Zero-Knowledge University (ZKU):

Solidity Programming

Tal Derei

01 March 2022

Course Registration Email: Tad222@lehigh.edu

Discord Username: Margulus#4273

Github (Personal Repistory): `https://github.com/TalDerei`

Gitub (ZK Research): `https://github.com/TalDerei/Zero-Knowledge`

> The "Zero-Knowledge" github repository above contains personal presentations on emerging zero knowledge research as an incoming masters student studying CS and blockchain. Member of the Scalable Systems and Software Research Group (SSS) at Lehigh University: `https://sss.cse.lehigh.edu/people/`. My research currently focuses on zero knowledge scaling solutions like zkEVM (general purpose EVM computations on L2). Additionlly, researching ZK acceleration using ASICs/GPUs to improve proof generation and verification times.

Codebase (Git Commits)

`https://github.com/TalDerei/ZKU/commit/f454e194da175b628712dba7f7d265a56f1389ac`

https://github.com/TalDerei/ZKU/commit/ee6787d44acd32812ac1d4519c604ea584998f60

Hello World Program: Implementing the program involved declaring a state variable stored in contract storage, and defining a setter function for storing unsigned integer.

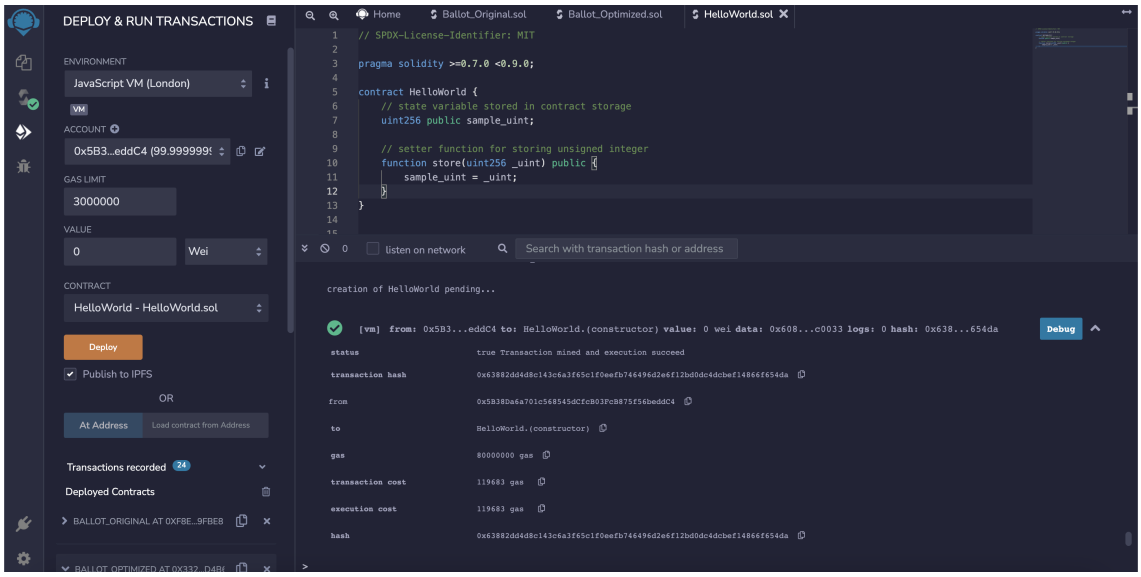

Figure 1: *starter "hello world" solidity program*

Ballot Program: The **current** implementation involves mapping 1 transaction to 1 right to vote. The **desired** implementation implements bulk assign voting rights in one transaction. So the goal is to bulk this operation in a single transaction to amortize the gas cost amongst all the voters in the transaction, similar L2 ZK-Rollups that move computation and state storage off-chain, and post the zk proof of the transactions back on chain for a fraction of the cost.

I thought of two ways of approaching the problem. The first method (and the one implemented in my solution) is setting an array of addresses as a parameter, and looping through the addresses in the function. The second method is more interesting, involv-

ing a clever compression techniques. The way it works is: [1] the "address" type stores a 160-bit Ethereum address, but we don't need that much space, [2] so instead, represent each bit in this 160-bit field slot as separate address, [3] this grants many addresses the right to vote with one SSTORE operation (i.e. function call). You're "packing" many addresses (who's actual size is much small than 160-bites of course) into different sections of one "address" types 160-bit fields. So it's an smart encoding solution. The idea is a variation of the "ERC-721A" standard that introduces "balance packing": a clever encoding reduces minting gas cost by as much as 50%. Typically with any ERC-721 transaction, there are two storage operations (updating the balance and address), and ERC-721A cleverly packs multiples of these operations into one call.

Analyzing the gas costs yielded expected results: the amortized gas cost was 50% for the optimized ballot contract!

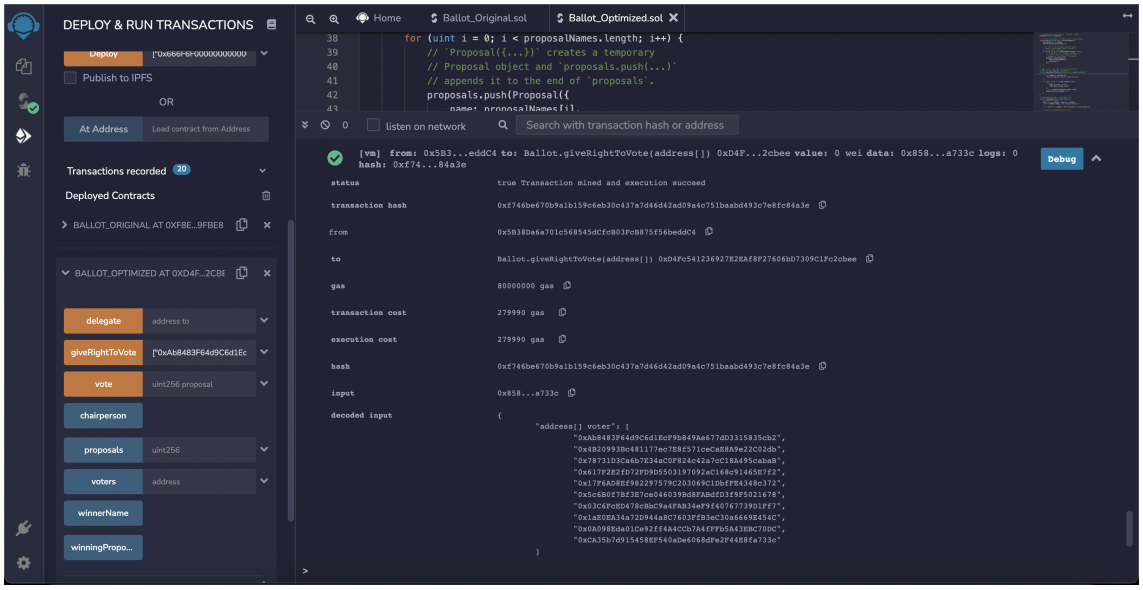

Figure 2: *optimized "ballot" contract*

The gas cost for 10 addresses 280,000 gas, which comes out to 28,000 gas/address when amortized to account for the batch operation. Comparing it to a single transaction with

a single address yields a gas cost of approximately 50,000.

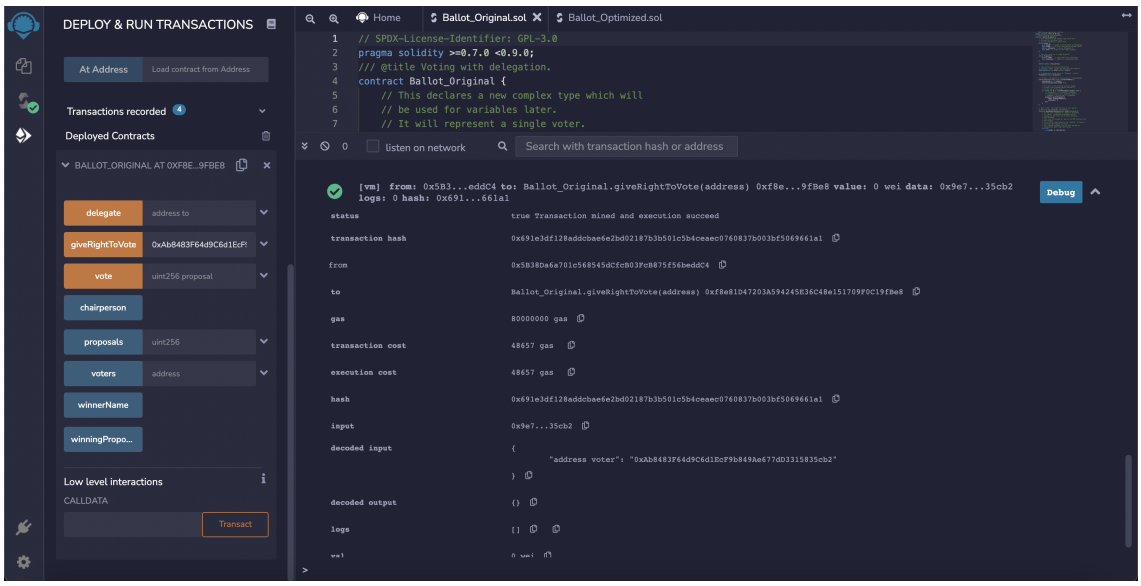

Figure 3: *original "ballot" contract*

If we were to call the "giveRightToVote" separately for each address in the original contract, that would cost 10 x 50,0000 = 500,000 gas, or about 2x more expensive.

We can further optimize this and reduce gas costs by implementing some clever encoding scheme, similiar to ERC-721A described above or by using merkle trees (and storing their hash). Another technique is further storing on decentralized storage like IPFS to reduce costs.