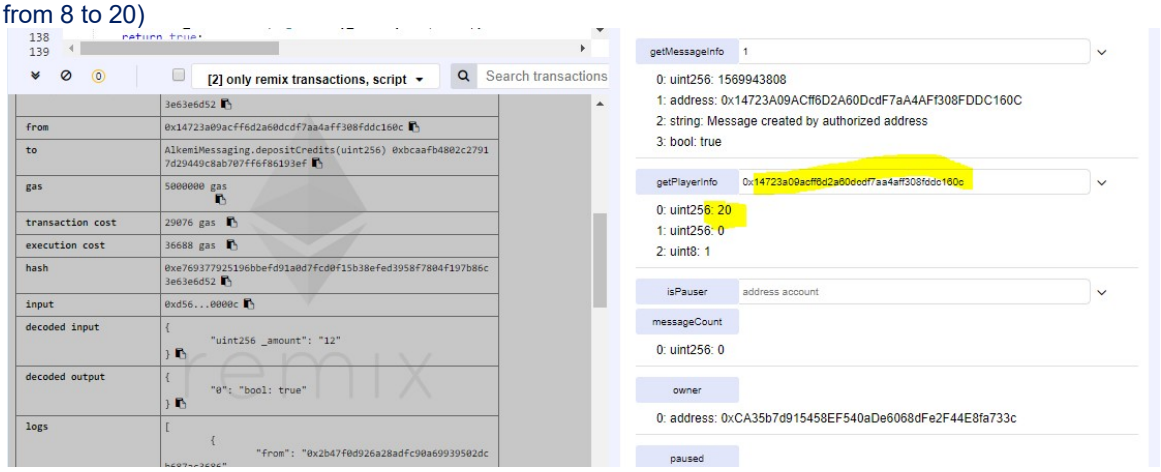


- Execute “DepositCredits” transaction and deposit 12 tokens from ERC20 contract to messaging contract



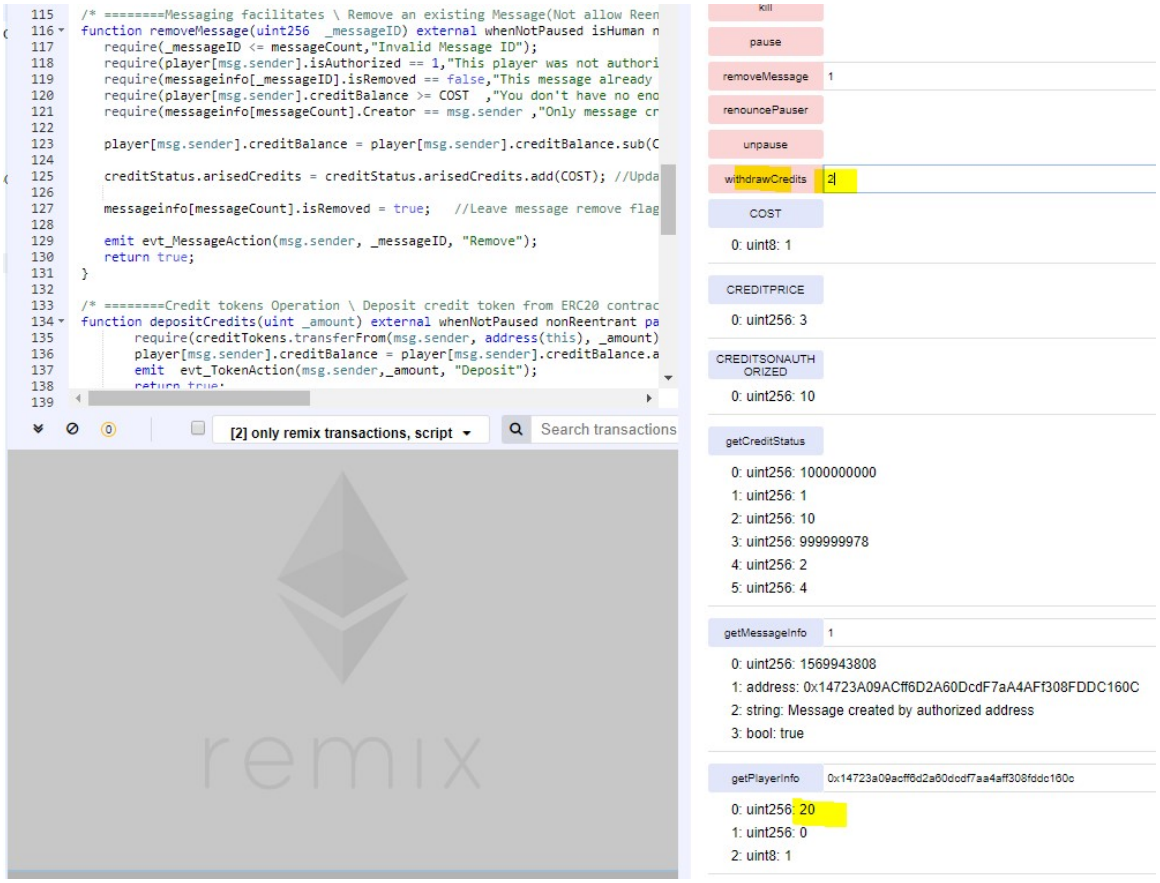
- Verify the 12 tokens are reduced from ERC20 contract (changed from 12 to 0)

- Verify the 12 tokens are added to his account at the messaging contract (changed from 8 to 20)

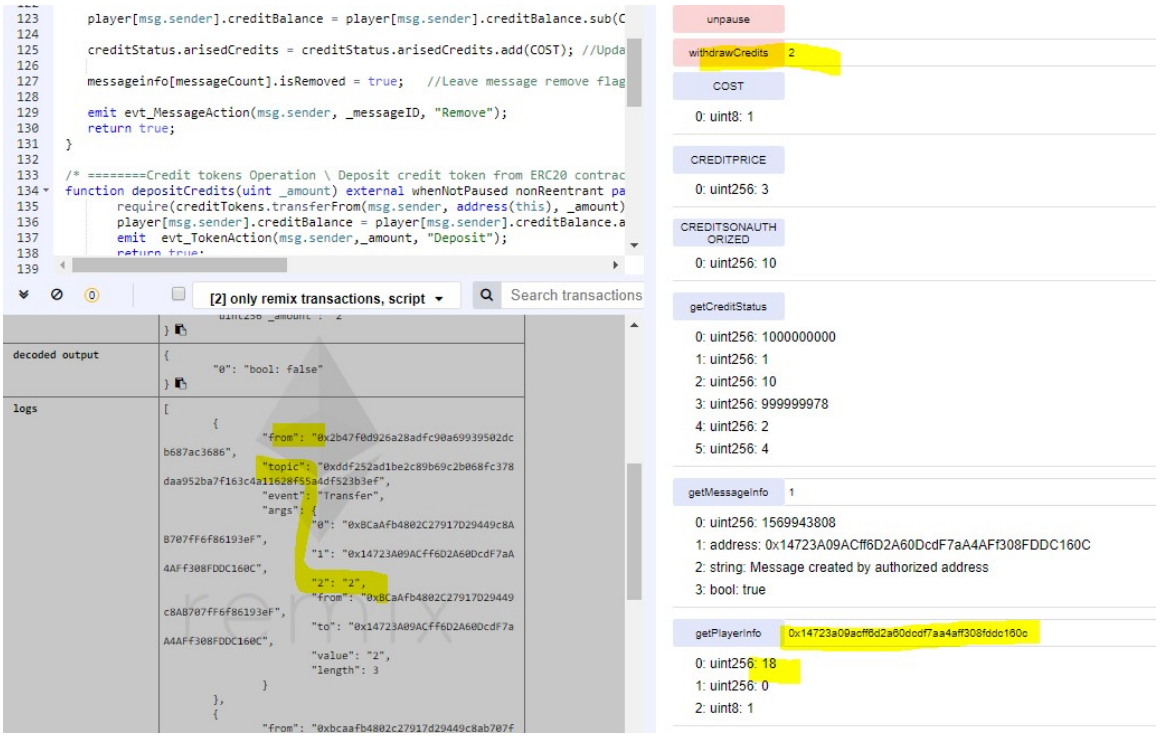


Withdraw credits from messaging contract “AlkemiMessaging” to “ALAKEMI” ERC20 smart contract

- Specify 2 tokens and withdraw 2 tokens from messaging contract “AlkemiMessaging” to “ALAKEMI” ERC20 smart contract (Before the transaction, there are 20 tokens balance at his account of messaging contract “AlkemiMessaging”



- After the transaction, his account changed from 20 to 2 at the messaging contract “AlkemiMessaging”



- Verify the 2 tokens are transferred to his account at the essaging contract “AlkemiMessaging”