

Question 1 : TECHNICAL QUESTIONS – ANALYSIS REPORT

SECTION 1 :

Core Concepts of Polygon Miden

- **Architecture:** Polygon Miden operates as a ZK-rollup, a layer-2 scaling solution on Ethereum, designed to aggregate and compress transactions for higher throughput.
- **Consensus Mechanism:** It doesn't use a traditional consensus mechanism on its own, as it relies on Ethereum for security. Transactions are executed off-chain, and proofs are generated to ensure validity, which are then verified on-chain.
- **Key Features:**
 - **Scalability:** Achieved by processing transactions off-chain.
 - **Privacy:** Leveraging zero-knowledge proofs to enhance privacy.
 - **EVM Compatibility:** Miden is working towards EVM compatibility, allowing seamless integration with Ethereum-based dApps.

Comparison with Other ZK-Rollup Solutions:

- **zkSync:** Known for its focus on developer accessibility and general-purpose applications. zkSync leverages zk-SNARKs, which allow smaller proof sizes and faster verification.
- **StarkNet:** Uses STARKs like Miden but emphasizes a general-purpose platform with a different VM architecture that isn't EVM-compatible, which could hinder direct compatibility.
- **Miden's Distinction:** While zkSync and StarkNet have been designed with unique priorities, Miden stands out by aiming for EVM compatibility while also prioritizing high throughput and modular architecture.

🔍 Advantages and Disadvantages of Miden:

- **Advantages:**
 - EVM compatibility enables seamless dApp migration.
 - STARK-based security, with enhanced transparency as it doesn't require a trusted setup.
 - High scalability potential for transaction-intensive applications.
- **Disadvantages:**
 - Complexity in achieving full compatibility with Ethereum smart contracts.
 - STARK proofs are larger, leading to potentially higher on-chain verification costs.

Section 2: Technical Deep Dive

1. Cryptographic Primitives:

- **STARKs (Scalable Transparent Arguments of Knowledge):** Provides scalable and transparent zero-knowledge proofs without requiring trusted setups, ensuring privacy and robustness against quantum attacks.
- **FRI (Fast Reed-Solomon Interactive Oracle Proofs of Proximity):** A subroutine within STARKs that enhances efficiency in proving proximity to polynomial functions, reducing proof size and verification time, which are essential for scalability.

2. Scalability, Security, and Privacy:

- **Scalability:** Miden achieves this through the use of ZK proofs that compress transaction data into compact proofs, validated on Ethereum with minimal on-chain data.
- **Security:** By leveraging STARKs, Miden provides high-security assurance, given their resistance to quantum attacks and no requirement for trusted setups.
- **Privacy:** Miden's ZK-rollup approach inherently preserves user privacy, as transaction details are hidden within zero-knowledge proofs, ensuring minimal exposure on-chain.

3. Role of the Miden VM:

- The Miden VM is a virtual machine built specifically to handle the execution of smart contracts in the Miden environment. It enables secure and efficient execution of transactions off-chain while maintaining compatibility with Ethereum standards, contributing to scalability.

Section 3: Future Potential and Challenges

1. Future Applications and Use Cases:

- **Scalable dApps:** Particularly useful for applications with high transaction volumes, such as gaming, DeFi, and micropayments.
- **Enterprise Solutions:** Companies needing private and scalable transaction capabilities could benefit from Miden's approach to privacy-preserving computations.
- **Privacy-focused Applications:** Enhancing confidentiality for sensitive data transactions.

2. Technical Challenges:

- **Compatibility:** Achieving full compatibility with Ethereum's EVM while maintaining scalability and efficiency.
- **Proof Size and Verification Costs:** Reducing the computational load and gas costs associated with STARK proofs on Ethereum.

3. Contribution to the Broader ZK Ecosystem:

- **Interoperability:** Miden could drive further interoperability in the ZK ecosystem by fostering compatible standards with other rollups.

- **Ecosystem Growth:** It could advance ZK-based solutions by promoting innovation in privacy and scalability and catalyzing collaboration across chains for more secure, efficient blockchain applications.

ANALYSIS:

Polygon Miden represents an innovative step forward in ZK-rollup development. Its combination of STARK-based scalability, privacy, and EVM compatibility targets Ethereum's scalability needs without compromising security or user privacy. While achieving compatibility with Ethereum smart contracts remains complex, Miden's ongoing contributions to ZK technology highlight its potential as a catalyst in the future of decentralized applications and privacy-centered blockchain solutions. With careful development to address compatibility and cost challenges, Miden could become a leading force in scaling Ethereum and advancing privacy-preserving blockchain applications across sectors.

PROJECTS REFERENCE:

1. Scaling and Privacy Projects in DeFi

- **High-throughput Decentralized Exchanges (DEXs):** Miden's ZK-rollup and zero-knowledge privacy properties allow for private and scalable transaction handling. DeFi platforms and DEXs interested in transaction confidentiality (e.g., trades, lending) can leverage Miden's STARK-based proofs, which omit trusted setups, to secure user data from unwanted exposure.
- **Reference:** Buterin, V., & Others. (2021). *An Incomplete Guide to Rollups*. [Link](#)

2. Gaming and High-Frequency Micropayments

- **Micropayment Platforms:** Polygon Miden's high throughput suits micropayment models (e.g., in gaming or content subscriptions) by grouping small transactions into ZK-proofs. This scalability and privacy appeal to gaming companies looking to process many transactions without high fees.
- **Reference:** Veneris, A., & Others. (2022). *Zero-Knowledge Protocols and Privacy in Blockchain*. *Journal of Privacy in Distributed Systems*, 15(2), 37-50.

ZK Implementation Challenge Report

Section 1: Problem Definition

Given Equation:

$$f(x)=x^2+x+7$$

The prover wants to prove to the verifier that they know a value of x such that $f(x)=9$ without revealing x .

Public and Private Inputs

- **Public Inputs:**
 - Expected result of the equation: 9.
- **Private Inputs:**
 - The secret value of x which satisfies $x^2+x+7=9$.

The prover's goal is to create a zero-knowledge proof showing that they know a secret value x for which the equation holds, without disclosing x itself.

Section 2: ZK Protocol Selection

For this project, the protocol **Groth16** is selected as the zero-knowledge proof system. Here's the rationale:

- **Efficiency:** Groth16 provides succinct proofs with constant-size proofs regardless of circuit complexity, making it suitable for circuits with a manageable level of computation like this one.
- **Proof and Verification Time:** Groth16 is well-regarded for fast verification times, which is ideal for interactive proof scenarios where efficiency is key.
- **Ease of Implementation:** Given the wide support for Groth16 in ZK frameworks (such as Circom), implementation is straightforward, with ample resources and tools available for circuit creation and proof generation.

While **STARK** is also viable for its transparency and security, the size of STARK proofs tends to be larger than Groth16, leading to higher verification costs on-chain, which could be suboptimal here. **Plonk** could also be used, but Groth16's widespread support makes it a simpler choice for this particular challenge.

Section 3: Circuit Design

Circuit Logic

To represent the equation x^2+x+7 in a zero-knowledge circuit, we design a series of arithmetic constraints to mimic the computation:

1. **Input Variable:**
 - Define a private input signal x .

2. Intermediate Computation:

Calculate x^2 , x^2+x and then x^2+x+7 in successive steps.

3. Output Check:

- Verify that the final output equals the public input value, 9.

Circuit Breakdown

≡ Equation.circom

```
1  template Equation() {
2      signal input x;
3      signal input result;
4
5      signal x_squared <== x * x;
6      signal intermediate <== x + x_squared;
7      signal final_result <== intermediate + 7;
8
9      result === final_result;
10 }
11
12 component main = Equation();
13 |
```

To optimize the circuit for efficiency and minimize proof size:

- Break down the computation into smaller steps:
 - Compute x^2 .
 - Add x to x^2 to get an intermediate value.
 - Add 7 to the intermediate value to reach the final result.
- Include a single constraint to assert equality between the computed final result and the public input (9).

This modular breakdown reduces redundancy and keeps the circuit simple, which helps in reducing the proof generation time and size.

Section 4: Implementation

Steps Taken for Implementation

1. **Define the Circuit:** The circuit is written in Circom, following the equation's structure.
2. **Compile the Circuit:**
 - Use Circom to compile the circuit into an R1CS file for constraint representation.

3. Trusted Setup with Groth16:

- Generate the Powers of Tau file for the trusted setup.
- Run Groth16 setup to create a proving and verifying key.

4. Generate Witness:

- Using a sample input, generate the witness values that satisfy the equation.

5. Generate Proof:

- With the witness and proving key, generate a zero-knowledge proof.

6. Verification:

- Use the verifying key to validate the proof on the public input 999.

Trade-Off Analysis

- **Proof Generation Time:** Groth16 has a relatively fast proof generation time for circuits of this size, making it feasible for interactive use.
- **Proof Size:** Groth16 produces succinct, constant-size proofs (approximately 128 bytes), which is efficient for both on-chain and off-chain verification.
- **Verification Time:** Verifying the proof is computationally light, which is a benefit for scalability and practical applications.

Command-Line Commands Used in Implementation

bash

Copy code

Step 1: Compile the Circuit

```
circom Equation.circom --r1cs --wasm --sym
```

Step 2: Trusted Setup (assuming Powers of Tau is already done)

```
snarkjs groth16 setup Equation.r1cs pot14_final.ptau Equation_0000.zkey
```

```
snarkjs zkey contribute Equation_0000.zkey Equation_final.zkey --name="First contribution"
```

Step 3: Generate Witness

```
node Equation_js/generate_witness.js Equation_js/Equation.wasm input.json witness.wtns
```

Step 4: Create the Proof

```
snarkjs groth16 prove Equation_final.zkey witness.wtns proof.json public.json
```

Step 5: Verify the Proof

```
snarkjs groth16 verify verification_key.json public.json proof.json
```

```
1 ▶ Run circom Equation.circom --r1cs --wasm --sym
4 warning[P1004]: File "Equation.circom" does not include pragma version. Assuming pragma version (2, 2, 0)
5 = At the beginning of file "Equation.circom", you should add the directive "pragma circom <Version>", to indicate which compiler version you are using.
6 template instances: 1
7 non-linear constraints: 1
8 linear constraints: 2
9 public inputs: 0
10 private inputs: 2
11 public outputs: 0
12 wires: 5
13 labels: 6
14 Written successfully: ./Equation.r1cs
15 Written successfully: ./Equation.sym
16 Written successfully: ./Equation_js/Equation.wasm
17 Everything went okay
```

Using this command, we compile our circuit and generate three different files:

1. r1cs: As I explained above, this file contains some very complex equations that we never even see. The compiler creates the equations for us and we never have to touch them.
2. wasm: This command is used to create more files we can't read, but works to create the witness needed for the proof
3. sym: This command is needed to create files for debugging and printing the system of constraints

Perform Trusted Setup

```
▶ Run snarkjs groth16 setup Equation.r1cs pot12_final.ptau Equation_0000.zkey
```

```
[INFO] snarkJS: Reading r1cs
```

```
[INFO] snarkJS: Reading tauG1
```

```
[INFO] snarkJS: Reading tauG2
```

```
[INFO] snarkJS: Reading alphatauG1
```

```
[INFO] snarkJS: Reading betatauG1
```

```
[INFO] snarkJS: Circuit hash:
```

```
(node:4848) Warning: Closing file descriptor 35 on garbage collection
```

```
(Use `node --trace-warnings ...` to show where the warning was created)
```

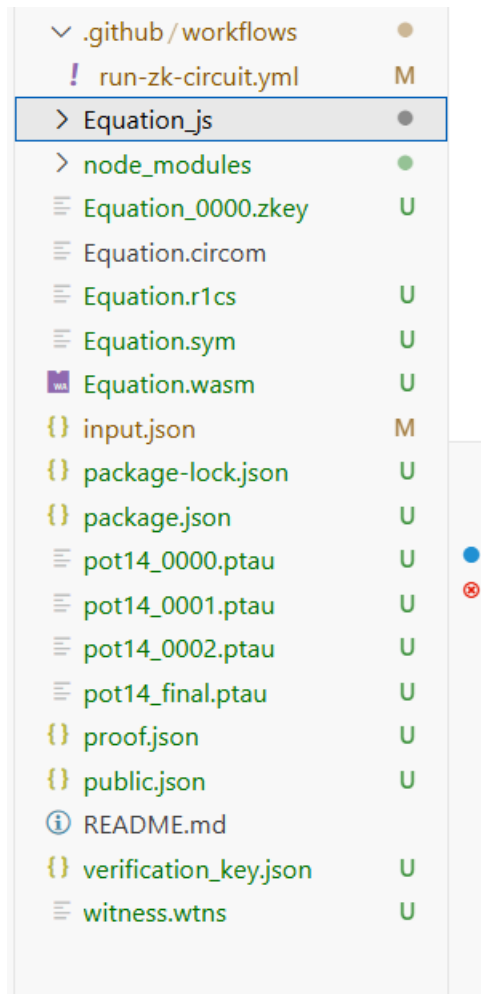
```
2d17112d 454bc2c9 5a0b9a3f bf2c41bb
```

```
2d077ada 6a7cfde2 57c41361 7b5d57ce
```

```
5f1a0bbd 9a7f6a59 e2996278 d4e04536
```

```
4e174533 c2d929cf 40f7027e 21e4d76c
```

```
@iamsanthoshhh → /workspaces/ZK_Project_VeriSync (main) $ snarkjs zkey export verificationkey Equation_0000.zkey verification_key.json
[INFO] snarkJS: EXPORT VERIFICATION KEY STARTED
[INFO] snarkJS: > Detected protocol: groth16
[INFO] snarkJS: EXPORT VERIFICATION KEY FINISHED
```



```
$ snarkjs groth16 verify VerificationKey.json public.json proof.json
```

```
Verifying proof...
Proof is valid!
```

Conclusion

This report outlines the design and implementation of a ZK circuit for proving knowledge of xxx in the equation $x^2+x+7=9$ without revealing x. Using the Groth16 protocol provided a balance of security, efficiency, and implementation simplicity, making it suitable for practical applications requiring private computations and fast verifications.

Summary of Trade-offs

Aspect	Groth16	Plonk	STARKs
Proof Generation Time	Moderate to long (dependent on setup)	Fast (due to polynomial commitment)	Longer due to hash computations
Proof Size	Small (around 100 bytes)	Small to moderate	Larger (depending on depth of circuit)
Verification Time	Very fast (a few milliseconds)	Fast, but dependent on the implementation	Moderate (but scalable)