Benchmarking the Plonk, TurboPlonk, and UltraPlonk Proving Systems

Lehigh University

Tal Derei, Caleb Geren, Michael Kaufman, Jon Klein, Rishad Islam Shantho

February 2023

1 Abstract

We evaluate the performance and scalability of the Plonk, TurboPlonk and UltraPlonk zero-knowledge proof systems using custom gates based on pedersen hashes. This expands on our prior performance benchmarks of Plonk [1]. All measurements are obtained using Aztec's Barretenberg cryptographic library and backend [2] using CPUs.

2 Background

TurboPlonk generalizes the constraint system by introducing custom gates that can represent complicated statements with fewer gates in a circuit [3]. For instance, cryptographic primitives like a fixed-base elliptic curve scalar-multiplication, elliptic curve point arithmetic, and 8-bit logical XORs can be expressed and evaluated with a single custom gate. UltraPlonk extends this construction with precomputed lookup tables, which represent efficient key-value mappings [4]. This enables a prover to prove that a witness is in a table instead of proving the computation itself.

SNARK-friendly hashing algorithms like Pedersen are crucial because they are used in polynomial commitment schemes and provide collision resistance. Additionally, hashes dominate **99%** of the computation for merkle-tree calculations. For example, adding a single UTXO note to the note and nullifier merkle trees on Aztec costs **60** hashes in total. For context, each SHA-256 hash in PLONK requires **~27,000** gates, consuming **1.6m** gates [5].

3 Cloud Computing Environment

We measure the relative speed up in performance across the following bare-metal machine instantiated on **Oracle Cloud**: 32-core Intel(R) Xeon(R) Platinum 8358 CPU @ 2.60GHz Base Frequency, 1024 GB DDR4 DRAM, 128 GB SSD.

4 Experimental Results

The raw benchmarks can be found here [6].

We measure performance using the Oracle Cloud Infrastructure (OCI) service and real-time process monitoring tools. We enabled multithreading using the OpenMP API, BMI2 x86-64 assembly instructions, and Clang compiler optimizations. The maximum constraint size for the circuits is 2^{26}. Our results extend Aztec's benchmarks [7] with larger circuit sizes and a greater number of hashes.

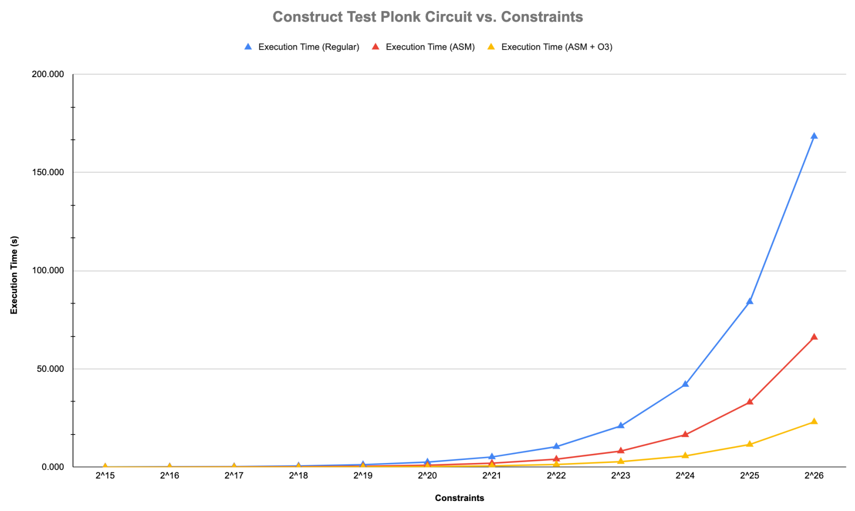
We are benchmarking the workload for the prover, which is divided into multiple tasks:

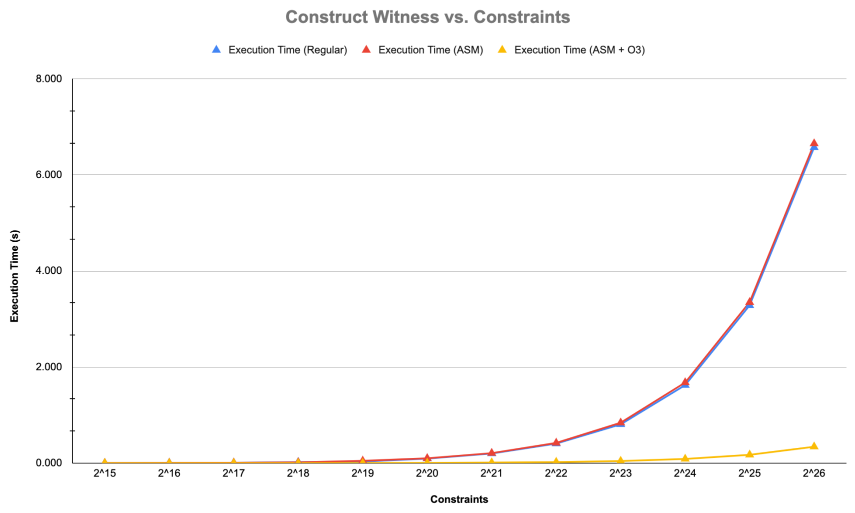
1. Construct the arithmetic circuit
2. Calculate witness polynomials
3. Compute the proving key (including q_l, q_r, etc. and sigma polynomials)
4. Compute the verifier key
5. Generate proofs
6. Verify proofs

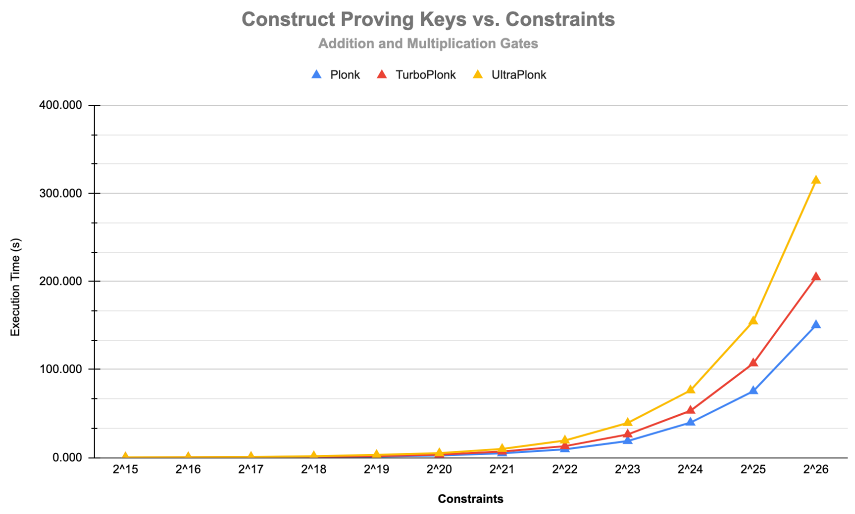
4.1 Addition and Multiplication Gates

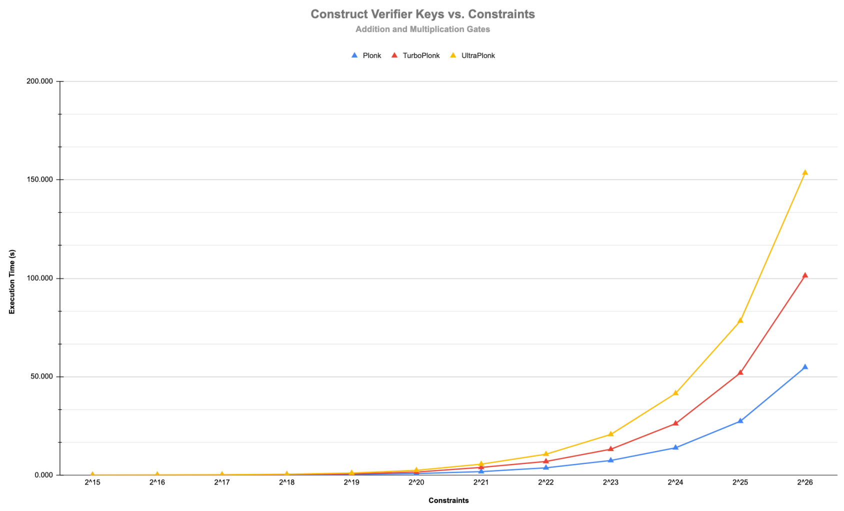
The following charts highlight the performance of Plonk, TurboPlonk, and UltraPlonk using addition and multiplication gates. TurboPlonk and UltraPlonk exhibit worse performance than Plonk with only multiplication and addition gates since they are structured to optimize performance in the presence of custom gates. For 2^{26} constraints, generating a TurboPlonk proof **(~138s)** is **37%** slower and UltraPlonk **(~165s)** is **63%** slower compared to generating a Plonk proof **(101s)**. UltraPlonk is **19%** slower than TurboPlonk in the same setting.

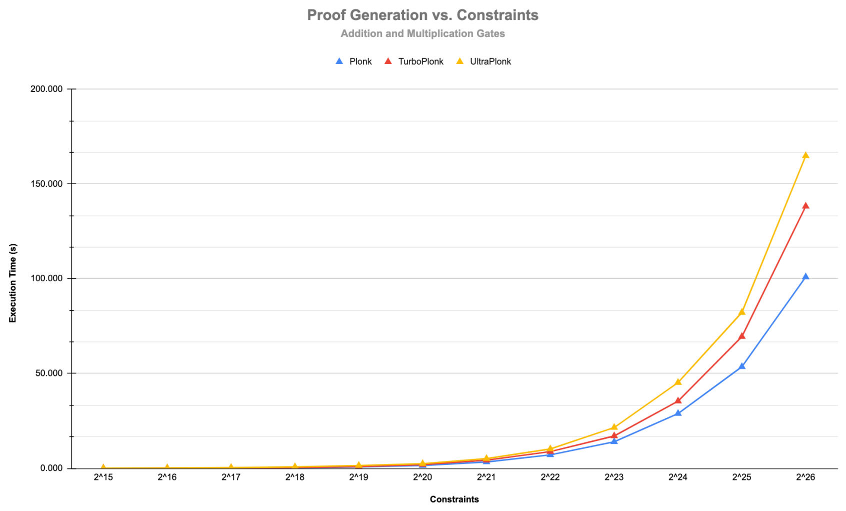
Figures 1 – 7: Prover workloads for multiplication and addition gates

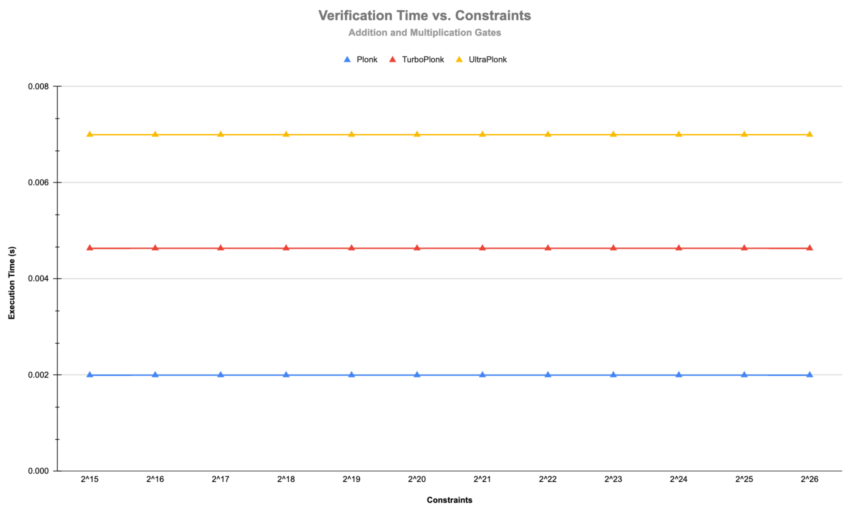


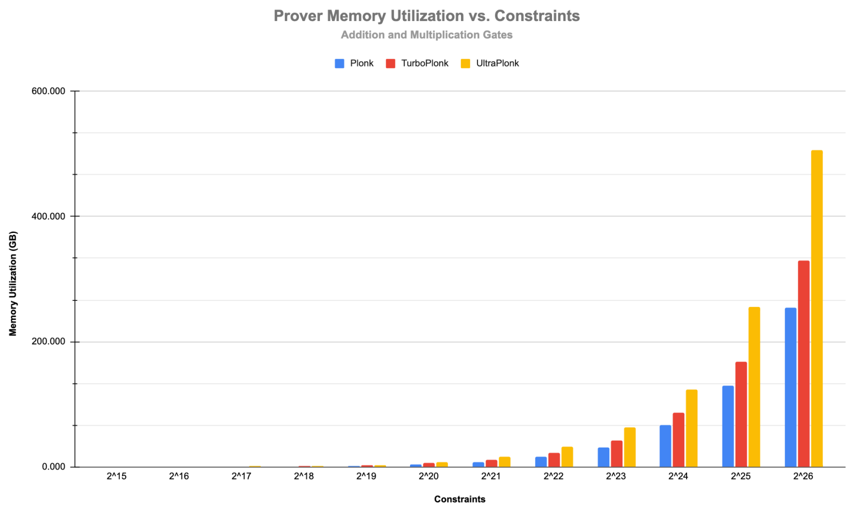



Construct Proving Keys vs. Constraints
Addition and Multiplication Gates
Plonk
TurboPlonk
UltraPlonk
Execution Time (s)
400.000
300.000
200.000
100.000
0.000
2^15
2^16
2^17
2^18
2^19
2^20
2^21
2^22
2^23
2^24
2^25
2^26
Constraints


Construct Verifier Keys vs. Constraints
Addition and Multiplication Gates
Plonk
TurboPlonk
UltraPlonk
Execution Time (s)
200.000
150.000
100.000
50.000
0.000
2^15
2^16
2^17
2^18
2^19
2^20
2^21
2^22
2^23
2^24
2^25
2^26
Constraints


Proof Generation vs. Constraints
Addition and Multiplication Gates
Plonk
TurboPlonk
UltraPlonk
Execution Time (s)
200.000
150.000
100.000
50.000
0.000
2^15
2^16
2^17
2^18
2^19
2^20
2^21
2^22
2^23
2^24
2^25
2^26
Constraints





TurboPlonk and UltraPlonk also have larger memory utilization footprints than Plonk. For 2^{26} constraints, generating a TurboPlonk proof consumed **29.5%** more memory **(330 GB)** and UltraPlonk consumed **~99%** more memory **(506 GB)** with respect to generating a Plonk proof **(255 GB)**.

In the chart below, the multi-scalar multiplication for 2^{26} constraints takes **~6.7** seconds with assembly instructions and compiler optimizations enabled, consuming **45.5 GB** of memory. The prover memory utilizations of **255 GB, 330 GB, and 506 GB** are significantly higher than the multi-scalar multiplication memory of **45.5 GB.** Plonk computes a collection of polynomials and stores each in three forms:

1. Coefficient form: **n * p**
2. Lagrange form: **n * p**
3. Coset-FFT form: **4n * p**

This yields a total memory usage of **6n * p**. It's possible to run the plonk prover by storing just the coefficient form, which would reduce the memory utilization by a factor of **6**. This aligns with the memory utilization for computing the multi-scalar multiplication. This comes at an additional runtime computational cost from not pre-computing and storing the polynomials in all three forms.

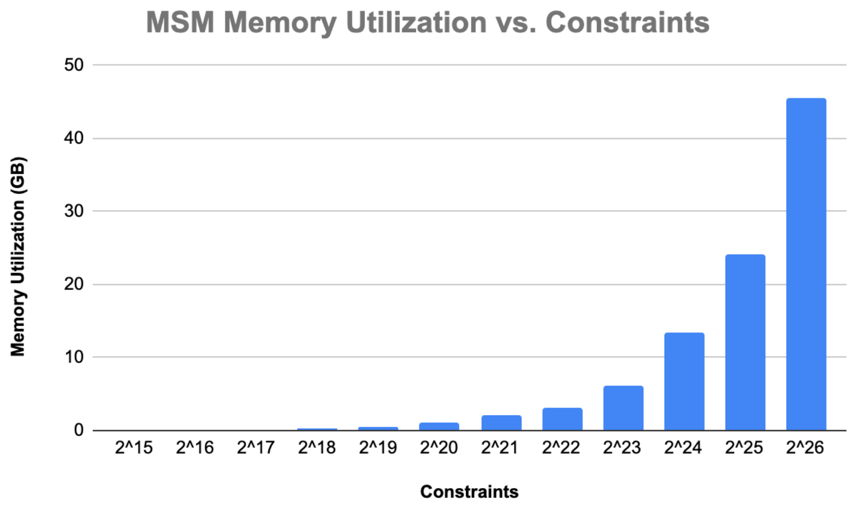


Figure 8: Multi-scalar multiplication (MSM) memory utilization

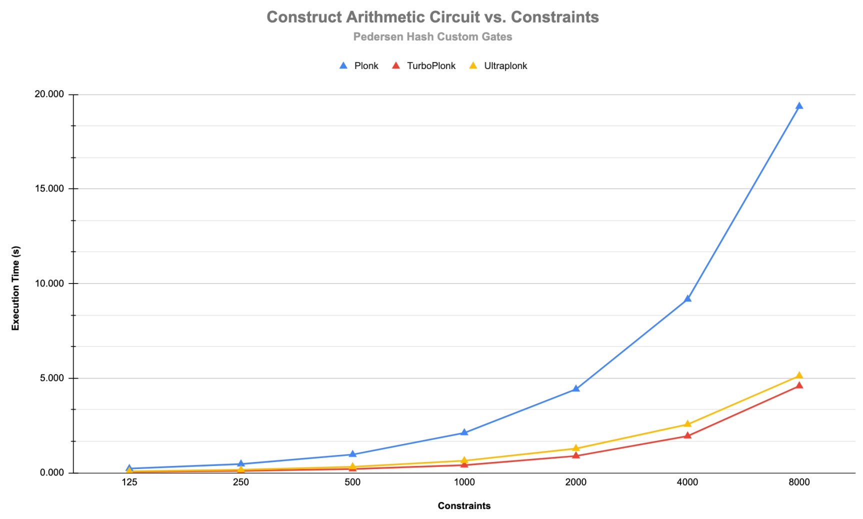
4.2 Custom Gates and Lookup Tables

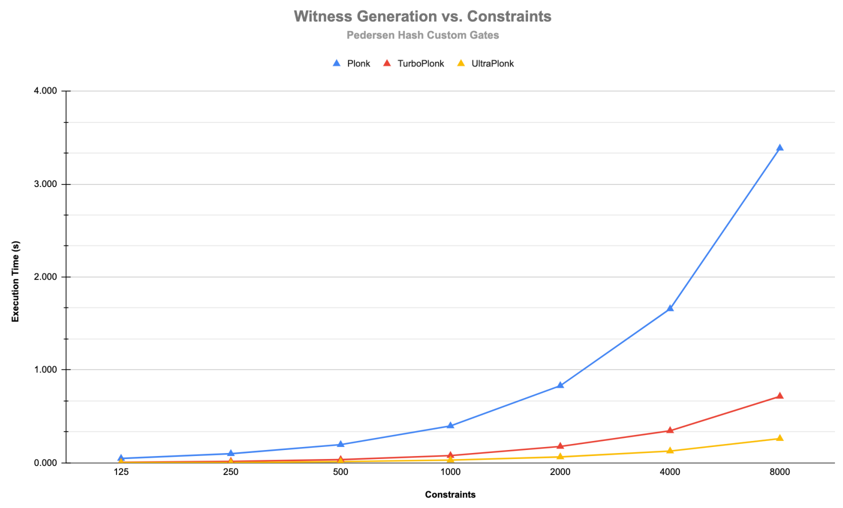
The following charts highlight the performance of Plonk, TurboPlonk, and UltraPlonk using custom hash gates and lookup tables. This will improve the performance and memory usage for the TurboPlonk and UltraPlonk proving systems.

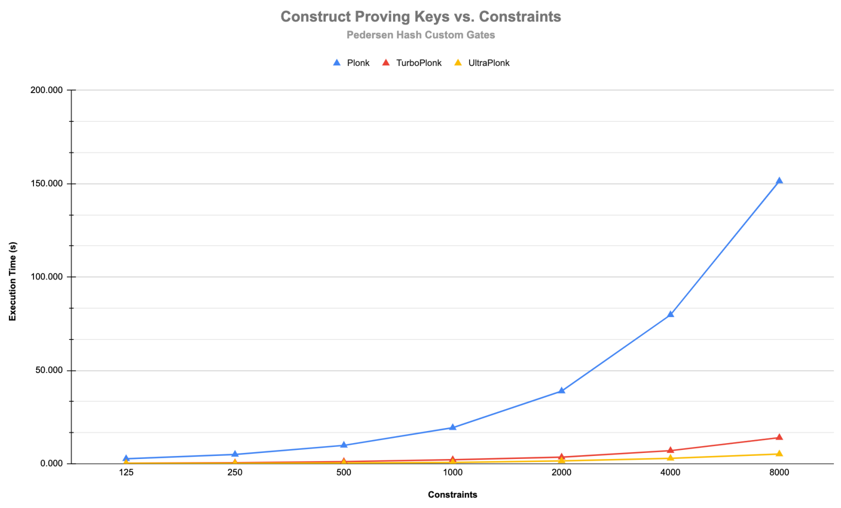
4.2.1 Comparison of Pedersen Hashes

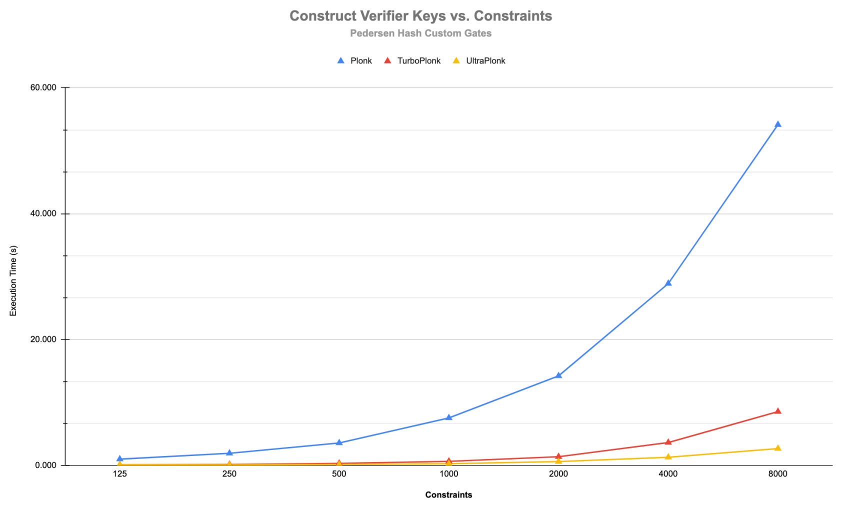
We compare the execution time and memory usage for **125 – 8k** pedersen hashes. Past 8k hashes, the performance of Plonk dramatically degrades. Without assembly instructions and compiler optimizations enabled, these workloads wouldn't be practical and the proof generation is **~4x slower**. The results highlight that execution time and memory utilization grow exponentially with respect to the number of constraints in the program.

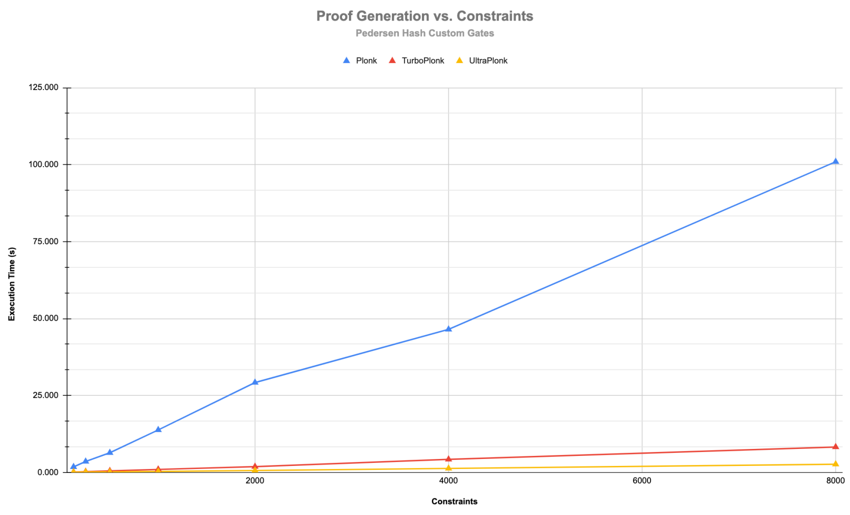
Figures 9 – 15: Prover workloads for custom gates based on pedersen hashes

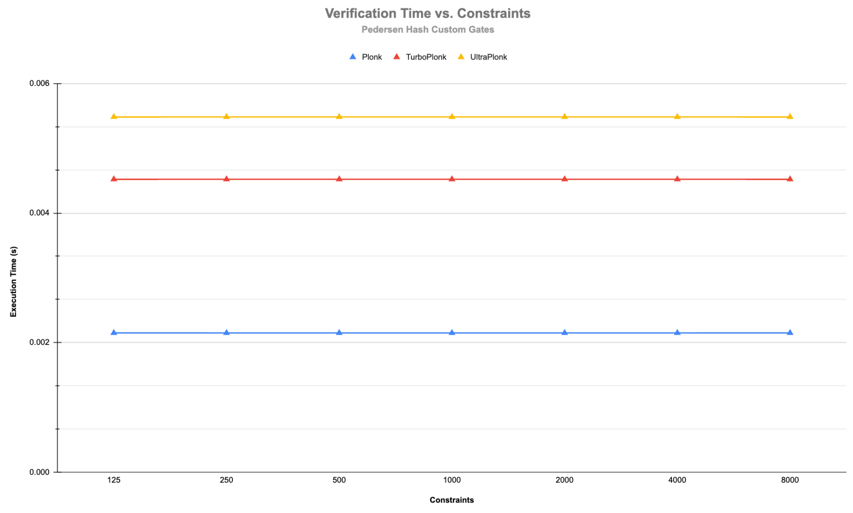


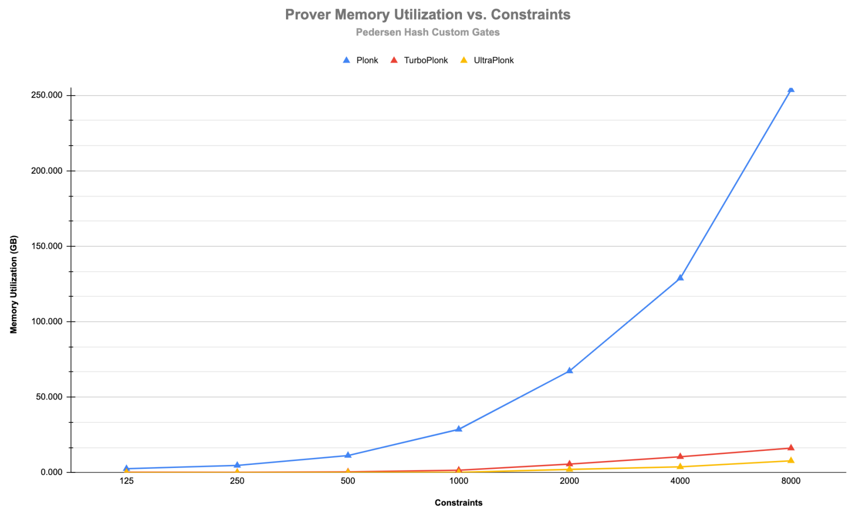



Construct Proving Keys vs. Constraints
Pedersen Hash Custom Gates
Plonk
TurboPlonk
UltraPlonk
Execution Time (s)
200.000
150.000
100.000
50.000
0.000
125
250
500
1000
2000
4000
8000
Constraints


Construct Verifier Keys vs. Constraints
Pedersen Hash Custom Gates
Plonk
TurboPlonk
UltraPlonk
Execution Time (s)
60.000
40.000
20.000
0.000
125
250
500
1000
2000
4000
8000
Constraints


Proof Generation vs. Constraints
Pedersen Hash Custom Gates
Plonk
TurboPlonk
UltraPlonk
Execution Time (s)
125.000
100.000
75.000
50.000
25.000
0.000
2000
4000
6000
8000
Constraints

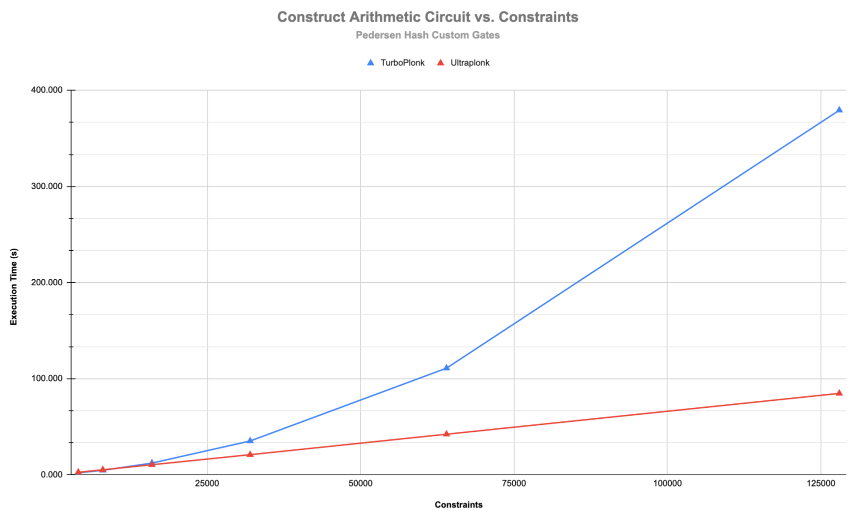


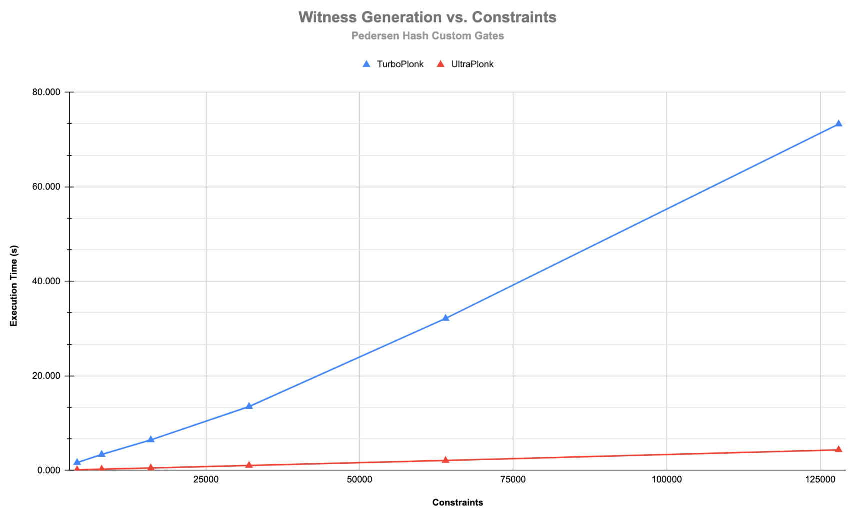


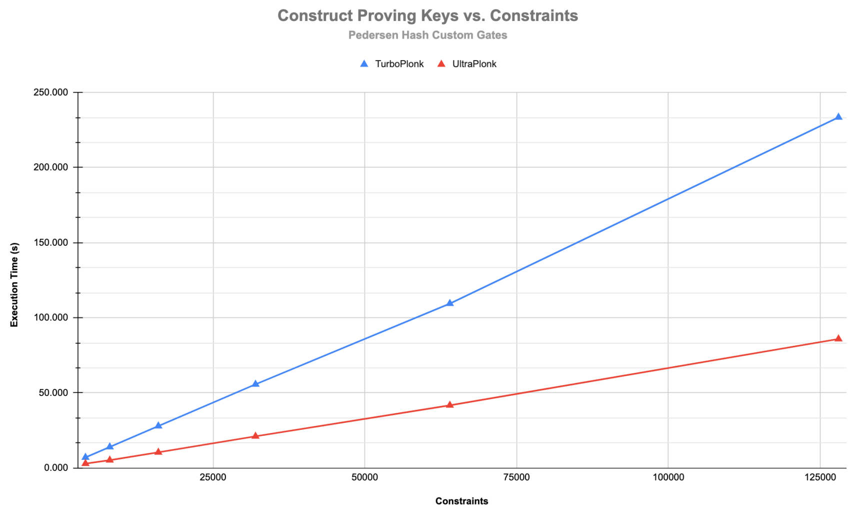
The memory consumption of Plonk and TurboPlonk is greater compared to UltraPlonk, which is contrary to our initial expectations. This is due to the fact that we are evaluating the proof systems based on the same task size, i.e. **8k** hashes, rather than the same problem size, i.e. 2^{25} constraints. Section **4.2.2** describes this in greater detail for **128k** hashes.

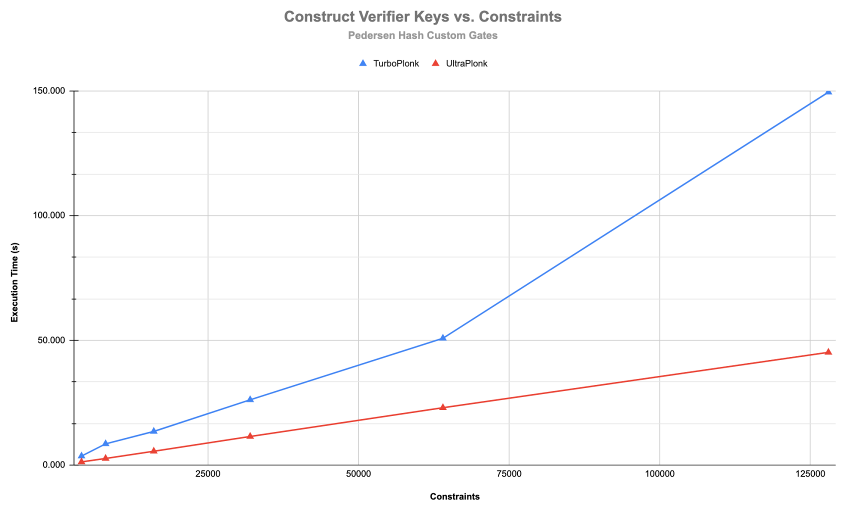
4.2.2 Comparison of Pedersen Hashes

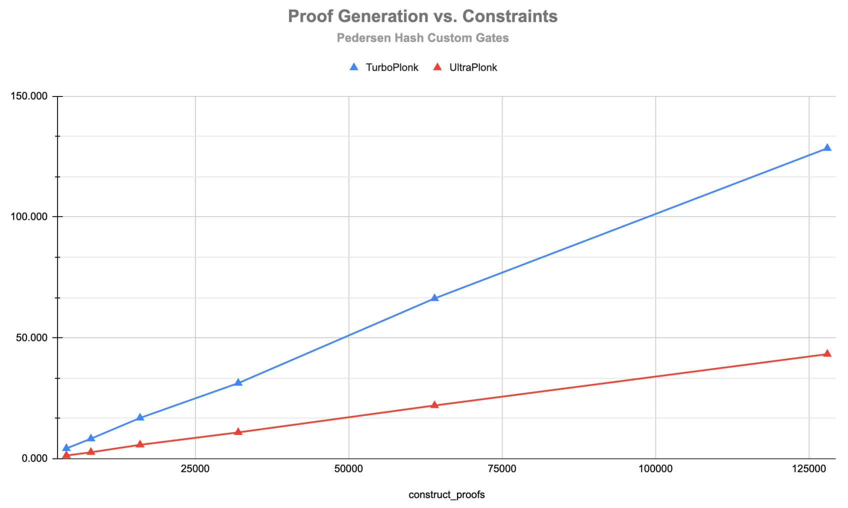
This extends section **4.2.1**, comparing TurboPlonk and UltraPlonk for **4k – 128k** pedersen hashes.

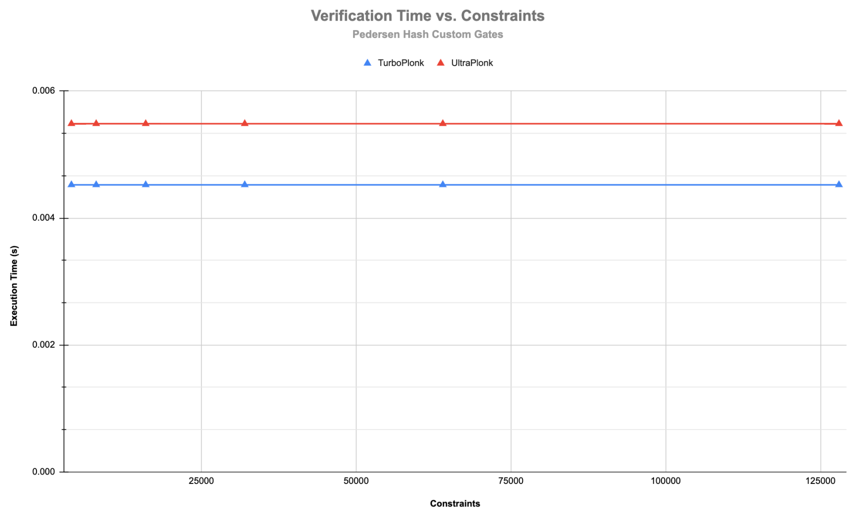

Construct Arithmetic Circuit vs. Constraints
Pedersen Hash Custom Gates
TurboPlonk
Ultraplonk
Execution Time (s)
400.000
300.000
200.000
100.000
0.000
25000
50000
75000
100000
125000
Constraints

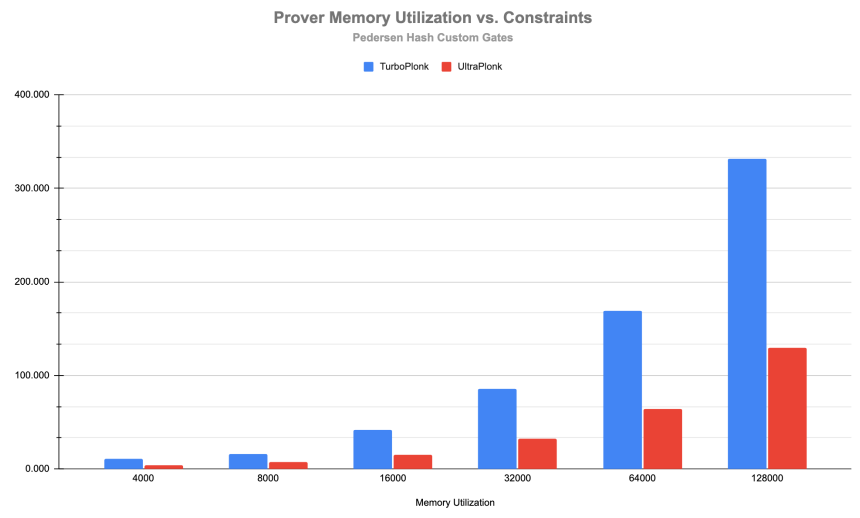

Witness Generation vs. Constraints
Pedersen Hash Custom Gates
TurboPlonk
UltraPlonk
Execution Time (s)
80.000
60.000
40.000
20.000
0.000
25000
50000
75000
100000
125000
Constraints


Construct Proving Keys vs. Constraints
Pedersen Hash Custom Gates
TurboPlonk
UltraPlonk
Execution Time (s)
250.000
200.000
150.000
100.000
50.000
0.000
25000
50000
75000
100000
125000
Constraints


Construct Verifier Keys vs. Constraints
Pedersen Hash Custom Gates
TurboPlonk
UltraPlonk
150.000
100.000
50.000
0.000
Execution Time (s)
25000
50000
75000
100000
125000
Constraints


Proof Generation vs. Constraints
Pedersen Hash Custom Gates
TurboPlonk
UltraPlonk
150.000
100.000
50.000
0.000
25000
50000
75000
100000
125000
construct_proofs


Verification Time vs. Constraints
Pedersen Hash Custom Gates
TurboPlonk
UltraPlonk
0.006
0.004
0.002
0.000
Execution Time (s)
25000
50000
75000
100000
125000
Constraints



Figures 16 – 22: Prover workloads for custom gates based on pedersen hashes

Prover times for UltraPlonk were **~3x** faster than TurboPlonk, and the memory utilization for UltraPlonk was **~2.5x** lower than TurboPlonk. This can be explained by the fact that generating a proof for **128k** hashes requires radically different circuit sizes between TurboPlonk and UltraPlonk.

→ **UltraPlonk**: 13,191,211 constraints / 103 gates per hash = **128k hashes**

→ **TurboPlonk**: 44,184,152 constraints / 345 gates per hash = **128k hashes**

TurboPlonk has **~2.5x** increased memory utilization than UltraPlonk when evaluating **128k** hashes because it exhibits a **~3.3x** increase in circuit size. TurboPlonk has more gates per hash than UltraPlonk, since UltraPlonk employs lookup tables that reduce the circuit's constraint size. 128k hashes in TurboPlonk will therefore be more expensive than 128k hashes in UltraPlonk in terms of the number of gates.

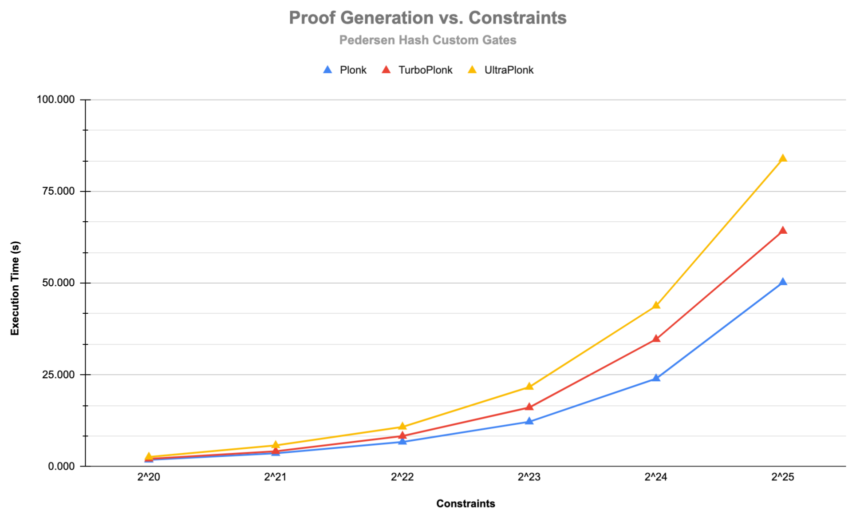
In general, custom gates increase the degree of the identity to be proven, which increases the computational work for the prover. This is in tandem with the reduction in circuit size. For example, the degree of the identity may double while the circuit size may be cut by half [8].

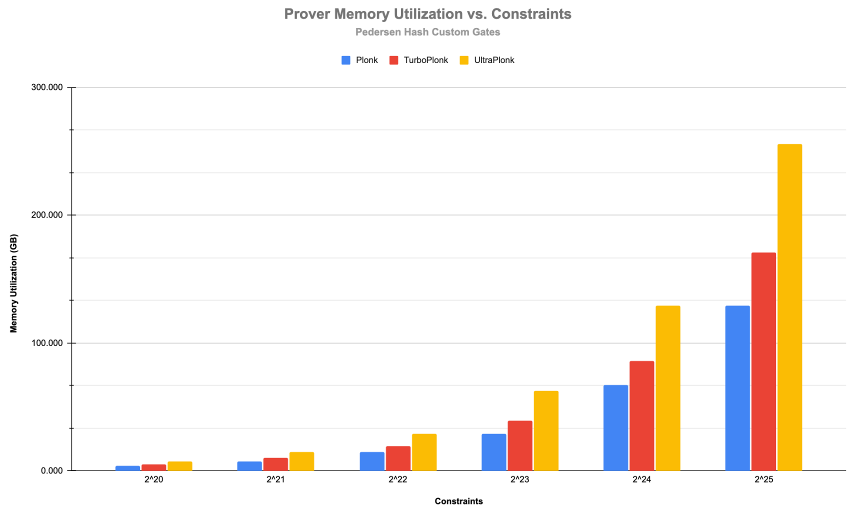
In section **4.2.3,** the memory utilization for UltraPlonk will be higher than TurboPlonk because we are evaluating the proof systems based on the same problem size, i.e. 2^{25} constraints, rather than on the same task size, i.e. **8k or 128k** hashes, as in sections **4.2.1 and 4.2.2** respectively.

4.2.3 Comparison of Circuit Size

In this section, we compare the throughput of these proving systems based on the circuit size rather than number of hashes. The following charts highlight that for 2^{25} constraints, proof generation took **50s** (Plonk), **64s** (TurboPlonk) and **84s** (UltraPlonk). The memory utilization was **129 GB** (Plonk), **171 GB** (TurboPlonk), and **256 GB** (UltraPlonk). The verification time was constant at approximately **2 – 5 ms.**

Figures 23 –24: Proof generation and memory utilization for workloads based on circuit size





In this setting, UltraPlonk has the highest memory profile because it has more selector polynomials than TurboPlonk, and each selector polynomial is stored in 3 forms. The increase in memory utilization is not due to the precomputed lookup table because the table sizes don't exceed 2^{16} constraints in the current configuration.

Exceptionally large lookup tables, used for performing efficient range proofs, will have a negative effect on the prover computation. UltraPlonk adds extra prover commitments, and a larger table size can add extra prover work. This will also increase memory utilization as a larger table needs to be stored in system memory [9]. In terms of the proof sizes, TurboPlonk requires **11 N** scalar multiplications and has a proof size of **11 G1** curve elements, while UltraPlonk requires **13 N** scalar multiplications and has a proof size of **13 G1** curve elements.

4.2.3 Comparison of Circuit Size vs. Pedersen Hashes

The following compares the circuit size as a function of the number of constraints it can process. In our configuration, Plonk requires **5113** gates per hash, TurboPlonk requires **345** gates per hash, and UltraPlonk: **103** gates per hash. For circuits with 2^{25} constraints, Plonk processed **6,562** hashes, TurboPlonk processed **97,259** hashes, and UltraPlonk processed

325,771 hashes. TurbPlonk and UltraPlonk were able to prove **14.8x** and **49.6x** more hashes than Plonk respectively for the same circuit size. UltraPlonk was able to process **~3.4x** hashes than Turboplonk.

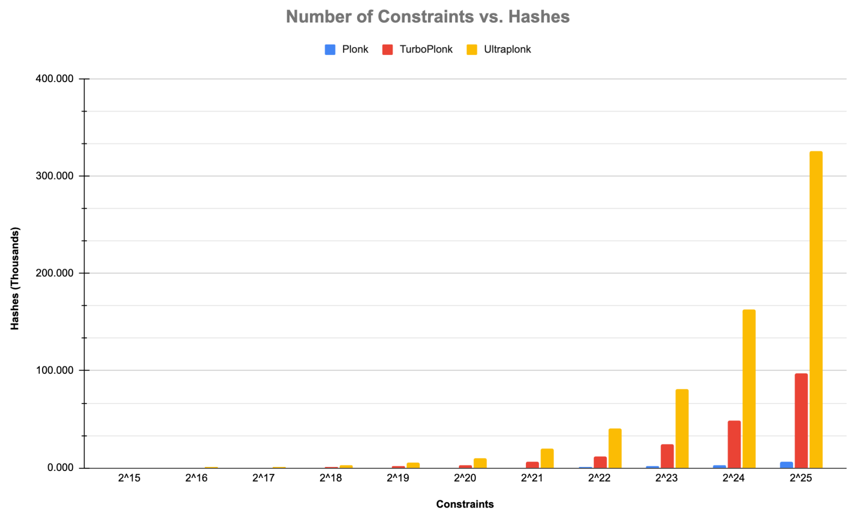


Figure 25: Comparing number of hashes and circuit size

5 Future Research

We obtained our using one machine running on a single **32-core** CPU. The next steps are executing these prover workloads on a single Nvidia server GPU using Cuda, and then multiple GPUs.

6 Acknowledgements

This work was supported in part by Oracle Cloud and related resources provided by the Oracle for Research program. The benchmarking data shown here was run on the Oracle Cloud. Other support for this work includes a gift from Steel Perlot and Google, and a Lehigh CORE grant. We thank Suyash Bagad from the Aztec network for his insights into the TurboPlonk and UltraPlonk proving systems.

7 References

[1] Benchmarking Plonk Proving System:
https://github.com/TalDerei/Masters-Research/blob/main/Benchmarking%20PlonK%20Proving%20System.pdf

[2] Barretenberg:
https://github.com/AztecProtocol/barretenberg

[3] The Turbo-Plonk Program Syntax for Specifying SNARK Programs:
https://docs.zkproof.org/pages/standards/accepted-workshop3/proposal-turbo_plonk.pdf

[4] Plookup: A Simplified Polynomial Protocol for Lookup Tables:
https://eprint.iacr.org/2020/315.pdf

[5] Why Hashes Dominate in SNARKs
https://medium.com/aztec-protocol/why-hashes-dominate-in-snarks-b20a555f074c

[6] Raw Benchmarks:
https://drive.google.com/file/d/1uXLUabgh18XHdz0FWvaZo7jMRkqfa1qz/view?usp=share_link

[7] PLONK Benchmarks II — ~5x faster than Groth16 on Pedersen Hashes:
https://medium.com/aztec-protocol/plonk-benchmarks-ii-5x-faster-than-groth16-on-pedersen-hashes-ea5285353db0

[8] Proof Compression:
https://medium.com/aztec-protocol/proof-compression-a318f478d575

[9] zkSummit: Plookup, Speeding up the PLONK Prover - Zac Williamson & Ariel Gabizon:
https://www.youtube.com/watch?v=Vdlc1CmRYRY&ab_channel=ZeroKnowledge