

## Albert Zhang ZKU Background Assignment

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 a computer program stored at an address on a blockchain that automatically executes when certain conditions are met. For the Ethereum blockchain, smart contracts are usually written in Solidity, a high-level programming language. A very simple example of a smart contract would be a smart contract that accepts a number as input and stores that number, as well as returning that number when asked for it. If deployed on a blockchain, anyone with access to the address of this smart contract and enough assets to pay for gas fees could use this smart contract to store a number there and retrieve it.

On Ethereum, you deploy a smart contract by sending an Ethereum transaction containing the smart contract's compiled code without specifying a recipient. Usually, a convenient way to do this is by using tools such as Hardhat or Truffle which will help you deploy your smart contract. There are similar methods for deploying smart contracts on other blockchains, such as Harmony.

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

Gas is the unit of computational work or "energy" needed to run certain operations on a blockchain, such as the Ethereum or Harmony networks, so you can think of gas like a fuel that powers operations on a blockchain. In order to run smart contracts, gas must be paid as the fuel used to perform the computations involved in those smart contracts. The more operation and computations involved in the smart contracts, the more gas is necessary. This is analogous to how in the real world, more gas/fuel is needed to drive a truck than a car. Thus, gas optimization is a big focus when building smart contracts because we want to reduce the cost of running and operating those smart contracts. Having lower gas fees for a smart contract makes using it more accessible to people and more likely to be used often, both of which are important.

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

A hash is an output of a hash function, which is a function that maps data of arbitrarily large size to fixed-size values while attempting to relatively evenly and non-transparently map the data such that it is hard to invert the process and obtain the original data from the outputted hash. Thus, a hash can hide information because it's hard to reverse the hashing process: it is easy to hash a password but extremely difficult to take that resulting hash and convert it back into the original password. So, we can use hashes to hide private information. People use hashing to hide information because this can protect information that they want to keep private and secure, such as passwords. For instance, if storing passwords in a database, a developer should always store hashes of the passwords rather than plaintext/original passwords in a database because this way, if the database is hacked, the hacker only has access to the hashes and not the original passwords and thus the users are protected.

How would you prove to a colorblind person that two different colored objects are actually of different colors?

I can prove to a colorblind person (let's call her Alice) that two different colored objects are actually of different colors using zero-knowledge proofs. Without loss of generality, assume that the two different

colored objects are a small black ball and a small white ball, identical in every way except for color. Then, I tell Alice which ball is black and which is white. I then turn around, facing away, and tell Alice to hold one ball in each hand without telling me which hand has which, and letting her decide which hand to hold which (so then she can shuffle them around while tracking which is which). I claim to identify which is black and white still because of their color difference, so I turn around and correctly identify them (which Alice can verify since she knows which is in which hand). If I guessed, I would only have a  $\frac{1}{2}$  chance of getting this right. I then repeat this 20 times, each time correctly, then the odds I would do this randomly are  $1/2^{20} = 1/1048576$ , so this should be sufficient to convince Alice that I actually can distinguish the balls using color (since too unlikely that I just guessed 20 times correctly in a row). Note that I haven't fully proven with certainty that the balls are two different colors, but I've shown that it's extremely, extremely unlikely that they are the same color, which is good enough for practical purposes.

B. You sure you're solid with Solidity?

**Program a super simple "Hello World" smart contract:**

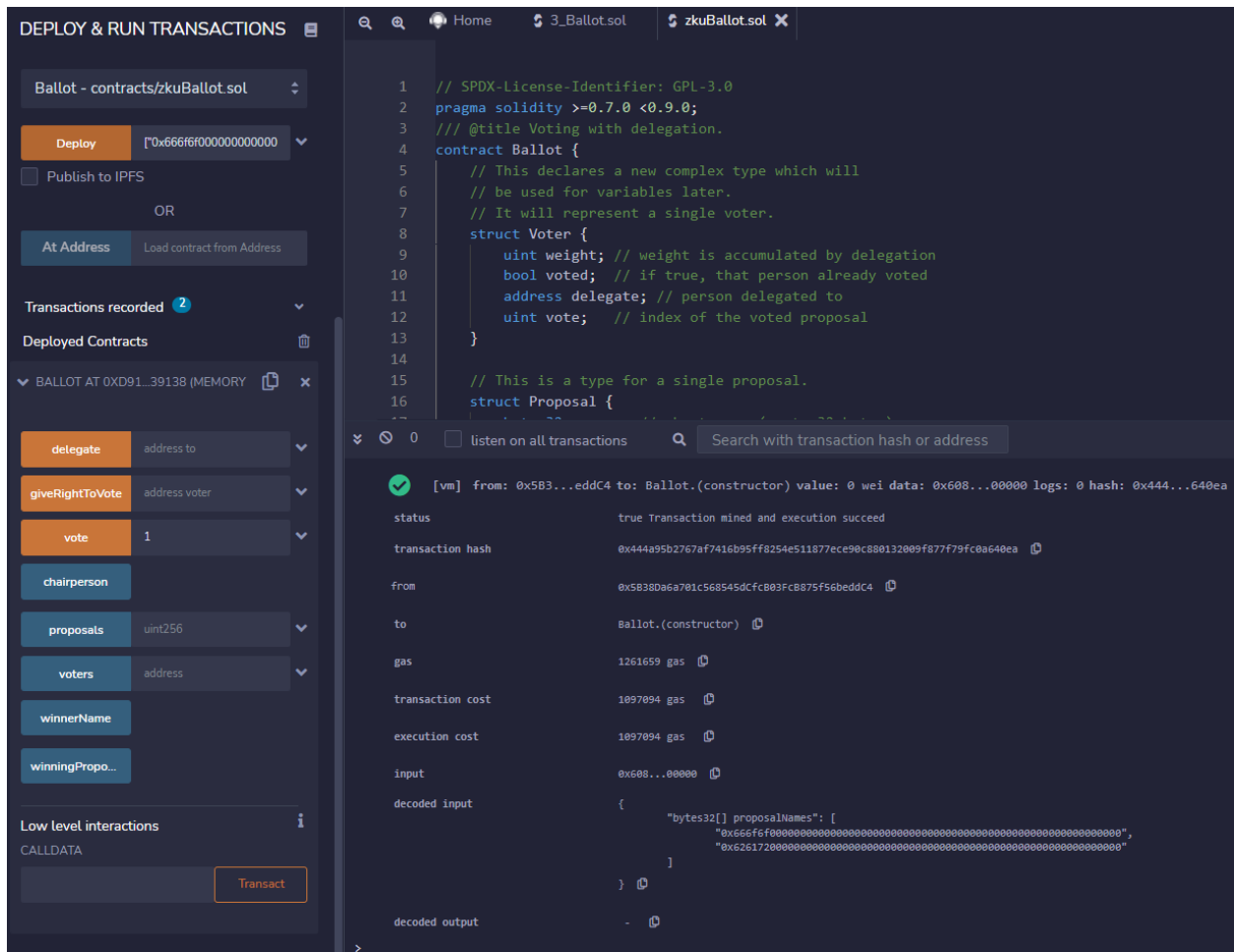
The screenshot shows a web interface for interacting with a smart contract. On the left, there's a sidebar with deployment options: ACCOUNT (0x583...eddC4), GAS LIMIT (3000000), VALUE (0 Wei), and CONTRACT (HelloWorld - contracts/helloWorld.sol). Below this are buttons for 'Deploy', 'Publish to IPFS', and 'At Address'. A section titled 'Transactions recorded' shows 'Deployed Contracts' with 'HELLOWORLD AT 0XD91...39138'. Below that, there's a 'storeNumber' button with a value of 15, and a 'retrieveNumber' button. The 'Low level interactions' section shows 'CALLDATA' and a 'Transact' button. On the right, the Solidity code for 'helloWorld.sol' is displayed, showing a contract with a 'storeNumber' function and a 'retrieveNumber' function. The bottom of the interface shows a 'ContractDefinition HelloWorld' with 1 reference(s) and a search bar for transaction hash or address. Below the search bar, there's a call to 'HelloWorld.retrieveNumber' and a transaction log entry: 'CALL [call] from: 0x58380a6a701c568545dcfc803fc8875f56beddC4 to: HelloWorld.retrieveNumber() data: 0xa00...9491b'.

Code at

<https://github.com/alzh9000/ZKU/blob/1d4e7d378724ea03d1c80566baf430c5e1346f62/helloWorld.sol>

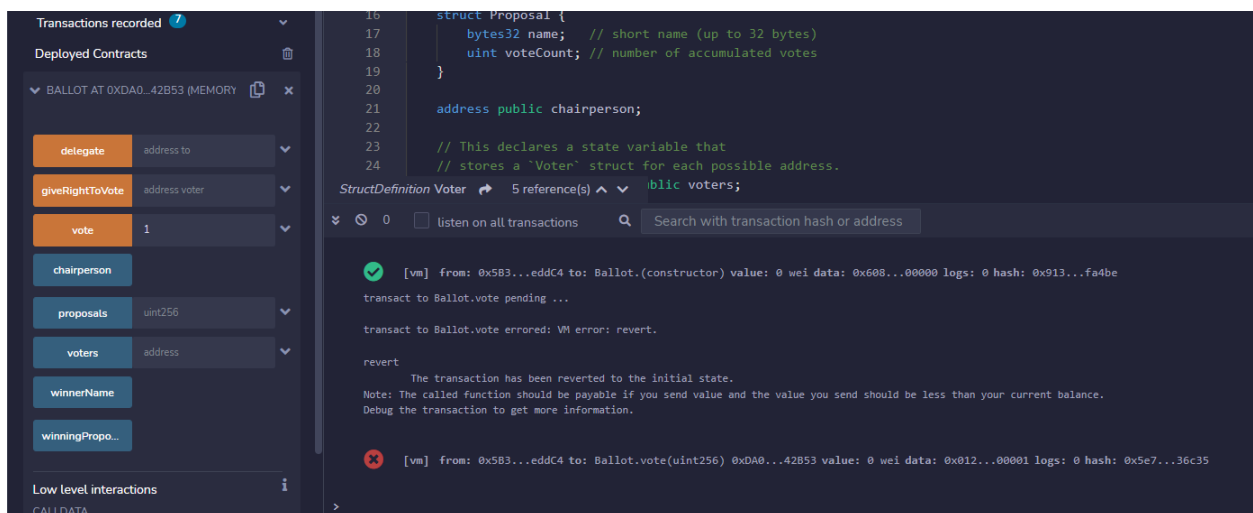
Suppose we want to limit the voting period of each Ballot contract to 5 minutes.

Screenshots showing the time of contract deployment



The screenshot shows the Remix IDE interface. On the left, the 'DEPLOY & RUN TRANSACTIONS' panel is open, displaying the 'Ballot - contracts/zkuBallot.sol' file. The 'Deploy' button is highlighted. Below it, there are options to 'Publish to IPFS' or 'At Address'. The 'Transactions recorded' panel shows a list of transactions, including the deployment of the Ballot contract. The main editor shows the Solidity code for the Ballot contract, which includes a 'Voter' struct and a 'Proposal' struct. The right sidebar shows the 'Transactions recorded' panel with a list of transactions, including the deployment of the Ballot contract.

as well as the transaction being reverted once past the voting period.



The screenshot shows the Remix IDE interface. On the left, the 'DEPLOY & RUN TRANSACTIONS' panel is open, displaying the 'Ballot - contracts/zkuBallot.sol' file. The 'vote' button is highlighted. The main editor shows the Solidity code for the Ballot contract, which includes a 'Voter' struct and a 'Proposal' struct. The right sidebar shows the 'Transactions recorded' panel with a list of transactions, including the deployment of the Ballot contract and a transaction that was reverted. The transaction details show a 'VM error: revert'.

More than 5 minutes after deploying the contract, the `vote` function reverts the transaction because the voting period is over, as desired.

Code at

<https://github.com/alzh9000/ZKU/blob/110d16ba0e039eee46bab2ef340540cf69bbab8c/zkuBallot.sol>