Assignment 4

Q0 Which stream do you choose (answer with A or B)?

A

Q1 Scaling the future

In the blockchain there is a well known scalability trilemma. We can't have decentralized, secure and scalable L1 blockchain. Many blockchains tend to be secure and decentralized, but they lack scalability.

1. Based on the above, a number of solutions have been proposed to solve this trilemma. Briefly describe the different scalability solutions and write pros and cons of each approach. What was the biggest problem with the Plasma approach?

Answer 1.1:

The above statement isn't entirely correct, **first**, on a macro scale Bitcon "Blockchain" is a unique data structure built for transparency, observability and censorship resistance and it's main purpose is TX finality, for that Bitcoin scales perfectly, so the take away from this is market needs is what defines the scalability ceiling.

However, for alternate networks that seeks other functionalities beyond a peer-to-peer cash system:

Disadvantages include low bandwidth in comparison with centralized solutions and high cost of operations "operations complexity".

Second: for precise distinction of the solutions for this trilemma, the following options for scaling must be accounted for; consensus modification and sharding that are embedded in the **core of a blockchain**, second layer solutions and sidechains that can be built **on top of a network.**

At the core of a blockchain:

The core of any decentralized system is the consensus protocol - the rules for coordinating the state transition of the system. The level of decentralization of the system and its throughput directly depends on it.

Chain	Consensus	Security	Bandwidth	Relevancy/Market
Bitcoin	PoW	perfect	Block~10mins	perfect
Ethereum	PoW	perfect	Block~12secs	moderate
ETH2	PoS	low	64 shard blocks~6mins	moderate
Cosmos	DPoS	moderate	Block~7secs	low

On top of a network:

Changing consensus or implementing sharding requires modifying the core of the blockchain and usually a hard fork. Such deep changes require community approval, long development, and testing; therefore, such updates are rarely introduced. There is an alternative solution - the so-called second-layer solutions. They do not change the blockchain core but use only the platform's programmable logic, such as smart contracts. They allow performing calculations outside the main blockchain network (layer 1) and, accordingly, not pay a commission for these calculations, however, these protocols record the necessary information (the contents of transactions or proof of their correctness) in layer 1 to maintain the required level of security.

Solution	Example	Decentralization	Security	Scalability	Capital Effeciency	L1 Chain
Payment channels	Lightning Network	Low	High	High	Low	Agnostic
Sidechains	RSK	Moderate	High	Moderate	High	Dependent
Plasma		Low	Low	High	Low	Dependent
Optimistic Rollups	Arbitrum	Moderate	Moderate	High	Low	Dependent
ZK Rollups	ZKSync	Moderate	High	High	High	Dependent

Problems with Plasma:

- The technology leverages the human factor “Block producers” to opt out scalability issues, which leads to “in rat-race scenario” to a number of limitations associated with the need to verify a large number of transactions by the block producers posing a critical increase in the load on the main network.
- Meanwhile, Block producer acts as a centralized points of governance by gets transaction and uses it to update the merkle tree root, so in the case block producer doesn't make the transaction available and doesn't update the ledger of the Plasma chain, failures can be detected in the secondary chain, In theory, Block producers have the ability to freeze people's Plasma assets, Leading to mass outage of that Plasma network.

2. One of the solutions that has been gaining a lot of traction lately is zkRollups. With the use of a diagram explain the key features of zkRollups. Argue for or against this solution highlighting its benefits or shortcomings with respect to other solutions proposed or in use.

Answer 1.2:

ZK-rollups usually based on one of two classes of algorithms ZK-SNARK (Zero-Knowledge Succinct Non-Interactive Argument of Knowledge) or ZK-STARK (Zero-Knowledge Scalable Transparent Argument of Knowledge). Each of the cryptographic protocols has advantages and disadvantages. The main limitations of ZK-rollup:

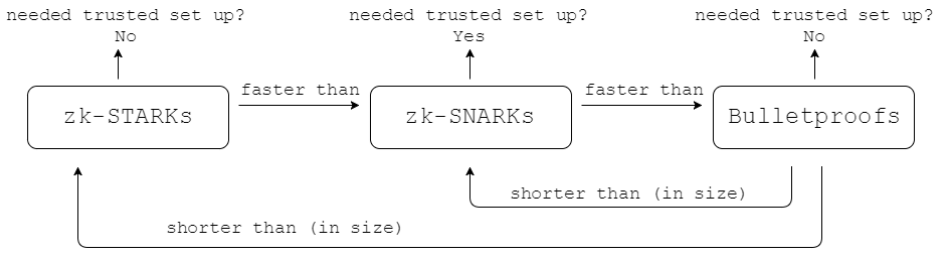
- High computational complexity of proof constructing, very hard to audit.
- The proofs remain highly specialized and are suitable only for specific types of transactions

however the provability of arbitrary program logic's execution is starting to surface like ZKSync EVM.

A diagram highlighting benefits and shortcomings

	SNARKs	STARKs	Bulletproofs
Algoritmic complexity: prover	O(N * log(N))	O(N * poly-log(N))	O(N * log(N))
Algoritmic complexity: verifier	~O(1)	O(poly-log(N))	O(N)
Communication complexity (proof size)	~O(1)	O(poly-log(N))	O(log(N))
- size estimate for 1 TX	Tx: 200 bytes, Key: 50 MB	45 kB	1.5 kb
- size estimate for 10.000 TX	Tx: 200 bytes, Key: 500 GB	135 kb	2.5 kb
Ethereum/EVM verification gas cost	~600k (Groth16)	~2.5M (estimate, no impl.)	N/A
Trusted setup required?	YES 😒	NO 😄	NO 😄
Post-quantum secure	NO 😒	YES 😄	NO 😒
Crypto assumptions	Strong 😒	Collision resistant hashes 😄	Discrete log 😒

Zero-Knowledge trilemma:



SNARKs:

- There has been significant debate on whether there is a backdoor into elliptic curve random number generators.
- Quantum attacks do loom over cryptography based on elliptic curves.
- zk-SNARKs also require a trusted set up, there is a hidden parameter linked between the verification key and the keys sending private transactions.

STARKs:

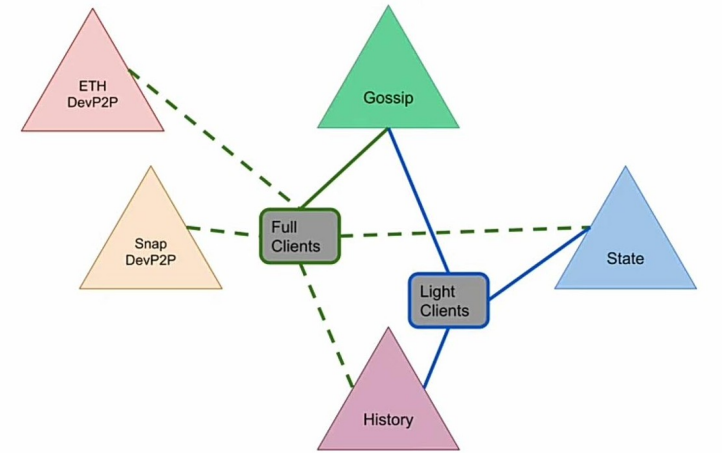
- Unlike SNARKs, the base technology for STARKs relies on hash functions.
- No trusted set-up is required to begin utilizing STARKs in a network.
- STARKs have far larger proof sizes than SNARKs, which means that verifying STARKs takes more time than SNARKs and also leads to STARKs requiring more gas.

3. Ethereum is a state machine that moves forward with each new block. At any instance, it provides a complete state of Ethereum consisting of the data related to all accounts and smart contracts running on the EVM. The state of Ethereum modifies whenever a transaction is added to the block by changing the balances of accounts. Based on the massive adoption of Ethereum across the globe, this state has become a bottleneck for validators trying to sync with the network as well as validate transactions. Briefly describe the concept of stateless client, and how they help resolve this issue? Explain how Zero-Knowledge improves on the concept of stateless client?

Answer 1.3:

The concept of stateless client

I guess there will never be a total stateless node in the current Ethereum theme, for the fact that there must be a level of knowledge a node must retain to fetch the EVM chain and compare it to it's own "ledger", however a generalization of this problem might drift the developer community away from a viable solution, as some of Ethereum top developers seeks shardning the underlying protocol to three protocols contrary to the already know "sharding the chain", they research a way to divide "Democratizing Ethereum protocol" the protocol to (Gossip, State, and History) protocols, and push the dApp developers "each relative to it's own services" to Push/Pull APIs native to which "protocol shard" they wish without offloading other shards, In this scenario developers might back to use "In some cases" HTTP/TCP APIs other than JSON/RPCs.



The concept of semi-stateful node

A semi-stateful node (not client) can be a success only in case if its securely maintains a level of knowledge for the chain state changes like, Hash of the block header, merkle root of the current state, or a zero-knowledge proof of either, exactly where Zero-Knowledge primitives comes useful, as they can provide computational Integrity "perfect for zero-trust" and succinctness "perfect for scalability".

Q2 Roll the TX up

1. **[Infrastructure Track only]** Review the RollupNC source code in the learning resources focusing on the contract and circuit and explain the below functions (Feel free to comment inline)
    1. UpdateState (Contract)
    2. Deposit (Contract)
    3. Withdraw (Contract)
    4. UpdateStateVerifier (Circuit)

    Propose possible changes that can be made to the rollup application to provide better security and functionalities to the users

Answer 2.1:

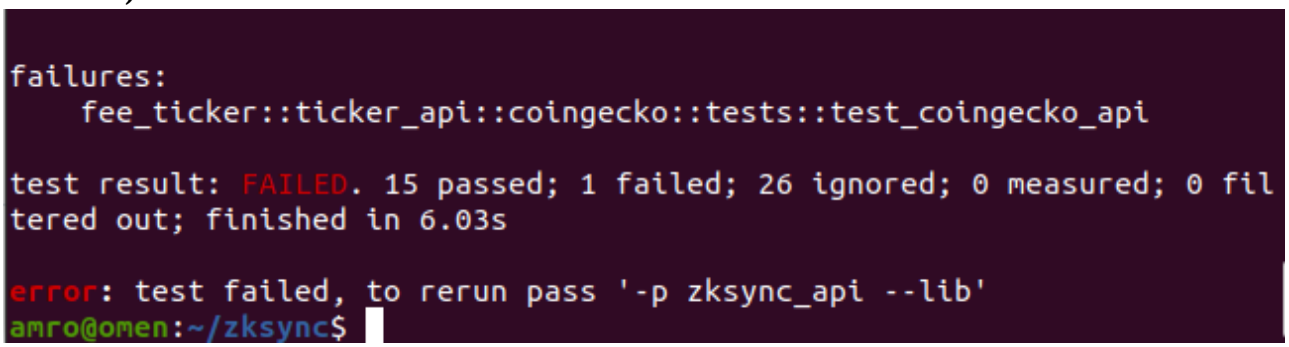
Link (Contract): https://github.com/amrosaeed/zku/blob/main/Assignment4_Q2/Q2.1/RollupNC.sol
Link (Circuit):
https://github.com/amrosaeed/zku/blob/main/Assignment4_Q2/Q2.1/update_state_verifier.circom

2. **[Infrastructure Track only]** Clone a copy of the ZKSync source code and run the unit test on it. Submit a screenshot of the test result and justification in the event any of the tests fails.

Answer 2:

Screenshots: (**Testing)**

- **Running the rust unit-tests (heavy tests such as ones for circuit and database will not be run):**



Justification for test failure:

After tokio client implemented the initial authentication handshake and talks to the remote server “coingecko” a runtime is dropped from within zksync asynchronous context as it depends on nested runtimes (nested async), also sometime it happens with fetching coinmarketcap API and sometimes it works.

- **Running the database tests:**

```
test result: ok. 75 passed; 0 failed; 0 ignored; 0 measured; 1 filtered out; fin
ished in 1.86s

   Doc-tests zksync_storage

running 0 tests

test result: ok. 0 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; fini
shed in 0.00s

amro@omen:~/zksync$
```

Justification for test failure:

A proper database test will require more than a local database instance, as so tests didn't even run

- **Running the integration test: (failed test)**

Justification for test failure:

I've been talking to ZkSync community members and devs on discord over the bast week, what I concluded from them about the main reason for some tests to fail like (Integration test, benchmarks and loadtests) was the in-feasibility to run tests locally with local data base instance with local server instance, as they mentioned the need for a specialized database instance.

- **Running the circuit tests: (won't work, Atleast 64~196 GB RAM needed)**

Justification for test failure:

I tried to upgrade my machine to 64 GB RAM, however aside it will take some time to get the right hardware from vendor, a community head mentioned that some circuits tests require up to 196 GB RAM.

- **Running the prover tests:**

```
    Running tests/tests.rs (target/release/deps/tests-ac590807ce300de4)

running 3 tests
test test_shutdown_request ... ok
test test_receiving_heartbeats ... ok
test test_publishing_proof ... ok

test result: ok. 3 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; fini
shed in 10.24s
```

- **Running the benchmarks:**

```
Gnuplot not found, using plotters backend
Benchmarking get_txs_batch_fee: Warming up for 3.0000 sthread 'main' panicked at
 'called `Result::unwrap()` on an `Err` value: reqwest::Error { kind: Request, u
rl: Url { scheme: "http", cannot_be_a_base: false, username: "", password: None,
 host: Some(Ipv4(127.0.0.1)), port: Some(3001), path: "/api/v0.2/fee/batch", que
ry: None, fragment: None }, source: hyper::Error(Connect, ConnectError("tcp conn
ect error", Os { code: 111, kind: ConnectionRefused, message: "Connection refuse
d" })) }', core/bin/zksync_api/benches/criterion/lib.rs:29:10
stack backtrace:
   0: rust_begin_unwind
             at /rustc/9d1b2106e23b1abd32fce1f17267604a5102f57a/library/std/src/
panicking.rs:498:5
   1: core::panicking::panic_fmt
             at /rustc/9d1b2106e23b1abd32fce1f17267604a5102f57a/library/core/src
/panicking.rs:116:14
   2: core::result::unwrap_failed
             at /rustc/9d1b2106e23b1abd32fce1f17267604a5102f57a/library/core/src
/result.rs:1690:5
   3: api_service::get_txs_batch_fee
   4: criterion::bencher::Bencher<M>::iter
   5: <criterion::routine::Function<M,F,T> as criterion::routine::Routine<M,T>>:
:warm_up
   6: criterion::routine::Routine::sample
   7: criterion::analysis::common
   8: criterion::benchmark_group::BenchmarkGroup<M>::bench_function
   9: api_service::main
note: Some details are omitted, run with `RUST_BACKTRACE=full` for a verbose bac
ktrace.

error: bench failed
```

Justification for test failure:

For benchmarks tests as for Integrations tests it in-feasible to connect both the server, dummy-prover and the test console to the same instance of the naive local database, a certain database setup required to do the test properly.

- **Running the loadtest: (failed test)**

Justification for test failure:

Also for the loadtest tests as for both Integrations/benchmarks tests it was in-feasible to connect both the server, dummy-prover and the test console to the same instance of the naive local database, a certain database setup required to do the test properly.

3. Review the source code of ZKSync and provide detailed explanations for the below functionalities Docs (You can use either pseudocode or comment inline):
    1. How the core server maintains transactions memory pool and commits new blocks?

Answer 2.1.1:

Link : https://github.com/amrosaeed/zku/blob/main/Assignment4_Q2/Q2.2/block_proposer.rs

    2. How the eth_sender finalizes the blocks by sending corresponding Ethereum transactions to the L1 smart contract?

Answer 2.1.2:

Link : https://github.com/amrosaeed/zku/blob/main/Assignment4_Q2/Q2.2/eth_sender.rs

    3. [Bonus] How the witness_generator creates input data required for provers to prove blocks?

Answer 2.1.3:

Link : https://github.com/amrosaeed/zku/blob/main/Assignment4_Q2/Q2.2/witness_generator.rs

4. **[All Tracks]** ZKSync 2.0 was recently launched to testnet and has introduced ZKPorter. Argue for or against ZKPorter, highlighting the advantages or shortcomings in this new protocol.

Answer 2.3:

After launching ZKPorter, zkSync 2.0 will present two main services for developers to chose regarding their dApps priorities, In an effort trying to fill the gap in market and compete with their rivals starkware solutions (StarkNet & Validium), two flavours for two different markets (Data-Sensitive / Data-Insensitive) dApps.

- ZKRollup smart contract, with full on-chain data availability but higher fees, for Data-Sensitive dApps with regular fees (quite lower than current ETH gas fees).
- ZKPorter, with off-chain (off Eth L1) data availability guaranteed by zkSync PoS staking nodes, for Data-Insensitive dApps, with 10x lower fees.

Developers will have to decide to market their dApps relative to their core services sensitivity, whether the dApp requires maximum security regarding data availability on-chain for all the network participants to observe, in that case they will leverage zkSync 2.0 zkrollup for full data availability (to prevent state being inaccessible in the event of important transactions), On the other sides for dApps that less sensitive operations-data dependency like NFTs, gameFi, they will leverage ZKPorter with 10x less transaction throughput.

Q3 Recursive SNARK's

1. Why would someone use recursive SNARK's? What issues does it solve? Are there any security drawbacks?

Answer 3.1:

Why would someone use recursive SNARK's?

In blockchain's world a "Zero-Trust environment" consensus is the bottle-neck in such distributed systems, As a consequence Developers always seeks preserving transactions throughput as the only scheme of mirroring state changes without sacrificing "Zero-Trust". For such problem Zero-Knowledge proofs as cryptographic primitives shined for it's both privacy & succinctness effect on concluding state changes in a transaction, the possibility of proof recursion without a costly trusted setup like in case of SNARKs promised a new ceiling for scalability Blockchains can achieve.

What issues does it solve?

There are a number of use cases for recursive composition. One of the most well-known at the moment is that of Coda, a fully-succinct blockchain protocol allowing clients to validate the entire chain by checking a small cryptographic proof in the tens of kilobytes. Recursive proofs have also been proposed for other blockchain scaling solutions, such as recursively proving validity of Bitcoin's proof of work consensus algorithm, and at Celo using bounded recursion to efficiently verify aggregated signatures for mobile clients.

Are there any security drawbacks?

- A relevant to computation immplemention of recursive SNARKs is not post-quantum secure.
- SNARKs Trusted setups will always be a security drawback in comparison to STARKs.
- The choice of elliptic curve and arithmetic circuit used can be prone to errors.
- Harder to audit.

2. What is Kimchi and how does it improve PLONK?

Answer 3.2:

Kimchi is a collection of improvements, optimizations, and alterations made on top of PLONK, I***nside the recursion layer called Pickles***, Kimchi is aimed at improving PLONK.

Pickles, the recursion layer, is the protocol that MINA uses to create proofs of proofs of proofs of … to reduce the blockchain to a fixed size of under 22KB.

- Firstly, it overcomes the trusted setup problem by "using a bulletproof-style polynomial commitment inside of the protocol". This way, there is no risk that the setup might be done improperly.
- Kimchi has 15 registers, compared to 2 for PLONK.Having, 15 registers allow for more inputs inside a gate. Therefore, this allows for hash function gates (Poseidon), elliptic curves operations gates, and much more.
- That gates can directly write their outputs to the registers of the next gate, this allows for faster iterative computation like that of Poseidon's.
- Lookup tables, they allow for large logic gates to be expressed as a table, and operations within it to be looked up. This makes operations way faster.

3. **[Infrastructure Track only]** Clone github repo. Go through the snapps from the src folder and try to understand the code. Create a new snapp that will have 3 fields. Write an update function which will update all 3 fields. Write a unit test for your snapp.

Answer 3.3:

Link: https://github.com/amrosaeed/zku/blob/main/Assignment4_Q3/ThreeFields.ts

4. [Bonus] In bonus.ts we can see the implementation of zkRollup with the use of recursion. Explain 87-148 lines of code (comment the code inline).

Answer 3.4:

Link: https://github.com/amrosaeed/zku/blob/main/Assignment4_Q3/bonus.ts

Q4 Final Project Ideas [Application Track Only]

1. Describe 2-3 different, independent ideas for your final project. For each, describe the idea in about 150 words, explaining which aspect of ZK you are capitalizing on (privacy, succinctness, or fairness), and why this may be a timely product for the market. See the advice below for guidance and inspiration.
2. Compare the pros and cons of the ideas, and try to make a strong case for each project. Describe some shortcomings including feasibility given the time frame and existence of competing products, as well as how you may be able to address these shortcomings.

Answer 4: Stream A

Question 5: Thinking in ZK

1. If you have a chance to meet with the people who built ZKSync and Mina, what questions would you ask them about their protocols?

Answer 5.1:

What mostly I’m interested in, is the existence of a viable Full-light client node Implementation for Android system, to query a smart contract chain state, so mainly my question would be: what’s the development state for building a MINA-ZKsync bridge, so it would be possible to run a MINA full node on low-resources devices that can query ZKsync chain?