Final Term Peer to Peer Systems and Blockchains

Fausto F. Frasca 559482

May 2021

1 Part 1 - Ethereum e Smart Contracts

1.1 Cost in gas of the functions provided by the smart contract

To evaluate the cost in gas of the functions provided by smart contract it has been used the test file *1W_vs_1L.js* in the test directory. The test involves a yay voter that sends 50 eth and a nay voter that sends 1 eth and the gas price is 20000000000.

The cost in gas of the single functions is:

- *constructor*: 1.283.823
- *compute_envelope*: 23.052
- *cast_envelope*: 53.023
- *open_envelope*: 141.715
- *mayor_or_sayonara*: 69.141

Has been applied some variation about the quorum and souls sent by the voters to check the variation about the gas cost of the function mayor_or_sayonara. For do so, the tests *50W_vs_20L.js*, *1W_vs_1000L.js* and *50W_vs_50L.js* were used.

The results are:

- *50W_vs_20L.js*: 50 voters vote true and send 50 eth while 20 voters vote false and send 1 eth. The final gas cost is 250.383.
- *1W_vs_1000L.js*: 1 voter votes true and send 50 eth while 1000 voters vote false and send 1 wei. The final gas cost is 9.600.574.
- *50W_vs_50L.js*: 50 voters vote true and send 50 eth while 50 voters vote false and send the same amount of eth. The final gas cost is 536.551.

1.2 Security considerations and potential vulnerabilities related to the mayor_or_sayonara

The function's vulnerability is the reentrancy. A function or procedure is reentrant if its execution can be interrupted in the middle, it can be initiated over and both the runs complete without any error. An attacker can exploits this bug for steal the ethers in the contract account. For example, suppose that a smart contract contain a function that allows to withdraw the amount in the account's balance. The function first send the tokens to the caller and only after reset the balance of the account. So, the attacker, using this bug, call again the function before it terminate and reset the balance, receiving more tokens than he should receives.

To avoid this bug, the smart contract must first set the balance to zero and only then it sends the tokens. Another solution could be to use a flag to make sure whether the function has been called or not. This check must be done at the beginning of the function. Also the language, Solidity, provides some patch to this attack. In fact, is advisable to use the functions *send* and *transfer* to transfer the tokens, because they have a limited amount of 2.300 gas and this implies that the bugged function can be called unless all the fixed amount of gas is consumed. Instead, if *call.value* is used, you must manually limit the amount of gas.

This bug was exploited in the attack known as "Dao attack", where an attacker stole 3.6M of ethers in a few minutes, leading to an hard fork of the blockchain.

The function has also a gas limit that is equal at most to the block gas limit. This means that if the number of losers to refund is very high because of quorum required and it is necessary to "burn" more gas than block limit, this function wont be never mined. An attacker can exploit this limit. In fact, the attacker can deploys the smart contract to the network, conscious about what we explained before, the voters provide their vote but the election will be never completed because the call to mayor_or_sayonara require too much gas. A possible attacker can be a malicious mayor that can exploit this for know who are the snipers, who says that he/she votes for you but he/she lies.

The block gas limit in 2020 was more or less 10 million and, from the previous tests, refund 1000 losers required 9 million of gas more or less. This means that if the losers are 5000, for example, the attack can be executed. So it is required a new way to refund the losers.

This is not strictly a security issue, but it can be a problem and perhaps can be exploited by a larger attack.

1.3 Compute_envelope issue

This function could be a problem if the smart contract is deployed to the Ethereum network and the voters use another smart contract that call Mayor for vote, that we call Tmp in the next example. Suppose that Tmp has a non-view/pure function take as input the sigil, doblon and souls and call the function

compute_envelope. In this case, that function will be in a transaction and will be mined, allowing to anyone to know the votes before that all the envelopes are opened. This is a huge problem, because it allows to the current Mayor to falsify the elections checking the yay souls and making sure that yaySoul is strictly greater than naySoul.

2 Parte 2 - BITCOIN and the Lightning network

2.1 Bitcoin's countermeasures against double spending attack

A decentralised financial system has to solve problems that do not exist in the traditional electronic payment system. One of these is the double spending, that is solved in the traditional system thanks the central authorities like Visa and MasterCard, that ensure that their clients can spend only once their moneys. Instead, with a decentralized system an attacker can make two transactions with the same tokens and it's the network's task to avoid the problem of the double spending.

When a double spending is performed in the Bitcoin network, the two transactions end up in the mempool of the miners. If the miner that mine the block is honest will insert only one of the two transactions in the block, while the second one will be considered invalid.

But there is the possibility that two honest miners mine simultaneously two blocks, validating the two transactions and creating a fork in the blockchain. In this case, the network will consider only one of the two chains, the longest. For this reason is recommended to wait 6 confirmations, or 1 hour, before to accept the transaction. After 6 blocks, we are mathematically sure that the transaction is stored in the main chain.

What happens if the attacker is a miner? He can spends it's bitcoins in the main chain while he keeps secret a fork of the blockchain, without its transactions. As soon as its chain is longer than the main chain, he broadcasts it to the network. The network, according the longest chain rule, will accept the new chain and the malicious miner will have again its bitcoins. But, for create a chain longer than the main chain, the attacker must have 51% of the hashing power of the network, to add blocks to his chain faster than the other miners. In fact, this is called "51% hashing power attack" and even if the attack succeeds, as soon as the network realises that the attack has been performed, all the nodes migrate to another blockchain without the attack. The old chain is abandoned and value of the token drop down.

This attack is still possible in the Bitcoin's network but, compared to others blockchains were the cost to perform this attack is not too much, attack the network of Bitcoin require a huge amount of moneys and doesn't produce any profit.

Recently Verge's blockchain suffered this type of attack. According this news, it took only 47$ per hour to perform the 51% attack, while this attack in Bitcoin requires 716.000$ per hour.

2.2 Avoid cheating in the Lightning Network

The Lightning Network has been designed to encourage to publish on chain the most updated balances of the two parties. For do that, the protocol provide some mechanisms that, in case of one part try to cheat, the other part can punish him/her taking all the amount registered in the channel. The mechanisms used are two: *Time Lock* and *Hash Preimages*.

2.2.1 Time Lock

The time lock is used to lock the transaction output for a certain time and make it spendable in the future.

There are two types of time lock:

- **CheckLockTimeVerify (CLTV)**: The output can be spent from a specific time onwards, like for example after block X or a specific date/time.
- **CheckSequenceVerify (CSV)**: This use a relative time, like for example the output can be spent after x blocks from now or after x time from now.

An example is reported in Figure 1

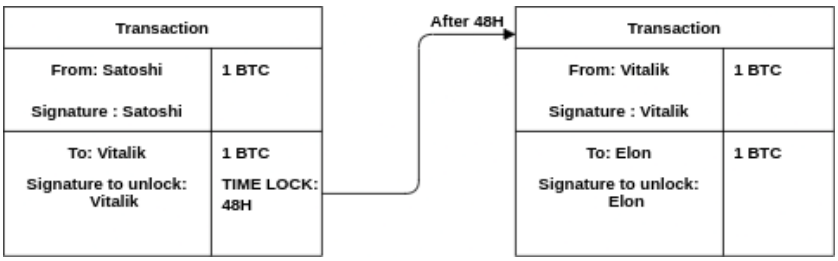

Figure 1: Transaction with time lock CSV.

2.2.2 Hash Preimage

A preimage, called also secret, is a randomly generated string, impossible to guess. It is hashed using an hash function, so anyone that know the secret can easily generate the hash. In an hashlocked transaction, in addition to the amount to transfer, the sender and receiver, it contains also the script with hash of the preimage. So the output is locked and for spend it you need provide the preimage to the script, it hash the preimage and if the output correspond to the hash stored, the amount is unlocked.

2.2.3 Example

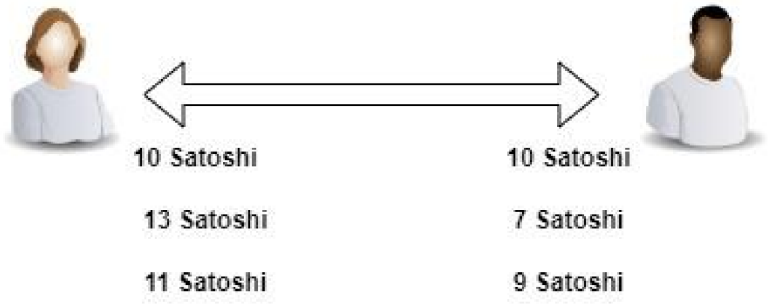

Figure 2: Transactions in a lightning network channel.

When Alice and Bob decide to open a channel, they perform a transaction from the on-chain to the off-chain account. This represent the start balance of the channel and the initial transaction is called opening transaction. They can send also a different amount of tokens to the channel, is not required that both the parties deposits the same amount. After the channel is opened, they perform two symmetric commitment transactions, one on each side, and they differ only in the output locking scripts. For each new payment, Alice and Bob generate a new preimage, send the hash of it to the other side and execute a new commitment transaction. The output of the transaction contain time and hash lock. We take into account the second transaction in Figure 2. In this case, the commitment transaction generated and signed from Bob to Alice contains:

- **From: Alice, Bob** with a balance of **20 Satoshi** and signed by **Bob**
- **To: Bob** with a balance of **7 Satoshi** and it is required the signature of **Bob** to unlock them
- **To: Alice, Bob** with a balance of **13 Satoshi** with a **hash lock for Bob** (he needs the Alice's preimage to unlock them) or a **time lock for Alice**

The commitment transaction generated and signed from Alice to Bob is the opposite:

- **From: Alice, Bob** with a balance of **20 Satoshi** and signed by **Alice**
- **To: Alice** with a balance of **13 Satoshi** and it is required the signature of **Alice** to unlock them
- **To: Alice, Bob** with a balance of **7 Satoshi** with a **hash lock for Alice** (she needs the Bob's preimage to unlock it) or a **time lock for Bob**.

Only one of the two commitment transactions can be confirmed on chain, to avoid the double spending. As seen, the two commitment transactions includes the anti-cheating mechanisms.

Now suppose that the third commitments transactions are executed by Alice and Bob. But, before to do that, the parties reveal each other the preimage used in the previous commitment transaction.

As soon as Bob notices that Alice tries to cheat him, he signs his second commitment transaction using his second preimage and taking his Satoshi, while Alice is locked by the time lock. But Bob knows Alice's second preimage, so he use it to unlock the hash script and he takes also her 13 Satoshi. Thank to that, Bob punish Alice redeeming the whole amount deposited in the channel.

2.2.4 Watchtower

Bob must periodically check the blockchain to monitor possible cheating behaviour by Alice, but he can delegates this task to a watchtower. A watchtower is a service that periodically monitors the blockchain, checking if were executed behaviours to cheat its client. If it is so, then the watchtower broadcasts the punishment transaction, moving all funds in the channel to the on-chain wallet of its client.