Blockchains and Distributed Ledgers Assignment4

Qi Zhou

January 2021

1 Contract Design.

My contract allows only one process of fair swap at a time. Two users should commit their swapping details along with 1 ether input before any fair swap to happen. The first one who committed is deemed as User A and the second is User B. After both users are committed, there is a cooling period long enough for both of them to perform token transfer from their account to the fair swap contract address. After the cooling period is an announce period saved specifically for User B. User B can call announce for contract to determine whether or not the current status is appropriate for fair swap (fair swap will be executed if appropriate) and whether any user shall be punished with 1 ether deprivation. After the announce period for B is another announce period for A, who can announce for the same purpose. It is possible that both of them do not announce. No matter it's B or A or nobody announced, we end up in the final period of claim. Anyone can call the claim for the current round of fair swap to take care of the remaining ethereum and token transfer. This claim can only be called once, which will reset the contract to start over again.

Inner variables contractStatus and playerStatus are to distinguish different stages of the contract (see Table 1). There are exactly five contract status. For each round of fair swap, the contract status always starts from #00 and ends in #20. The contract status transition possibilities and conditions are shown in Table 2. To assist determining the timing condition, other inner variables are used — commitStart(record the first commitment time), commitGap(predetermined time period for the second commitment), transferStart(record the second commitment, the start of the cooling time for token transfer), transferGap(predetermined time period for token transfer), BAnnounceGap(predetermined time period for B to announce, following the transferGap), AAnnounceGap(predetermined time period for A to announce, following the BAnnounceGap), claimStart(record the announce time, after which claim is allowed — not applicable when no second user commit).

StatusID	contractStatus	playerStatus	Explanation
#00	0	0	Initial status, no one commit
#01	0	1	Only one user committed
#10	1	-	Both users committed, no announcement
#11	2	-	Announcement made
#20	10	-	Claimed

Table 1: Status Explanation for Fair Swap contract.

StatusID from	StatusID to	Explanation	Timing Condition
#00	#01	The first user committed	-
#01	#10	The second user committed	Within commit period
#10	#11	User A or B announced	Within their own announce period
#01	#20	No second user commit therefore claim	After commit period
#10	#20	No one announced therefore claim	After both announce period
#11	#20	someone announced therefore claim	After both announce period

Table 2: Status Transition for Fair Swap contract.

In the commit operation, each user input the following information — the address of the other user, the amount of token for me and the other user, the token contract address for me and the other user. Therefore, there are many inner variables for storing relevant information — playerA and playerB for the address of two users, AAmount and BAmount for the token amount, AContractAddr and

BContractAddr for the token contract address, AContract and BContract for the instance of token contract. The inputs in the first commitment are assumed correct and are saved in variables. The second commitment is reckoned as an agreement. It's only when the information in two commitment are matched that it's allowed into the next stage. Each user also input 1 ether into the fair swap contract, which will be redistributed according their honesty.

In the announce operation, user B is guided to announce first. If not, user A will have the chance to announce. In order to make an announcement, the user must have transferred the consented token to the fair swap contract address. Therefore, there are four possible outcome if an announcement is made. Inner variables AAllowance and BAllowance are used for storing the amount of ethereum for A and B to claim after the announcement. Since the input is 1 ether each, the possible allowance is 0, 1 or 2 (later in the unit of ether). In an announcement, two allowances are set according to user's honesty and the token will be transferred accordingly, which includes fair swap.

In the claim operation, one call can transfer ethereum back to both users. Three difference claim conditions bear difference. From #01 to #20, only the 1 ether of the first user is processed. From #10 to #20, we consider that perhaps both users agreed on aborting the swap. Therefore, no matter the token balance state, all tokens are going back to their origin and the 1 ether is sent back to its origin (in quantity). From #11 to #20, the token transfer is already done in the former announce operation. Therefore, it's only for withdrawing the allowance amount of ethereum.

Above all are the basic ideas for this fair swap contract. The detailed punishment plan will be illustrated in Section 3 as a part of fair swap mechanism.

2 Gas Analysis and Security Analysis.

Here (See Table 3) I list the gas cost for deploying and interacting with this Fair Swap contract. It corresponds to the transaction history shown in Section 4.

Interaction	Gas Amount	Executor
deploy	1,806,407	Constructor
commit	169,908	User A
commit	92,760	User B
announce	90,202	(Ideal)User B
claim	45,933	Anyone

Table 3: Gas cost for deploying and interacting with Fair Swap contract.

Certain variables with small value like contractStatus are set as type uint8 and announced earlier together to be put into the same storage block. The use of constant saves gas. Variables that are not going to be used are deleted to save storage and cost.

The first commit is always larger than the second. It also approximate the summation of the gas for second commit and announce. Therefore, it is ideal and guided by the design that B should do the announcement.

For security, we use SafeMath to tackle overflow and underflow problem.

The transition of contract status ensures that each operation can only be performed under certain condition. The status is normally modified at the start of an execution to avoid reentrancy attack. The order of execution is also used for preventing reentrancy benefits.

Notably, the timing condition is also essential. It not only ensures enough time for honest party to transfer tokens, but also prevent the malicious user tricking the contract into believing the other part is dishonest or taking up the contract just to block other users etc. The former trick can be further explained that if the malicious user has already sent the token and found out that the other user hasn't transferred in time or it's on progress, he may call the announce if no timing condition is set. In this announcement, it appears that the other user is cheating and not transferring token. Therefore, the 1 ether punishment will be executed and the malicious user will lose nothing and earn a whole new ether.

For safer execution, we adopted pull over push, asking user or others to help pull the ethereum back. The reason to include anyone to help claim is that there is no auto stop for each round of the game and it is only for one round at a time. When it reaches the commit stage, anyone can have it prepared for the next round without obstacle.

Since that the commit operation ask for the absolute match of fair swap information, there is no need to worry about the front-running problem.

3 How I Ensured Fairness.

Fairness lies in the functionality and the balanced gas. In functionality, we ensure that 1) two users either successfully swap the token together or not losing token at all. 2) if there's one party dishonest, we shall be able to punish him. In balanced gas, we ensure that 1) the gas cost in execution is approximated. 2) there's way to punish user if he try to escape gas payment.

The tokens agreed on are required to be sent to the fair swap contract address before any swap to take place. This ensures that the tokens to be swapped are secured and can not be further deducted actively by the user. When the fair swap is available, the contract actively transfer the token in an one-shot manner.

Table 4 shows the token and ethereum transfer scheme under different announcement status. Being honest means that the fair swap contract checked that the user has transferred enough token for swap.

Announcement Status	A Honest	B Honest	Token Transfer	Ethereum Transfer	Notes
B announced	True	True	Fair Swap	Back to Origin	
B announced	False	True	Back to Origin	Punish A	A Not Transfer
A announced	True	True	Fair Swap	Punish B	B Escape Gas
A announced	True	False	Back to Origin	Punish B	B Not Transfer
no announce	True	True	Back to Origin	Back to Origin	
no announce	False	False	Back to Origin	Back to Origin	
no announce	True	False	Back to Origin	Back to Origin	
no announce	False	True	Back to Origin	Back to Origin	

Table 4: Token and Ethereum Transfer Scheme under Announcement Status

Since that enough time is given for each user to transfer token, the balance result is deterministic when announcement is made. Each user has to be honest to declare the dishonesty of the other. Since that only when announce is called then a fair swap can be executed, we allow both users a period of time to announce, which presents a fair chance for them to claim their reward for being honest and to punish the malicious other.

When no one announced, we should not decide upon the current honesty of user. That is because the token transfer is still allowed during the announcement period. If user appears to be honest, why wouldn't he announce in his own time. No matter he missed the chance or purposefully skip announcement either for proper reason of ditching current round of fair swap or to trick the contract into sending him reward for being honest, we can not distinguish without further information like transfer time. Therefore, it's safe to have everything back to its origin.

To assign gas cost evenly between two users, I consider the order of interaction. The first user committed always has a large gas cost which can be around the summation of the second commitment and announcement. Therefore, I fix the announce order to seduce the second user to announce. Whether he announce is also driven by the punishment will be given to him if he try to escape the announce gas fee as is shown in Table 4. It may also be caused by mere bad luck of late transfer. However, two minutes is long enough and no further tolerance shall be granted. In this way, the ideal gas cost calculated according to Table 3 is 169908 for user A and 182962 for user B. The gas difference is 13054. If user B cheat gas, higher price will be paid with the value of 1 ether.

The current gas doesn't include claim operation though. Generally we consider it's the user who is more eager to get money out (probably with more benefit) or to start a new round that will execute the function. If it is between user A and user B, the gas difference will be 32879 or 58987.

4 Successful Fair Swap Transaction History on Ropsten.

User A address:
0x6BaC7d59992d80Df6E9A52D9C82AC0aE8f7e2543

User B address:
0xb354F445455e9a20B50C0F5C1649801010D13922

Constructor construct FairSwap:
0x0a850cce0a6f70edb2a1928877c71ba486593d543bf0539a975ce49523bfbe31

User A buy 10 token in C1, User B buy 8 token in C2;

User A commit (User B addr, 5, 7, C1 addr, C2 addr):
0x7fce5da67fb0d3ddabc0d53f0866f3d68271cc294eec2c3cc47c9d54e0b01026

User B commit (User A addr, 7, 5, C2 addr,c1 addr):
0x4091afbfc9702cf903847dc72fbce638eee51dd489d9e85e4d9b8ff0ffa79bde

User A transfer 5 on C1 to FairSwap addr, User B transfer 7 on C2 to FairSwap addr;

User B announce:
0x3277ce5304d689cd8d4ed708614d36512d3a9a97542f12c72723b5660888b542

User B claim:
0x16309c42a8461be10fd46710fff0bbd88442ce8705d732dd48b74f44cd3cd970

5 My Contract Code.

```
pragma solidity ^0.5.0;

import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v2.5.0/contracts/math/
SafeMath.sol";

contract FairSwap{

    using SafeMath for uint256;

    uint8 private contractStatus;
    uint8 private playerStatus;
    uint8 private AAllowance;
    uint8 private BAllowance;

    address private playerA;
    address private playerB;

    uint256 private AAmount;
    uint256 private BAmount;
    address private AContractAddr;
    address private BContractAddr;

    uint256 private commitStart;
    uint256 constant private commitGap = 60;
    uint256 private transferStart;
    uint256 constant private transferGap = 60;
    uint256 constant private BAnnounceGap = 60;
    uint256 constant private AAnnounceGap = 60;
    uint256 private claimStart;

    Moireum AContract;
    Moireum BContract;

    constructor () public{ }

    function commit(address counterpart, uint256 myamount, uint256 hisamount,
    address mycontractaddr, address hiscontractaddr) public payable{
        require(msg.value == 1 ether && contractStatus == 0 && msg.sender != counterpart);
        if(playerStatus == 0){
```

```
        commitStart = now;
        playerStatus = 1; // one commit
        playerA = msg.sender;
        playerB = counterpart;
        AAmount = myamount;
        BAmount = hisamount;
        AContractAddr = mycontractaddr;
        BContractAddr = hiscontractaddr;
    } else if(playerStatus == 1){
        require( now < commitStart.add(commitGap) && playerA == counterpart
        && playerB == msg.sender && AAmount == hisamount && BAmount == myamount
        && AContractAddr == hiscontractaddr && BContractAddr == mycontractaddr);
        contractStatus = 1;
        playerStatus = 0; // reset nothing for contract status 1
        transferStart = now;
        AContract = Moireum(AContractAddr);
        BContract = Moireum(BContractAddr);
        delete commitStart;
    } // first commit more gas. approxi commit 1 should be announce 1. and 2 be 2.
}

function announce() public{ // dont care about token stuck
    require(contractStatus == 1 && now > transferStart.add(transferGap));
    contractStatus = 2; // outcome can be determined, can withdraw
    claimStart = now;
    if(msg.sender == playerB){
        require(now < transferStart.add(transferGap.add(BAnnounceGap))
        && BContract.getBalance() >= BAmount);
        // B announce scenario:
        if(AContract.getBalance() >= AAmount){
            // FairSwap
            AAllowance = 1;
            BAllowance = 1;
            AContract.transfer(playerB, AAmount);
            BContract.transfer(playerA, BAmount);
        } else{ //(AContract.getBalance() < AAmount)
            // A dishonest, B can pull 2, B token back
            BAllowance = 2;
            BContract.transfer(playerB, BAmount);
        }
    } else if(msg.sender == playerA){
        require(now > transferStart.add(transferGap.add(BAnnounceGap))
        && now < transferStart.add(transferGap.add(BAnnounceGap.add(AAnnounceGap)))
        && AContract.getBalance() >= AAmount);
        // A announce scenario: (meaning that B didnt announce)
        if(BContract.getBalance() < BAmount){
            // B dishonest, A can pull 2, A token back
            AAllowance = 2;
            AContract.transfer(playerA, AAmount);
        } else{ //(BContract.getBalance() >= BAmount)
            // FairSwap, yet A can pull 2 (since that B cheat on gas fee)
            AAllowance = 2;
            AContract.transfer(playerB, AAmount);
            BContract.transfer(playerA, BAmount);
        }
    } else{
        revert();
    }
    delete transferStart;
}
```

```
    function claim() public{
        // anyone can call claim
        // if(contractStatus == 0 && playerStatus == 0){ nothing }
        if(contractStatus == 0 && playerStatus == 1 && now > commitStart.add(commitGap)){
            // A commit, B not commit, past commit gap, A can withdraw, reset status
            contractStatus = 10;
            address(uint160(playerA)).transfer(1 ether);
            resetContract();
        }
        // if(contractStatus == 1 && playerStatus == 0){ both committed, shouldnt withdraw }
        if(contractStatus == 1 && now > claimStart){
            // no one announce scenario:
            contractStatus = 10; // change status instead of using local variables
            if((AContract.getBalance() >= AAmount && BContract.getBalance() >= BAmount)){
                AContract.transfer(playerA, AAmount);
                BContract.transfer(playerB, BAmount);
            } else if(AContract.getBalance() < AAmount && BContract.getBalance() < BAmount){
            } else if(AContract.getBalance() >= AAmount){
                AContract.transfer(playerA, AAmount);
            } else{
                BContract.transfer(playerB, BAmount);
            }
            address(uint160(playerA)).transfer(1 ether);
            address(uint160(playerB)).transfer(1 ether);
            resetContract();
        }
        if(contractStatus == 2 && now > claimStart){
            // after announcement claim
            contractStatus = 10;
            address(uint160(playerA)).transfer(AAllowance * 1 ether);
            address(uint160(playerB)).transfer(BAllowance * 1 ether);
            resetContract();
        }
    }

    function resetContract() private{
        delete playerA;
        delete playerB;
        delete contractStatus;
        delete playerStatus;
        delete AAllowance;
        delete BAllowance;
        delete AAmount;
        delete BAmount;
        delete AContractAddr;
        delete BContractAddr;
        delete commitStart;
        delete transferStart;
        delete claimStart;
    }

    function getTime() view public returns(uint256 timenow){
        timenow = now;
    }

}
```