Background Assignment

A. Conceptual Knowledge

> 1. What is a smart contract? How are they deployed? You should be able to describe how a smart contract is deployed and the necessary steps.

A smart contract is self-executing, distributed chunk of code that describes and enforces terms or an agreement. A smart contract is compiled into bytecode and an ABI. The byte code is published to a blockchain (ethereum for example). The ABI is used by an application to interact with the smart contract on the block chain. Smart contracts are written usually in Solidity, personally I've been using remix for the live compile and testing.
The contract is compiled using some solidity compiler. To actually deploy the compiled smart contract, I've used truffle (with their own configs), or ethers.js and infura. You **should** also publish your source code to whatever explorer is most appropriate, but that isn't strictly speaking necessary for deployment of a smart contract.

> 2. What is gas? Why is gas optimization such a big focus when building smart contracts?

Gas is the fee/cost to execute the smart contract that is payed out to the person or system that is mining or confirming a transaction. Gas optimization is all about reducing the cost of the execution of your smart contract. The more logic / memory / processing it takes to execute a smart contract the more it will cost. Especially when building something like a dApp reducing the cost is critical to keeping your users around. If you are costing them unnecessary money, they won't want to use your app.

> 3. What is a hash? Why do people use hashing to hide information?

A hash is the output of a "hashing" algorithm, that takes some chunk of data in, usually encrypts it some way, for example SHA256, and the produces some sort of encoded character based output.

Hashing can add some important properties to data.

- Confidentiality - or encrypting data. This is part of the hashing function that

makes it so it is extremely difficult, or ideally all but impossible, to get the original data back out, unless you have the key to decrypt the data.

- Authenticity - you can be sure of who sent you this data, with the public key, because if it was someone else the keys would have produced a different hash.
- Integrity - you can also use a hash to check that data hasn't been altered because that would produce a different hash.

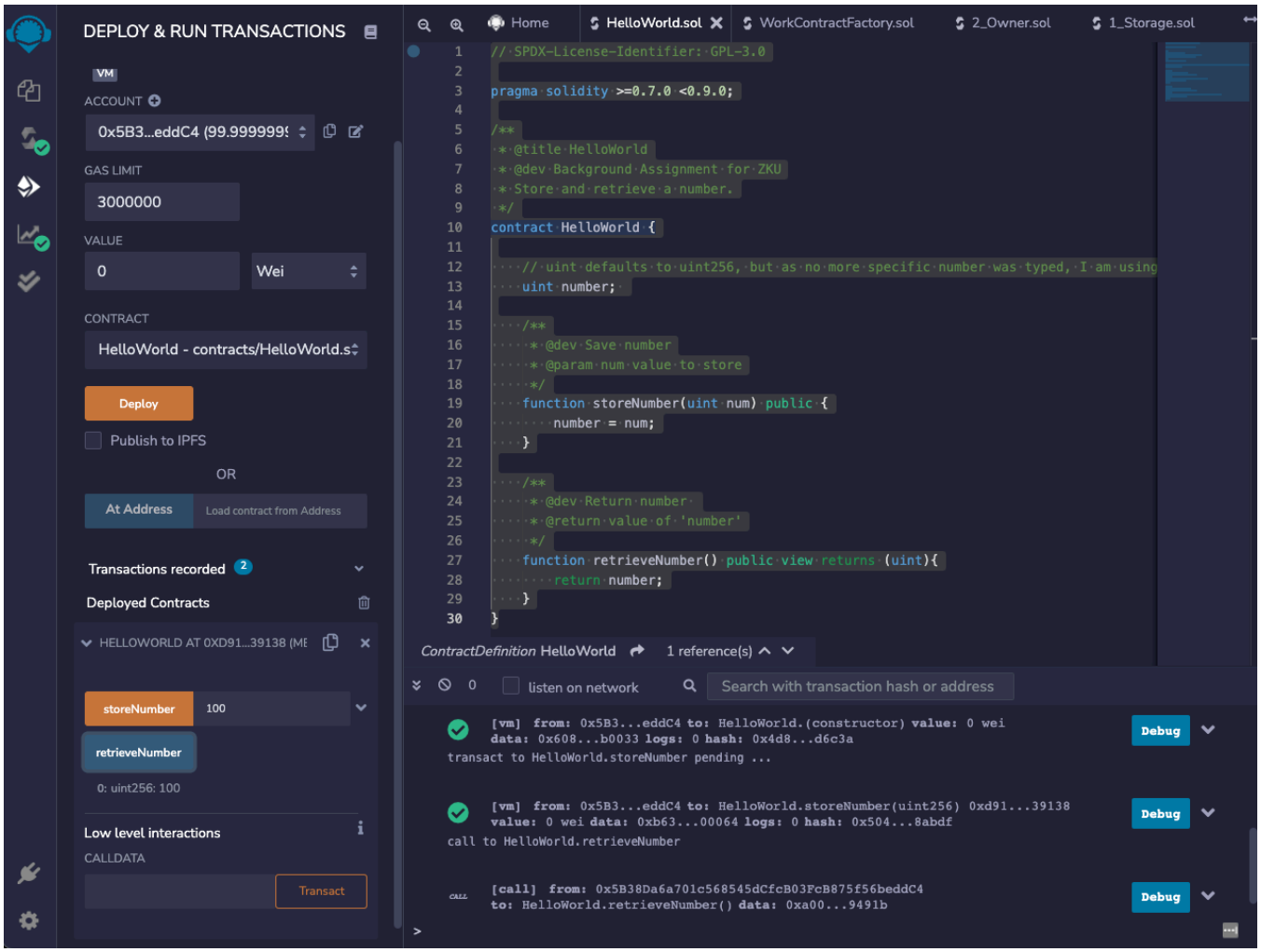
> 4. How would you prove to a colorblind person that two different colored objects are actually of different colors? You could check out Avi Wigderson talk about a similar problem here.

I would have the colorblind person make two different places / labels that I don't have access two, like a sheet of paper. Take two different colored objects, that are otherwise identical. Make sure they they know which is "object 1" and which is "object 2", for example by placing the first object I hand them on one sheet of paper (in a different room) labelled "Object one", and then give them the second object to place on the sheet of paper labeled "Object 2". I will be in a separate room so I don't have any reference to the labeling. The color blind person will randomly select one of the objects and bring it out to me. I (as the prover) will be able o tell them, this is the first (or second) object that I handed you. I could give them the information that it is "blue" or any other color, but I don't need to. Either way they would be able to verify that I can in fact tell the difference between these two objects based only on color.

B. You sure you're solid with Solidity?

> 1. Program a super simple "Hello World" smart contract

HelloWorld.sol



2. On the documentation page, the “Ballot” contract demonstrates a lot of features on Solidity. Read through the script and try to understand what each line of code is doing.

_Ballot.sol demonstrats the use of:

1. Structs
   programmer defined data type (structure).
2. Mappings
   an implementation of a hash table, a key -> value relationship.
3. a constructor
   The method called only at initialization (deployment) of the smart contract.
4. the `memory` keyword
   Temporary storage, only available for the length of the execution of the smart contract.

5. `msg.sender` refers to the address that initiated the transaction or function call on the smart contract.
6. Iterating with a `for` loop.
7. The use of a array / array accessors.
8. `require`
   check if some condition is true, throw an error if it is now, with optional user defined error messages.
9. if / else statements

> 3. Suppose we want to limit the voting period of each Ballot contract to 5 minutes. To do so, implement the following: Add a state variable startTime to record the voting start time. Create a modifier voteEnded that will check if the voting period is over. Use that modifier in the vote function to forbid voting and revert the transaction after the deadline.

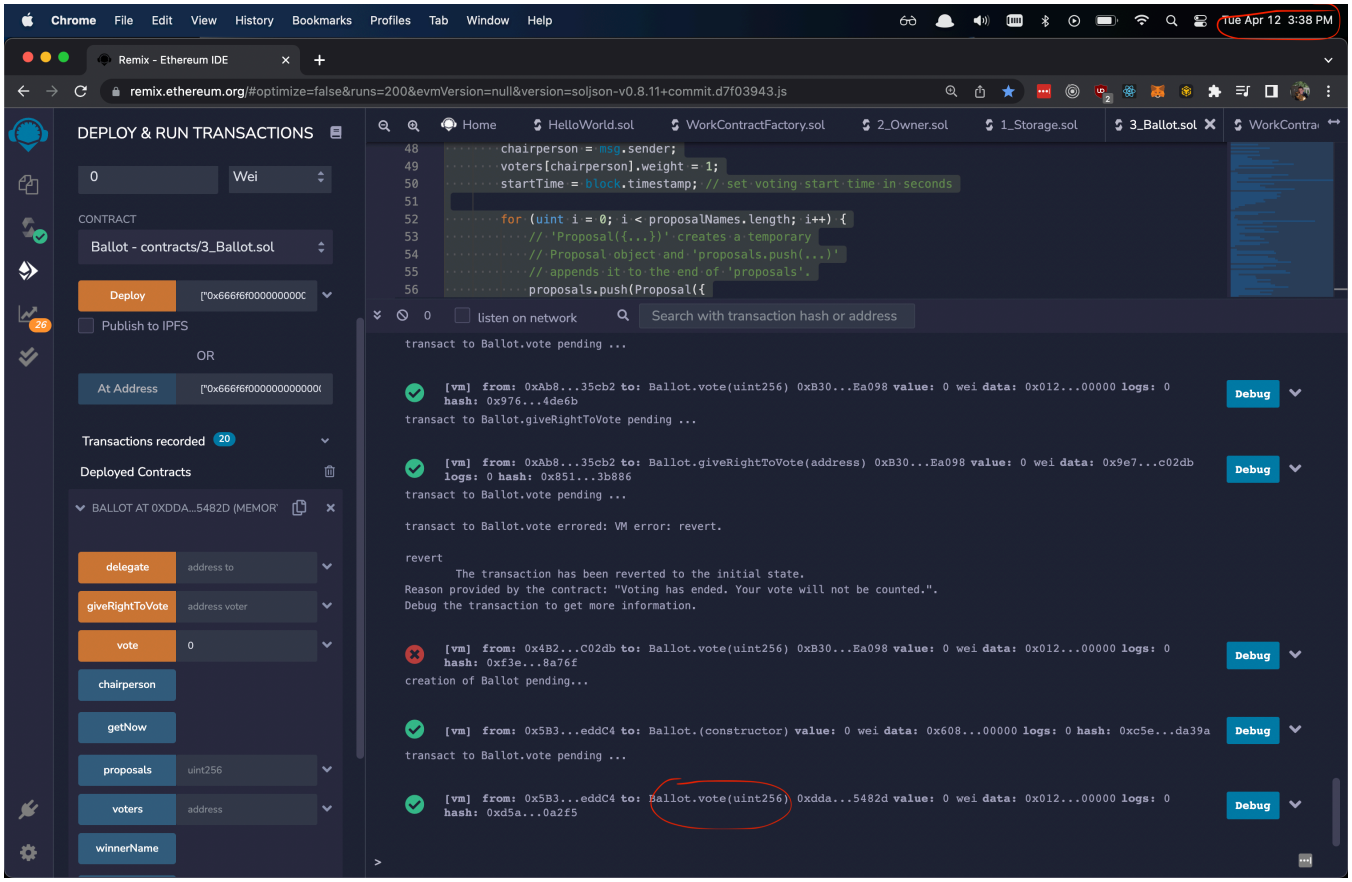
Relevant snippets of code:

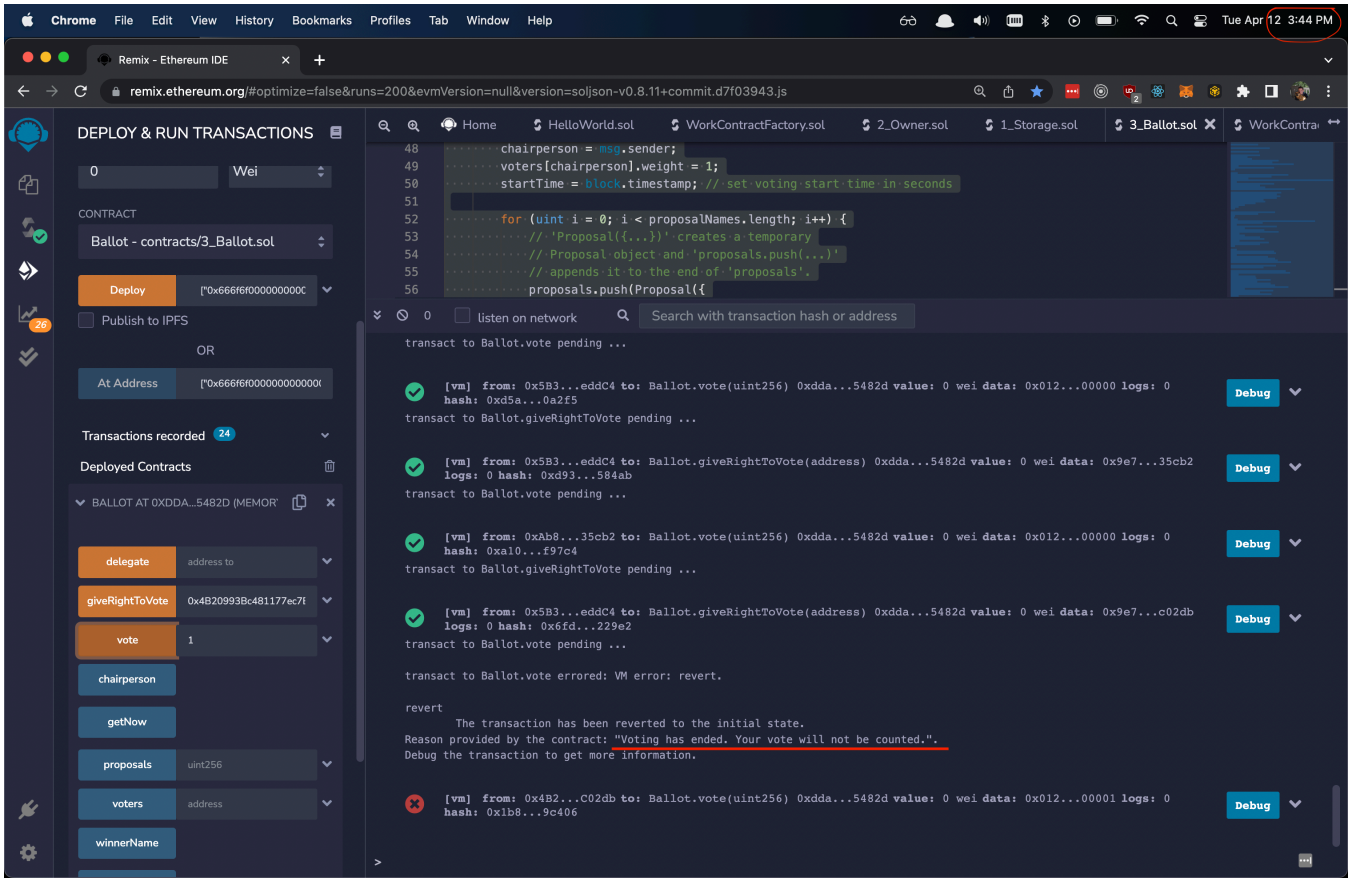
```
...
// Allow voting for 5 minutes
uint startTime;
modifier voteEnded {
        // 300 = 5 (minutes) * 60 (seconds) conversion
        if(block.timestamp > startTime + 300) {
                revert("Voting has ended. Your vote will not be counted.");
        }
        _; // Call the function the modifer wraps
}
...
constructor(bytes32[] memory proposalNames) {
        chairperson = msg.sender;
        voters[chairperson].weight = 1;
        startTime = block.timestamp; // set voting start time in seconds
...
function vote(uint proposal) public voteEnded {
...
```

Full Code

> 4. Deploy your amended script and test the newly implemented functionality in part 3. Submit (1) your amended version of the contract on Github or Gist and (2) screenshots showing the time of contract deployment as well as the

transaction being reverted once past the voting period.




Chrome File Edit View History Bookmarks Profiles Tab Window Help
Tue Apr 12 3:44 PM
Remix - Ethereum IDE
remix.ethereum.org/#optimize=false&runs=200&evmVersion=null&version=soljson-v0.8.11+commit.d7f03943.js
DEPLOY & RUN TRANSACTIONS
Home
HelloWorld.sol
WorkContractFactory.sol
2_Owner.sol
1_Storage.sol
3_Ballot.sol
WorkContra
chairperson = msg.sender;
voters[chairperson].weight = 1;
startTime = block.timestamp; // set voting start time in seconds
for (uint i = 0; i < proposalNames.length; i++) {
// 'Proposal({...})' creates a temporary
// Proposal object and 'proposals.push(...)'
// appends it to the end of 'proposals'.
proposals.push(Proposal({
0
Wei
CONTRACT
Ballot - contracts/3_Ballot.sol
Deploy
Publish to IPFS
OR
At Address
Transactions recorded 24
Deployed Contracts
BALLOT AT 0XDDA...5482D (MEMOR
delegate
address to
giveRightToVote
vote
1
chairperson
getNow
proposals
uint256
voters
address
winnerName
listen on network
Search with transaction hash or address
transact to Ballot.vote pending ...
[vm] from: 0x5B3...eddC4 to: Ballot.vote(uint256) 0xdda...5482d value: 0 wei data: 0x012...00000 logs: 0 hash: 0xd5a...0a2f5
Debug
transact to Ballot.giveRightToVote pending ...
[vm] from: 0x5B3...eddC4 to: Ballot.giveRightToVote(address) 0xdda...5482d value: 0 wei data: 0x9e7...35cb2 logs: 0 hash: 0xd93...584ab
transact to Ballot.vote pending ...
[vm] from: 0xAb8...35cb2 to: Ballot.vote(uint256) 0xdda...5482d value: 0 wei data: 0x012...00000 logs: 0 hash: 0xa10...f97c4
transact to Ballot.giveRightToVote pending ...
[vm] from: 0x5B3...eddC4 to: Ballot.giveRightToVote(address) 0xdda...5482d value: 0 wei data: 0x9e7...c02db logs: 0 hash: 0x6fd...229e2
transact to Ballot.vote pending ...
transact to Ballot.vote errored: VM error: revert.
revert
The transaction has been reverted to the initial state.
Reason provided by the contract: "Voting has ended. Your vote will not be counted.".
Debug the transaction to get more information.
[vm] from: 0x4B2...C02db to: Ballot.vote(uint256) 0xdda...5482d value: 0 wei data: 0x012...00001 logs: 0 hash: 0x1b8...9c406