Zero-Knowledge University (ZKU): Assignment 4

Tal Derei

28 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`

Q0. Stream:

Stream B: Application Track

**** I'll be attempting some of the infrastructure coding exercises for practice****

Q1. Scaling the Future:

1. Scalability Solutions:

There exists both layer-1 and layer-2 solutions to achieve the scalability trilemma: scalability, decentralization, and security. Sharding represents Ethereum's L1 approach, while rollups (and the rollup-centric future of Ethereum) represent the L2 approach.

[1] **State Channels** treat the blockchain as the settlement layer, locking funds in a smart contract and transacting with an external user using signed messages. Once the other user decides to cash out, the Ethereum smart contract verifies the signatures of the signed messages, pays the user funds from the contract, and closes the channel. In this model, rather than initiating a transaction every time a payment needs to be made, the channel processes a single, final transaction and makes a lumpsum payment that's settled on-chain. Benefits: transactions are happening off-chain with low gas fees, and incur low network fees since you only transact on-chain when you open and close a channel. Drawbacks: state-channels limit communication to users *inside* the state-channel, so transactions with external users requires opening an additional state-channel. More complex applications are also prohibitive (i.e. like Uniswap), since they feature complex intermediary steps to execute a swap that don't have a user's authorization / permission. There's also capital efficiency efficiency concerns of locking up liquidity. And similar to Optimistic Rollups (that require an external party to submit a fraud proof in the case of malicious behavior), these state-channels require an external party to monitor the network periodically. State channels are ultimately useful for two-party communication (e.g. customer and merchant) in a fast, cheap, and reliable way, but become a non-starter scaling solutions when considering more complex interaction models and lack of capital efficiency.

[2] **Side Chains** are independent blockchain pegged to the main blockchain, and have their own security properties and consensus mechanisms (than the main chain) to process transactions. Polygon's PoS (Ethereum) and Liquid (Bitcoin) are two well-known examples of side-chains. Benefits: locking up funds on main chain and minting equivalent tokens on side-chain network allows for cheaper transactions off-chain. Drawbacks:

security guarantees are weaker since sidechains don't inherent the security properties of the main chain. These sidechains usually also have a smaller validator (full node) set, which further contributes to the security risks.

[3] **Plasma** is a series a smart contracts (”plasma chains”) outside the main chain. Operators of these plasma chains submit merkle roots (representative of transfers on the plasma chain) to the root chain (i.e. Ethereum). Benefits: [1] in terms of scalability, plasma allows you to send transactions to anyone, unlike state channels which require parties to be apart of the same channel. [2] in terms of security, the plasma chains are secured by the same security guarantees as the underlying Ethereum blockchain, while side-chains have their own consensus and security properties. Drawbacks: [1] Similar to an optimistic rollup construction, fraud proofs are used the check the validity of state in plasma chains. This involves a challenges period (typically   1 week) allowing any participant to issue a fraud proof if they suspect malicious transactions. [2] Plasma uses operators to post merkle root commitments on-chain, and these third-party operators can perform data-availability attacks where they withold some transaction data from the mainchain.

[4] **Rollups** are a layer-2 scaling solutions that maintain a smart contract on the main-chain, and move execution off-chain. But unlike plasma chains which only post the merkle root on-chain, rollups post the state merkle root along with the transaction data of each batch on-chain (as calldata) so anyone can reconstruct the state. Posting the transaction data on-chain mitigates the data availability attacks previously mentioned. Two flavors of rollups include zk-rollups (validity proofs) and optimistic rollups (fraud proofs).

2. ZK-Rollups:

ZK-Rollups are layer-2 scaling solutions that maintain a rollup contract on-chain, and move computation and state storage off-chain. Execution is performed off-chain by sequencers who generate a batch of transactions, execute the batch, and then post the generated proof of validity back on the main chain where a collection of smart contract pre-compiles perform proof verification. As seen in the diagram below, the main features include [1] moving execution off-chain through the rollup contract, [2] sequencers order transactions (and can potentially extract MEV in a decentralized model), [3] sequencers execute the transactions and aggregate them in a batch, [4] proof of validity is generated on the batch of transactions, and multiple proofs can be recursively aggregated into a single succinct proof, [5] relayer posts the proof back on L1 and pays gas fee, [6] smart contract verifies the proof. Vitalik, along with other researchers and developers in the blockchain space, have long argued for a "Rollup-Centric Ecosystem" where Ethereum remains the settlement layer, and zk-rollups are the primary execution chains to achieve scalability.

<u>Drawbacks</u>: [1] A short coming is the cost of posting the transaction data of each batch on-chain. Even though this is posted as call-data, the amortized is still not low enough to make rollups the cheapest option on the market nowadays. EIP-4488 aims to combat this by reducing call-data costs 5x, effectively making rollups 5x cheaper. [2] Ethereum Virtual Machine inefficiencies: Ethereum was not designed to validate zero knowledge cryptographic proofs, instead simple ERC-20 token transfers. The virtual machine is inefficient at performing the mathematical calculations, since you can only process so many transactions / sec, leading to high gas fees. The amount of computation an Ethereum node is performing to validate ZK-proofs is roughly 10x more than what's required because of inefficiencies of the smart contract platform and virtual machine. [3] Ethereum ONLY implements one elliptic curve BN-254. The elliptic curve are used in the verification smart contracts used on Ethereum main chain. Lack of curve options constrains

what aggregation schemes and polynomial commitments these zk-rollups can use.

Solutions: some potential solutions to the above problems is increased adoption and optimization of precompiles of smart contracts for expensive cryptographic operations, and improve / upgrading the EVM for increased efficiency.

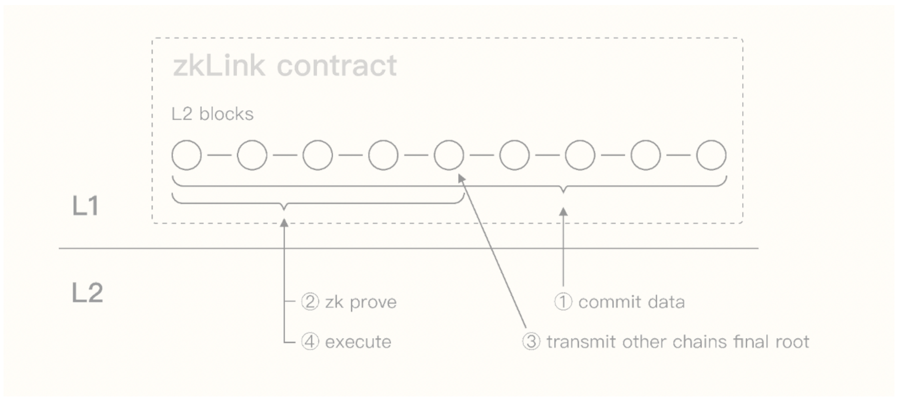

Figure 1: *ZK-Rollup Diagram*

3. Stateless Clients:

Ethereum's unbounded growth will eventually make it infeasible for a majority of miners to run a full node. As the state storage increases due to more accounts and deployed smart contracts, the world state becomes larger and full nodes have increased hardware (storage and memory) requirements. **Stateless clients** delegate storing the world state to another type of participant in the network known as "state providers". Stateless clients can therefore validate blocks without storing the Ethereum state.

Mathematically speaking, Ethereum's State Transition Function (STF) takes in some block B and state S, and the EVM outputs new state S': STF(B, S) = S'. Stateless Clients transform the STF by taking in the block B, 32-byte state root S (rather than the entire merklized patricia tree), and witness W (set of merkle branches), and the EVM outputs new state S': STF(B, S, W) = S'. Therefore rather than computing a complete state for each block, we're only computing the *state changes* for each block, and verifying the consistency of those changes against a previous block. If we consider where zero-knowledge fits in, recall that the witness W is a set of merkle branches. These merkle branches prove the values of all the data that the execution of block B accesses. An associated proof is required to verify that the data is not tampered with, so witness W = (data, proof). The bottleneck with such an approach is still the size of the witness. Some solutions to reduce the size of the witness include verkle tree and polynomial commitments. Overall, stateless clients will be used in the context of block verification and cross sharding communication in ETH 2.0 with the introduction of shard chains.

Q2. Roll the TX Up:

1. RollupNC Source Code:

[1] *UpdateState (Contract):*

- updateState() takes in as input parameters **a,b,c, input**
- a,b,c correspond to input parameters from $'1\_end\_to\_end.js'$, which correspond to proof elements stored in $'test\_1\_update\_proof.json'$ (which contains the actual proof). It then checks whether 'input' corresponds to the curren root node
- calls $update\_verifyProof()$ contained in 'Update_verifier.sol', and then uses pairing-based cryptography (Groth16) to check the validity of the proof
- finally update the merkle root and emit event containing *newRoot, txRoot, oldRoot*

[2] *Deposit (Contract):*

- deposit() takes in as input parameters **pubkey, amount, tokenType**
- checks the token type and amount against a registry *'registeredTokens'*

- 'depositArray' (containing [eddsapubkey, amount, nonce = 0, tokenType]) is hashed with *mimcMerkle.hashMiMC*, creating a 'depositHash' representing an account leaf. The depositHash is then pushed to an array of 'pendingDeposits' along with an *RequestDeposit* event emitted
- now that you have an array of deposits, you hash deposit array into on-chain Merkle root called '*deposit_root*'.
- coordinator then inserts d*deposit_root*into *balance tree* at deposit_root.height
- use deposit_proof to update balance root on-chain

[3] *Withdraw (Contract):*

- withdraw() takes a bunch of input parameters, and then checks whether transaction exists in specified transactions root
- 'withdraw_verifyProof' takes in as input parameters: a,b,c (corresponding to withdrawA/withdrawB/withdrawC variables from $'1\_end\_to\_end.js'$). These contain proof elements from $'test\_1\_withdraw\_proof.json'$. It takes another parameter, which is an array containing the pubkeyX, pubkeyY, hash of nonce and recipient address. It then asserts whether the withdraw proof is valid back in the javascript file

[4] *UpdateStateVerifier (Circuit):*

- SNARK circuit takes as input a collection of transactions and outputs a SNARK proof, which is the submitted on-chain to update the Accounts state root
- take in previous root of the *Accounts* merkle tree and a list of transactions, and output an updated state root. Circuit requires running a bunch of verification checks: transaction check, sender check, sender update, receiver check, and receiver update

An **improvement** to the rollup circuit would be publishing transaction data to the main chain as calldata, in addition to the merkle root, in order to provide data-availability guarantees.

3. ZKSync 2.0 and ZKPorter:

zkPorter puts data availability (i.e. transaction data required to reconstruct the state) off-chain rather than on Ethereum. The data availibility is therefore secured using PoS consensus mechanism by zkSync stakers. Transaction costs are fixed at approximately \$0.01 and 20,000+ TPS, with better security guarantees than optimistic rollups. zk-Porter accounts are also interoperable with the zkRollup accounts, and the only difference from the user's perspective is where the data is stored (on-chain vs off-chain). This dramatically reduces the cost of transacting on L2, since a majority of the fees come from relaying the proof back on the main-chain.

In regards to security, zkPorter is PoS-based, secured by a collection of staking nodes called "Gaurdians. Gaurdians secure the chain by signing blocks and providing data availability guarantees. It's important to note that malicious actors CANNOT steal funds, only freeze the state. If a malicious actors controls the sequencer and 2/3 of the total state, they can theoretically signed valid state transitions and withold data. But they would be "freezing" their own funds as well, not to mention the slippage costs in trying to acquire that much stake in the first place. There's no economic incentive, and the staked tokens require to reach 2/3 will high. Downside is that these data-availability layers will ALWAYS have weaker security guarantees than battle-tested L1 blockchains like Ethereum. This opens up the design space for interesting attack vectors surrounding data availability attacks.

Q3. Recursive SNARK's:

1. Recursive SNARK's

Recursion is a technique for making a SNARK which certifies a computation which includes the verification of another SNARK. Recursive SNARKs can be used in the context of:

[1] *Batching* (parallelism for faster computation)

[2] *Compactness* (e.g. MINA)

Recursive SNARKs are mainly used for efficiency. In the context of zksync 2.0 for example, they have a custom Proof System called RedShift based on PLONK. They use recursive SNARKs to aggregate many proofs in a tree-like structure, each representing a smart contract call, and then create a single proofs that's posted back on the mainchain. If we take Polygon Miden as another example, operators package 5,000 transaction in a single block and generate a STARK proof attesting their execution. Rather than simply posting that proof on-chain, and verifying only 5,000 transactions, we can go a step further with recursive SNARKs. In Miden, up to 200 block proofs are recursively aggregated into a single STARK proof and submitted back to the L1 contract. In that regard, we can verify the execution of 1,000,000 transactions, yielding a higher throughput and lower storage costs on-chain as the amortization costs are now 1 / 1,000,000 per tx. In regards to security concerns, there is a potential attack vector as the latency of these proving systems is cranked up higher in order to generate larger proofs packed with more transactions. We're essentially cranking latency to the maximum in order to reduce the gas costs per transaction. There is a "risk window" taken by these rollup providers / validators (that's abstracted away from the user), where the rollup provider accepts the risk of something happening (i.e. some attack vector) as they're generating the proof. Therefore each rollup needs to consider the maximum batch size (and therefore the maximum latency), which informs the amortized gas per transaction on-chain.

2. Kimchi and PLONK

Mina has a zkSNARK Proof System called *Pickles*: Recursive SNARKs based on PLONK without Trusted Setup. *Kimchi* is the main component for generating the recursive proofs (which keeps the chain size fixed at 22kb) used by pickles. Kimchi represents a collection of improvements / optimizations made on top of the PLONK. The most important is eliminating the trusted-setup by using a bulletproof-style polynomial commitment. In terms of circuit improvements, they add 12 of registers, which means they can have gates that take multiple inputs. Kimchi also adds 9 new gates, representing common operations.

Q4. Final Project Ideas:

1. Project Proposals

[1] **NFT Tooling**: airdrops to whitelisted users are NOT privacy preserving (i.e. you need to supply your public key and reveal your information to participate). My idea is an anonymous airdrop system with reputation service (i.e. you need to own a certain NFT to be eligible for the airdrop). In that regard, the service can be used for NFT merchandise drops that reward NFT token holders. The service will initially return an NFT anonymously (using a mixer) that proves you own an NFT without revealing your identify. Claiming the NFT will mix it will all other users entitled to the airdrop, protecting their anonymity. You can redeem the NFT airdrop with a zero-knowledge proof attesting you belong in the merkle tree without revealing which commitment is associated with your public key. This is a priavcy preserving tool to distribute ”rewards” to existing NFT holders without revealing their identities.

[2] **NFT Tooling**: Another idea could be building on top of the Railgun Protocol: “Rail-

gun is a smart contract system that gives zk-snark privacy to any transaction or smart contract interaction on Ethereum." For example, selling/buying (i.e. trading) NFT items between players privately through Railgun. Then create ZKP off-chain proving ownership, which can be used as a credential for accessing some information.

2. Thinking in ZK

The issue with the first idea is that the NFT would be publicly minted on-chain, which means in the worst case, you can assume that every token holder interacting with the protocol and received an anonymous airdrop. Another issue is that mixers deal with fungible tokens, so it's necessary to think about how mixers would work in a nun-fungible setting to distribute these airdrops. In the worst-case, token holders who can anonymously prove they own an NFT using my platform will receive ERC-20 airdrops. The idea would also require a trusted-third party to verify the validity of the data being supplied to the zkdapp. The second idea is simpler, in that the base logic for proving off-chain ownership using zkp for accessing some information is already implementing, and I would need to implement a proof that allows user to trade their NFTs without revealing which NFTs they're transacting with (i.e. you can prove that an exchange of NFTs took place, but you don't know which NFTs were traded). I think both ideas are able to be finished by the deadline (at least a basic implementation), but I'm worried that some of the more complex pieces of these zkdapps are future developments that are longer-term goals.

Q5. Thinking in ZK

Zksync 2.0 has a "*Progressive Decentralization*" mindset in place. This means you start with a single, centralized sequencer/validator, which means they can control the sequencing/ordering of transactions and extract MEV. Zksync plans on decentralizing the sequencers by implementing a consensus mechanism for adding the next block to the chain for censorship resistance. Hermez has a coordination protocol based on Proof-of-Donation where the sequencer is still centralized, but choosing the sequencer is decentralized. You still have a risk of denial of service attack if the sequencer goes offline. In the future of **zksync**, if some validators are faulty, will it be based on Byzantine Consensus (honest majority) to successfully publish the state, or would another consensus mechanisms such as Proof of Stake be more secure? What decentralization scheme works the best for achieving this progressive decentralization road-map and layering decentralization over time, and is the consensus mechanism the only security risk in mind? Another question is what would zksync 2.0 need to consider in order to decentralize the block generation as well (in addition to just the sequencers)?