

Assignment #3

Student name:

Course: *Blockchains & Distributed Ledgers*

Due date: *January 8, 2023*

Part 1

Internal Variables and Functions.

- `mapping(address => uint256) private balances` is used to associate each user with his token balance
- `uint256 private _supply` is used to keep track of circulating supply of tokens
- `string private _tokenName` is the name of the token set by the owner through the constructor
- `string private _tokenSymbol` is the symbol of the token set by the owner through the constructor
- `uint128 constant private _price = 600` price of the token
- modifier `onlyOwner()` is used to allow only the owner of the contract to mint tokens and close the contract

The process.

The process goes as follows. First, the owner deploys the contract and sets the name and symbol of the token. Once the contract is deployed, a user who wants to buy some tokens starts communicating with the owner. They agree on the number of tokens the user will buy and the price. The user also provides the address from which he will send the ether to the contract. Typically, the amount of ether that the user sends should also cover the owner's gas for the later minting of the tokens, or the user can also send some fee to the owner outside the blockchain. The user then sends the agreed amount of ether by a call with empty calldata. The call executes the `receive` function on the contract. After the transaction is confirmed, the user informs the owner and provides him with some proof that he did send the agreed amount of ether to the contract (e. g., Transaction Details on etherscan). The owner then calls the `mint` function in which he supplies the address of the user and the agreed amount of tokens that will be sent. The `mint` function checks that the contract balance is sufficient to mint new tokens.

The user can also transfer tokens to another user by calling the `transfer` function and supplying the amount he wants to send and the address he wants to send

to. The function checks that the receiving address is not `address(0)` and that the user's balance is sufficient to perform the transfer. The function then updates the balances of the sender and the receiver, emits the Transfer event and returns `true` if executed successfully.

If the user wants to sell tokens, he calls the `sell` function and supplies the number of tokens he wants to sell. The function first checks that the balance of the user is sufficient. The tokens are deducted from his balance, and ethers to the user are sent from the contract's balance. The tokens are sold for 600 wei per token.

Users can access their balance by calling the function `balanceOf` and supplying their address. The function returns the associated value from the mapping `balances`. Users can also easily check the balance of other users.

Gas evaluation.

Gas analysis. For a better understanding of costs, I have used `hardhat-gas-reporter`. The contract deployment, on average, costs 560515 units. On average, the mint function costs 70503 units, the sell function 37439 units and the transfer function 51473 units. More details can be seen in the picture.

| | | | | | | |
|---------------------|----------|-------------------------|-------|------------|---------------------------|-----------|
| Solc version: 0.8.7 | | Optimizer enabled: true | | Runs: 1000 | Block limit: 30000000 gas | |
| Methods | | | | | | |
| Contract | Method | Min | Max | Avg | # calls | usd (avg) |
| B32245_part1 | close | - | - | 28527 | 1 | - |
| B32245_part1 | mint | 70493 | 70505 | 70503 | 12 | - |
| B32245_part1 | sell | 35519 | 40319 | 37439 | 10 | - |
| B32245_part1 | transfer | 51463 | 51475 | 51473 | 7 | - |
| Deployments | | | | | % of limit | |
| B32245_part1 | | - | - | 560515 | 1.9 % | - |

13 passing (1s)

The current average gas price on etherscan (at the moment of writing) is said to be 17 gwei. Below is the estimated average cost in ether and pounds (by the current value of 1040,04 GBP for 1 ETH).

- deployment: $560515 * 17 \text{ gwei} = 9528755 \text{ gwei} = 0,009528755 \text{ ether} (= 9,91 \text{ pounds})$
- mint: $70503 * 17 \text{ gwei} = 1198551 \text{ gwei} = 0,001198551 \text{ ether} (= 1.25 \text{ pounds})$
- sell: $37439 * 17 \text{ gwei} = 636463 \text{ gwei} = 0,000636463 \text{ ether} (= 0,66 \text{ pounds})$
- transfer: $51473 * 17 \text{ gwei} = 875041 \text{ gwei} = 0,000875041 \text{ ether} (= 0,91 \text{ pounds})$

Below are the actual costs of deploying and interacting with the contract (on Metamask):

| Contract deployment | Mint | Sell | Send Token |
|--|---|---|---|
| Status View on block explorer Confirmed Copy transaction ID From 0xb89...27Da → To New contract Transaction Nonce 3 Amount -0 GoerliETH Gas Limit (Units) 600699 Gas Used (Units) 600699 Base fee (GWEI) 17.693971993 Priority fee (GWEI) 2.5 Total gas fee 0.01213 GoerliETH Max fee per gas 0.000000034 GoerliETH Total 0.0121305 GoerliETH | Status View on block explorer Confirmed Copy transaction ID From 0xb89...27Da → 0x99E...b5d4 To Transaction Nonce 4 Amount -0 GoerliETH Gas Limit (Units) 70914 Gas Used (Units) 70914 Base fee (GWEI) 21.285326403 Priority fee (GWEI) 2.5 Total gas fee 0.001687 GoerliETH Max fee per gas 0.00000005 GoerliETH Total 0.00168671 GoerliETH | Status View on block explorer Confirmed Copy transaction ID From 0x12B...3d5c → 0x99E...b5d4 To Transaction Nonce 2 Amount -0 GoerliETH Gas Limit (Units) 42529 Gas Used (Units) 35546 Base fee (GWEI) 23.369009296 Priority fee (GWEI) 2.5 Total gas fee 0.00092 GoerliETH Max fee per gas 0.000000045 GoerliETH Total 0.00091954 GoerliETH | Status View on block explorer Confirmed Copy transaction ID From 0x12B...3d5c → Account 3 To Transaction Nonce 1 Amount -0 GoerliETH Gas Limit (Units) 51730 Gas Used (Units) 51730 Base fee (GWEI) 21.027312932 Priority fee (GWEI) 2.5 Total gas fee 0.001217 GoerliETH Max fee per gas 0.000000041 GoerliETH Total 0.00121707 GoerliETH |

Techniques to make the contract more cost-effective.

- `mapping(address => uint256) private balances` is used to associate each user with the number of tokens he possesses. Instead of using an array or other structures, mapping presents a much cheaper way.
- `uint128 constant private _price = 600` is used to store the price. The constant enables the price to be stored directly in the contract's bytecode, instead of the contract's state, so it does not require the SLOAD operation to read it.
- `revert` instead of `require` is used since it can be more gas efficient when being executed
- `Error error NotOwner()`, `error ZeroAddress()`, `error LowBalance()`, `error LowMint()` are used to describe the error, instead of string messages, as they are encoded in only four bytes and therefore cost less in deployment and runtime.
- `receive` is an empty function. No events are emitted when the user sends ether to the contract.
- Use of Solidity optimizer that reduces gas for contract deployment.
- `uint256` is used instead of lower uints. Although some variables such as `_supply` could be defined in lower uints, that would result in more gas when computed.
- Addresses in events are indexed, as they present lower gas costs than unindexed.

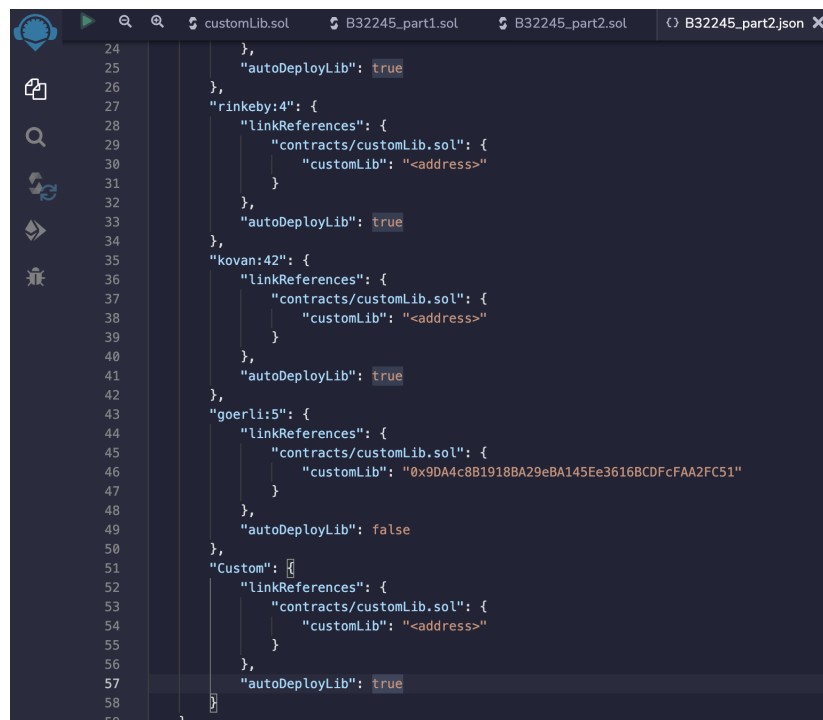
Potential vulnerabilities, mitigations and trust assumptions.

- An important trust assumption that poses a potential vulnerability is the owner of the contract. The owner can self-destruct the contract at any time, stealing other users' balance, by calling the `close` function. Therefore users must trust the owner that he will not self-destruct the contract without notifying them and enabling them to sell the tokens they own.
 - Another trust assumption comes when the user sends ethers to the contract and expects the owner to mint tokens to the user's address. The owner can mint tokens to any address, including his own. With the prescribed API, that would be possible to mitigate with additional mapping that would associate each user with the deposit he sent to the contract through the `receive` function. However, since it is not recommended for safety to allow fallback functions to perform any other operations than emitting the event, that is not implemented in the contract.
 - A potential vulnerability could be that the owner decides to mint tokens without paying for them and later selling them. That would disrupt the total supply and prevent some or all of the users from successfully selling their tokens. However, that is mitigated in the `mint` function, which checks that the balance of the contract, deducted by the ether value of supply is sufficient to mint the given amount of new tokens.
 - Another potential vulnerability for the users is the to address they submit in the `address` function. If it is an address of another smart contract that cannot function with tokens, the tokens may be lost. An example of a token standard that, to some extent, does mitigate that threat is the ERC223. However, with the prescribed API, handling the vulnerability in such a way is impossible. Although, the function does at least check that the receiving address is not `address(0)`, since it would also result in lost tokens.
 - To enable only the owner to mint and self-destruct the contract, an `onlyOwner` modifier is defined.
 - Another potential vulnerability is reentrancy in `sell` function. That is mitigated by completing all internal work before calling an regular function `transfer`. It is additionally mitigated with the `transfer` call in `sell` function as it possesses a gas limit of 2300 gwei, preventing the recursive call of the attacking contract.
 - A possible cross-function race conditions between `transfer` and `sell` are also prevented by the exact mechanism as reentrancy.
 - In contrast to ERC20, front-running attacks are irrelevant for this contract, as the users cannot approve other users to spend their tokens.
 - Sandwiching attacks that could affect the price are also irrelevant, as the price is unchangeable and already set to 600 wei per token.
-

Part 2

To use the customSend function from the library, I did the following:

1. I copied the library code from etherscan to Remix, saved it as customLib.sol and compiled the library
2. I compiled my contract B32245_part2.sol, opened the B32245_part2.json in the artifacts, and submitted the address of the deployed instance of the library on Goerli: 0x9DA4c8B1918BA29eBA145Ee3616BCDFcFAA2FC51, and turned autoDeployLib to false.



3. I changed the sell function from

```

function sell(uint256 value) public returns (bool succesfullSell) {
  uint256 senderBalance = balances[msg.sender];
  if(senderBalance < value){
    revert LowBalance();
  }
  balances[msg.sender] = senderBalance - value;
  _supply -= value;
  emit Sell(msg.sender, value);
  payable(msg.sender).transfer(value * PRICE);
  return true;
}

```

to

```

function sell(uint256 value) public returns (bool succesfullSell) {
  uint256 senderBalance = balances[msg.sender];
  if(senderBalance < value){
    revert LowBalance();
  }
  balances[msg.sender] = senderBalance - value;
  _supply -= value;
  emit Sell(msg.sender, value);
  customLib.customSend(value * PRICE, msg.sender);
  return true;
}

```

and imported the library as:

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.7;

import './customLib.sol';

contract B32245_part2 {
```

4. Then, I deployed my contract B32245_part2.sol.



Gas impact of using the deployed library instance.

I have made another contract in which I copied the code from customLib and used it in sell function. I deployed the contract on Goerli to address

0x99Ade23F424aD94E8D94E5229dFBf9E67142f7E8. Below it can be seen that contract from part 2 (634066 units) needed more gas to deploy than the contract from part 1 (600699 units), however contract that had the code included needed more than the contracts from part 2 and part 1 (649895 units).

Contract deployment X

Status [View on block explorer](#)
Confirmed [Copy transaction ID](#)

From  0xb89...27Da  To New contract



Transaction

| | |
|---------------------|-----------------------|
| Nonce | 3 |
| Amount | -0 GoerliETH |
| Gas Limit (Units) | 600699 |
| Gas Used (Units) | 600699 |
| Base fee (GWEI) | 17.693971993 |
| Priority fee (GWEI) | 2.5 |
| Total gas fee | 0.01213 GoerliETH |
| Max fee per gas | 0.000000034 GoerliETH |
| Total | 0.0121305 GoerliETH |

Figure 1: contract of part 1

Contract deployment X

Status [View on block explorer](#)
Confirmed [Copy transaction ID](#)

From  0x538...222b  To New contract



Transaction

| | |
|---------------------|-----------------------|
| Nonce | 7 |
| Amount | -0 GoerliETH |
| Gas Limit (Units) | 634066 |
| Gas Used (Units) | 634066 |
| Base fee (GWEI) | 6.774340574 |
| Priority fee (GWEI) | 2.5 |
| Total gas fee | 0.005881 GoerliETH |
| Max fee per gas | 0.000000015 GoerliETH |
| Total | 0.00588054 GoerliETH |

Figure 2: contract of part 2

Contract deployment X

Status [View on block explorer](#)
Confirmed [Copy transaction ID](#)

From  0x12B...3d5c  To New contract

Transaction

| | |
|---------------------|-----------------------|
| Nonce | 5 |
| Amount | -0 GoerliETH |
| Gas Limit (Units) | 649895 |
| Gas Used (Units) | 649895 |
| Base fee (GWEI) | 13.024915753 |
| Priority fee (GWEI) | 2.5 |
| Total gas fee | 0.01009 GoerliETH |
| Max fee per gas | 0.000000027 GoerliETH |
| Total | 0.01008957 GoerliETH |

Figure 3: contract with customSend code included

Similirlally, executing sell function from part 1 needed the least units of gas (35546), from part 2 (44418) and part 3 (40356). Expectedly, that means that the deployment of a contract that is linked with the deployed instance of the library needs fewer amounts of gas units than copying the code into the contract and deploying it, however using the sell function of the contract that is linked with the deployed instance of library needed more gas than the contract with the copied

code, since it needs additional resources to call the function on the deployed library and execute its code.



| Sell | | × |
|--|-----------------------|--|
| Status | | View on block explorer |
| Confirmed | | Copy transaction ID |
| From | | To |
|  0x12B...3d5c | |  0x99E...b5d4 |
| Transaction | | |
| Nonce | 2 | |
| Amount | -0 GoerliETH | |
| Gas Limit (Units) | 42529 | |
| Gas Used (Units) | 35546 | |
| Base fee (GWEI) | 23.369009296 | |
| Priority fee (GWEI) | 2.5 | |
| Total gas fee | 0.00092 GoerliETH | |
| Max fee per gas | 0.000000045 GoerliETH | |
| Total | 0.00091954 GoerliETH | |

Figure 4: contract of part 1



| Sell | | × |
|--|-----------------------|--|
| Status | | View on block explorer |
| Confirmed | | Copy transaction ID |
| From | | To |
|  0x12B...3d5c | |  0x4F0...7c7B |
| Transaction | | |
| Nonce | 4 | |
| Amount | -0 GoerliETH | |
| Gas Limit (Units) | 56081 | |
| Gas Used (Units) | 44418 | |
| Base fee (GWEI) | 7.617231891 | |
| Priority fee (GWEI) | 2.5 | |
| Total gas fee | 0.000449 GoerliETH | |
| Max fee per gas | 0.000000018 GoerliETH | |
| Total | 0.00044939 GoerliETH | |

Figure 5: contract of part 2



| Sell | | × |
|--|-----------------------|--|
| Status | | View on block explorer |
| Confirmed | | Copy transaction ID |
| From | | To |
|  0x538...222b | |  0x99A...f7E8 |
| Transaction | | |
| Nonce | 10 | |
| Amount | -0 GoerliETH | |
| Gas Limit (Units) | 52097 | |
| Gas Used (Units) | 40356 | |
| Base fee (GWEI) | 10.963897943 | |
| Priority fee (GWEI) | 2.5 | |
| Total gas fee | 0.000543 GoerliETH | |
| Max fee per gas | 0.000000029 GoerliETH | |
| Total | 0.00054335 GoerliETH | |

Figure 6: contract with customSend code included

Part 3

A way to use a public-key encryption in a way to associate each Mint operation with users ID can be the following.

1. The owner of the contract creates a public key and private key. The public key is stored on the contract, while the private key is hidden.
2. The owner uses a key generator algorithm that generates zk-SNARK proving key s_k and verification key v_k . He stores the verification key on the contract and also shares it with the regulator which can later check that the KYC was performed and the zk-proof is associated with the user's address.
3. The user encrypts his ID with the owner's public key and sends it to the contract which stores the encrypted file $C = Enc_{public}(ID)$.
4. The owner uses his private key to decrypt the sent file: $ID = Dec_{private}(C)$. He does that locally, outside of the blockchain.
5. Owner takes the proving key s_k , an ID file of the user that will be used as a witness and user's ethereum address that will be used as a public input; he then uses an algorithm P to generate zk-SNARK proof: $proof = P(s_k, address, ID)$.
6. Owner passes the proof along with other parameters to the contract, which verifies that the proof is correct by using a verifying algorithm: $V(v_k, address, proof) == true$.

Implementation on the contract.

To implement the above idea into the smart contract from part 2, the following would need to be added:

- owner's public key which a user can use to encrypt the ID

```
bytes public publicKey;
```

- owner's verification key which will be used to verify the zk-proof

```
bytes public verificationKey;
```

- a struct used to store the encrypted ID and a proof

```
struct IdentityDocument {  
    bytes encryptedDocument;  
    bytes proof;  
}
```


- a mapping that associates each user with the encrypted ID and zk-proof

```
mapping(address => IdentityDocument) IDs;
```

- a function that allows the user to send the encrypted ID and store it

```
function sendEncryptedID(bytes memory userID) public {
    IDs[msg.sender].encryptedDocument = userID;
}
```

- an algorithm that can verify the zk-proof (e.g., through a library)

```
function verify(bytes memory _proof, address user) internal view returns (bool isVerified){
    // check the verification
    return isVerified;
}
```

- additional check statements for verifying the zk-proof in the mint function

```
bytes memory _ID = IDs[to].proof;
if(!verify(_ID, to)){
    revert NotVerified();
}
```

such that final mint function would look like:

```
function mint(
    address to,
    uint256 value
) public onlyOwner returns (bool succesfullMint) {
    bytes memory _ID = IDs[to].proof;
    if(!verify(_ID, to)){
        revert NotVerified();
    }
    if(to == address(0)){
        revert ZeroAddress();
    }
    if(address(this).balance - _supply * PRICE < value * PRICE) {
        revert LowMint();
    }
    _supply += value;
    balances[to] += value;
    emit Mint(to, value);
    return true;
}
```

- A regulator can at any time ask the owner of the contract to provide a proof that KYC was completed according to the rules. An owner can then provide the zk-proof and an address of the user which a regulator can verify.

Thoughts and variations.

The process described above requires the owner and the user to perform some computation locally, outside the chain. Therefore, the process is not completely on chain. In my opinion, with the current ethereum's smart contracts constraints, it would be highly impractical to implement the process in a way that it would be completely on chain.

Below are briefly described some other variations that perhaps would allow the process to need less computation and performing locally.

- A variation could be that the user from the beginning sent the zk-proof he created and the verification key. However, there would need to exist a kind of storage (e.g. a database) which an owner could use to verify that the actual proof is correct and not forged by the user. It would also force the contract into storing more keys.
 - The user could also sign the ID before encrypting it which would provide additional integrity of the file, since it would not only prove the possession of the ID but also a possession of the signing key of the user.
 - There could exist some intermediary service that would take user's ID, compute the zk-proof and send it to the chain. It could allow additional integrity (e.g., by checking biometrics of the user at the time of providing the ID).
-

Appendix

Code of part 1.

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity 0.8.7;
3
4 contract B32245_part1 {
5
6     //=====
7     // Errors
8     //=====
9
10    error NotOwner();
11    error ZeroAddress();
12    error LowBalance();
13    error LowMint();
14
15    //=====
16    // Data Structures
17    //=====
18
19    address payable public owner;
20
21    mapping(address => uint256) private balances;
22
23    uint256 private _supply;
24
25    string private _tokenName;
26    string private _tokenSymbol;
27
28    uint128 constant private PRICE = 600;
29
30    //=====
31    // Events
32    //=====
33
34    event Transfer(address indexed from, address indexed to, uint256 value);
35    event Mint(address indexed to, uint256 value);
36    event Sell(address indexed from, uint256 value);
37
```

```
//=====
// Functions
//=====

function totalSupply() public view returns (uint256 amount) {
    return _supply;
}

function balanceOf(address _account) public view returns (uint256 amount) {
    return balances[_account];
}

function getName() public view returns (string memory name) {
    return _tokenName;
}

function getSymbol() public view returns (string memory symbol) {
    return _tokenSymbol;
}

function getPrice() public view returns (uint128 price) {
    return PRICE;
}

function transfer(
    address to,
    uint256 value
) public returns (bool succesfullTransfer) {
    if(to == address(0)){
        revert ZeroAddress();
    }
    uint256 senderBalance = balances[msg.sender];
    if(senderBalance < value){
        revert LowBalance();
    }
    balances[msg.sender] = senderBalance - value;
    balances[to] = balances[to] + value;
    emit Transfer(msg.sender, to, value);
    return true;
}
```

```

function mint(
    address to,
    uint256 value
) public onlyOwner returns (bool succesfullMint) {
    if(to == address(0)){
        revert ZeroAddress();
    }
    if(address(this).balance - _supply * PRICE < value * PRICE) {
        revert LowMint();
    }
    _supply += value;
    balances[to] += value;
    emit Mint(to, value);
    return true;
}

function sell(uint256 value) public returns (bool succesfullSell) {
    uint256 senderBalance = balances[msg.sender];
    if(senderBalance < value){
        revert LowBalance();
    }
    balances[msg.sender] = senderBalance - value;
    _supply -= value;
    emit Sell(msg.sender, value);
    payable(msg.sender).transfer(value * PRICE);
    return true;
}

function close() public onlyOwner {
    selfdestruct(payable(owner));
}

receive() external payable {}

```

```

//=====
// Constructor
//=====

constructor(string memory name, string memory symbol) {
    owner = payable(msg.sender);
    _tokenName = name;
    _tokenSymbol = symbol;
}

//=====
// Modifier
//=====

modifier onlyOwner() {
    if(msg.sender != owner){
        revert NotOwner();
    }
    _;
}

```

Transaction history of part 1.

I deployed the contract with token name "token" and symbol "TKN" to the Goerli network on 0x99EB9823CfBdbe9ce03dcFC07A76cAb1c284b5d4 from address 0xb892aEC81166aaeA7B13054C7908d8B64A9C27Da. Then, I sent 1800 wei from 0x12BD2c44Aad7069c6be493172191CB40e0C83d5c. After, I minted three tokens from 0xb892aEC81166aaeA7B13054C7908d8B64A9C27Da to

0x12BD2c44Aad7069c6be493172191CB40e0C83d5c. Later, I transferred one token from 0x12BD2c44Aad7069c6be493172191CB40e0C83d5c to

0x538CD595357D881783b68f07d8129B929EF2222b and sold the remaining 2 tokens (from 0x12BD2c44Aad7069c6be493172191CB40e0C83d5c).

```

"accounts": {
  "account{0}": "0x12BD2c44Aad7069c6be493172191CB40e0C83d5c"
},
"linkReferences": {},
"transactions": [
  {
    "timestamp": 1672830802753,
    "record": {
      "value": "0",
      "inputs": "(string,string)",
      "parameters": [
        "token",
        "TKN"
      ],
      "name": "",
      "type": "constructor",
      "abi": "0x96e9b4c9796d0ee1b5abc84a99a76ba852eca1f45367aefa99182fee4123de8c",
      "contractName": "B32245_part1",
      "bytecode": "60806040523480156200001157600080fd5b5060405162000b5f38038062000b5f833981016040819052",
      "linkReferences": {},
      "from": "account{0}"
    }
  },
  {
    "timestamp": 1672830901223,
    "record": {
      "value": "1800",
      "inputs": "()",
      "parameters": [],
      "type": "receive",
      "to": "created{1672830802753}",
      "abi": "0x96e9b4c9796d0ee1b5abc84a99a76ba852eca1f45367aefa99182fee4123de8c",
      "from": "account{0}"
    }
  }
],

```

```

{
  "timestamp": 1672830946775,
  "record": {
    "value": "0",
    "inputs": "(address,uint256)",
    "parameters": [
      "0x12BD2c44Aad7069c6be493172191CB40e0C83d5c",
      "3"
    ],
    "name": "mint",
    "type": "function",
    "to": "created{1672830802753}",
    "abi": "0x96e9b4c9796d0ee1b5abc84a99a76ba852eca1f45367aefa99182fee4123de8c",
    "from": "account{0}"
  }
},
{
  "timestamp": 1672830998580,
  "record": {
    "value": "0",
    "inputs": "(address,uint256)",
    "parameters": [
      "0x538CD595357D881783b68f0d8129B929EF2222b",
      "1"
    ],
    "name": "transfer",
    "type": "function",
    "to": "created{1672830802753}",
    "abi": "0x96e9b4c9796d0ee1b5abc84a99a76ba852eca1f45367aefa99182fee4123de8c",
    "from": "account{0}"
  }
},
{
  "timestamp": 1672831034233,
  "record": {
    "value": "0",
    "inputs": "(uint256)",
    "parameters": [
      "2"
    ],
    "name": "sell",
    "type": "function",
    "to": "created{1672830802753}",
    "abi": "0x96e9b4c9796d0ee1b5abc84a99a76ba852eca1f45367aefa99182fee4123de8c",
    "from": "account{0}"
  }
}
}

```

```

    ],
    "abis": {
      "0x96e9b4c9796d0ee1b5abc84a99a76ba852eca1f45367aefa99182fee4123de8c": [
        {
          "inputs": [],
          "name": "close",
          "outputs": [],
          "stateMutability": "nonpayable",
          "type": "function"
        },
        {
          "inputs": [
            {
              "internalType": "string",
              "name": "name",
              "type": "string"
            },
            {
              "internalType": "string",
              "name": "symbol",
              "type": "string"
            }
          ],
          "name": "",
          "stateMutability": "nonpayable",
          "type": "constructor"
        },
        {
          "inputs": [],
          "name": "LowBalance",
          "type": "error"
        },
        {
          "inputs": [],
          "name": "LowMint",
          "type": "error"
        }
      ],
    }
  },

```

```

    {
      "inputs": [
        {
          "internalType": "address",
          "name": "to",
          "type": "address"
        },
        {
          "internalType": "uint256",
          "name": "value",
          "type": "uint256"
        }
      ],
      "name": "mint",
      "outputs": [
        {
          "internalType": "bool",
          "name": "succesfullMint",
          "type": "bool"
        }
      ],
      "stateMutability": "nonpayable",
      "type": "function"
    },
    {
      "inputs": [],
      "name": "NotOwner",
      "type": "error"
    },
    {
      "inputs": [],
      "name": "ZeroAddress",
      "type": "error"
    }
  ],

```

```
{
  "anonymous": false,
  "inputs": [
    {
      "indexed": true,
      "internalType": "address",
      "name": "to",
      "type": "address"
    },
    {
      "indexed": false,
      "internalType": "uint256",
      "name": "value",
      "type": "uint256"
    }
  ],
  "name": "Mint",
  "type": "event"
},
{
  "inputs": [
    {
      "internalType": "uint256",
      "name": "value",
      "type": "uint256"
    }
  ],
  "name": "sell",
  "outputs": [
    {
      "internalType": "bool",
      "name": "succesfullSell",
      "type": "bool"
    }
  ],
  "stateMutability": "nonpayable",
  "type": "function"
},
},
```

```
{
  "anonymous": false,
  "inputs": [
    {
      "indexed": true,
      "internalType": "address",
      "name": "from",
      "type": "address"
    },
    {
      "indexed": false,
      "internalType": "uint256",
      "name": "value",
      "type": "uint256"
    }
  ],
  "name": "Sell",
  "type": "event"
},
{
  "inputs": [
    {
      "internalType": "address",
      "name": "to",
      "type": "address"
    },
    {
      "internalType": "uint256",
      "name": "value",
      "type": "uint256"
    }
  ],
  "name": "transfer",
  "outputs": [
    {
      "internalType": "bool",
      "name": "succesfullTransfer",
      "type": "bool"
    }
  ],
  "stateMutability": "nonpayable",
  "type": "function"
},
},
```

```
{
  "anonymous": false,
  "inputs": [
    {
      "indexed": true,
      "internalType": "address",
      "name": "from",
      "type": "address"
    },
    {
      "indexed": true,
      "internalType": "address",
      "name": "to",
      "type": "address"
    },
    {
      "indexed": false,
      "internalType": "uint256",
      "name": "value",
      "type": "uint256"
    }
  ],
  "name": "Transfer",
  "type": "event"
},
{
  "stateMutability": "payable",
  "type": "receive"
},
{
  "inputs": [
    {
      "internalType": "address",
      "name": "_account",
      "type": "address"
    }
  ],
  "name": "balanceOf",
  "outputs": [
    {
      "internalType": "uint256",
      "name": "amount",
      "type": "uint256"
    }
  ],

```

```
],
  "stateMutability": "view",
  "type": "function"
},
{
  "inputs": [],
  "name": "getName",
  "outputs": [
    {
      "internalType": "string",
      "name": "name",
      "type": "string"
    }
  ],
  "stateMutability": "view",
  "type": "function"
},
{
  "inputs": [],
  "name": "getPrice",
  "outputs": [
    {
      "internalType": "uint128",
      "name": "price",
      "type": "uint128"
    }
  ],
  "stateMutability": "view",
  "type": "function"
},
{
  "inputs": [],
  "name": "getSymbol",
  "outputs": [
    {
      "internalType": "string",
      "name": "symbol",
      "type": "string"
    }
  ],
  "stateMutability": "view",
  "type": "function"
},

```



```
{
  "inputs": [],
  "name": "owner",
  "outputs": [
    {
      "internalType": "address payable",
      "name": "",
      "type": "address"
    }
  ],
  "stateMutability": "view",
  "type": "function"
},
{
  "inputs": [],
  "name": "totalSupply",
  "outputs": [
    {
      "internalType": "uint256",
      "name": "amount",
      "type": "uint256"
    }
  ],
  "stateMutability": "view",
  "type": "function"
}
]
```

Etherscan Goerli Testnet Network

All Filters Search by Address / Txn Hash / Block / Token / Ens

Home Blockchain Tokens Misc Goerli

Contract 0x99EB9823CfBdb9ce03dcFC07A76cAb1c284b5d4

Contract Overview

Balance: 600 wei

More Info

My Name Tag: Not Available

Contract Creator: 0xb892aec81166aeea7b... at txn 0x48c791e2d5a74e6da7...

Transactions Internal Txns Erc20 Token Txns Contract Events

Latest 5 from a total of 5 transactions

| Txn Hash | Method | Block | Age | From | To | Value | Txn Fee |
|-------------------------|-------------------|---------|------------|-------------------------|-----------------------|---------|------------|
| 0x8410bd7f911533c051... | Sell | 8254903 | 4 mins ago | 0x12bd2c44aad7069c6b... | 0x99eb9823cfdb9ce0... | 0 Ether | 0.00091953 |
| 0x29577690027309343e... | Transfer | 8254901 | 5 mins ago | 0x12bd2c44aad7069c6b... | 0x99eb9823cfdb9ce0... | 0 Ether | 0.00121706 |
| 0xddb5a7556c3bea8d57... | Mint | 8254898 | 6 mins ago | 0xb892aec81166aeea7b... | 0x99eb9823cfdb9ce0... | 0 Ether | 0.00168671 |
| 0xd33b775a9b6248cecd... | Transfer | 8254894 | 7 mins ago | 0x12bd2c44aad7069c6b... | 0x99eb9823cfdb9ce0... | 0 Ether | 0.00046971 |
| 0x48c791e2d5a74e6da7... | Contract Creation | 8254889 | 8 mins ago | 0xb892aec81166aeea7b... | Contract Creation | 0 Ether | 0.01213049 |

[Download CSV Export]

Code of part 2.

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.7;

import './customLib.sol';

contract B32245_part2 {

    //=====
    // Errors
    //=====

    error NotOwner();
    error ZeroAddress();
    error LowBalance();
    error LowMint();

    //=====
    // Data Structures
    //=====

    address payable public owner;

    mapping(address => uint256) private balances;

    uint256 private _supply;

    string private _tokenName;
    string private _tokenSymbol;

    uint128 constant private PRICE = 600;

    //=====
    // Events
    //=====

    event Transfer(address indexed from, address indexed to, uint256 value);
    event Mint(address indexed to, uint256 value);
    event Sell(address indexed from, uint256 value);
```

```
//=====
// Functions
//=====

function totalSupply() public view returns (uint256 amount) {
    return _supply;
}

function balanceOf(address _account) public view returns (uint256 amount) {
    return balances[_account];
}

function getName() public view returns (string memory name) {
    return _tokenName;
}

function getSymbol() public view returns (string memory symbol) {
    return _tokenSymbol;
}

function getPrice() public view returns (uint128 price) {
    return PRICE;
}

function transfer(
    address to,
    uint256 value
) public returns (bool succesfullTransfer) {
    if(to == address(0)){
        revert ZeroAddress();
    }
    uint256 senderBalance = balances[msg.sender];
    if(senderBalance < value){
        revert LowBalance();
    }
    balances[msg.sender] = senderBalance - value;
    balances[to] = balances[to] + value;
    emit Transfer(msg.sender, to, value);
    return true;
}
```

```

function mint(
    address to,
    uint256 value
) public onlyOwner returns (bool succesfullMint) {
    if(to == address(0)){
        revert ZeroAddress();
    }
    if(address(this).balance - _supply * PRICE < value * PRICE) {
        revert LowMint();
    }
    _supply += value;
    balances[to] += value;
    emit Mint(to, value);
    return true;
}

function sell(uint256 value) public returns (bool succesfullSell) {
    uint256 senderBalance = balances[msg.sender];
    if(senderBalance < value){
        revert LowBalance();
    }
    balances[msg.sender] = senderBalance - value;
    _supply -= value;
    emit Sell(msg.sender, value);
    customLib.customSend(value * PRICE, msg.sender);
    return true;
}

function close() public onlyOwner {
    selfdestruct(payable(owner));
}

receive() external payable {}

```

```

//=====
// Constructor
//=====

constructor(string memory name, string memory symbol) {
    owner = payable(msg.sender);
    _tokenName = name;
    _tokenSymbol = symbol;
}

//=====
// Modifier
//=====

modifier onlyOwner() {
    if(msg.sender != owner){
        revert NotOwner();
    }
    _;
}


```

Transaction history of part 2.

I deployed the contract with the token name "token2" and symbol "TKN2" to the Goerli network on address 0x4F0Ed14f0Ec5f07F2a1D736d8b5317E099BF7c7B from address 0x538CD595357D881783b68f07d8129B929EF2222b. Then, I sent 1800 wei from 0x12BD2c44Aad7069c6be493172191CB40e0C83d5c. After, I minted three tokens from 0x538CD595357D881783b68f07d8129B929EF2222b to

0x12BD2c44Aad7069c6be493172191CB40e0C83d5c. Later, I sold three tokens (from 0x12BD2c44Aad7069c6be493172191CB40e0C83d5c).


```
24      },
25      "autoDeployLib": true
26    },
27    "rinkeby:4": {
28      "linkReferences": {
29        "contracts/customLib.sol": {
30          "customLib": "<address>"
31        }
32      },
33      "autoDeployLib": true
34    },
35    "kovan:42": {
36      "linkReferences": {
37        "contracts/customLib.sol": {
38          "customLib": "<address>"
39        }
40      },
41      "autoDeployLib": true
42    },
43    "goerli:5": {
44      "linkReferences": {
45        "contracts/customLib.sol": {
46          "customLib": "0x9DA4c8B1918BA29eBA145Ee3616BCDFcFAA2FC51"
47        }
48      },
49      "autoDeployLib": false
50    },
51    "Custom": {
52      "linkReferences": {
53        "contracts/customLib.sol": {
54          "customLib": "<address>"
55        }
56      },
57      "autoDeployLib": true
58    }
59  }
```

 Etherscan

All Filters Search by Address / Txn Hash / Block / Token / Ens

Goerli Testnet Network

Home Blockchain Tokens Misc Goerli

 Contract 0x4F0Ed14f0Ec5f07F2a1D736d8b5317E099BF7c7B

Contract Overview

Balance: 0 Ether

More Info

My Name Tag: Not Available

Contract Creator: [0x538cd595357d881783...](#) at txn [0x77de43f87b5d97d4fe0...](#)

Transactions Erc20 Token Txns Contract Events

Latest 4 from a total of 4 transactions

| Txn Hash | Method | Block | Age | From | To | Value | Txn Fee |
|--|----------------------------|---------|-------------|---|---|---------|------------|
| 0xb166a7ac854c80a510... | Sell | 8255610 | 6 secs ago | 0x12bd2c44aad7069c6b... | 0x4f0ed14f0ec5f07f2a1d... | 0 Ether | 0.00044938 |
| 0xa9a984da727b4dac64... | Mint | 8255609 | 30 secs ago | 0x538cd595357d881783... | 0x4f0ed14f0ec5f07f2a1d... | 0 Ether | 0.00071241 |
| 0x8b4951b3ad5004991d... | Transfer | 8255608 | 54 secs ago | 0x12bd2c44aad7069c6b... | 0x4f0ed14f0ec5f07f2a1d... | 0 Ether | 0.00020408 |
| 0x77de43f87b5d97d4fe0... | 0x60806040 | 8255603 | 2 mins ago | 0x538cd595357d881783... | Contract Creation | 0 Ether | 0.00588054 |

[Download CSV Export]

A contract address hosts a smart contract, which is a set of code stored on the blockchain that runs when predetermined conditions are met. Learn more about addresses in our [Knowledge Base](#).