Performance evaluation of DIZK and Bellman for Blake2s

Decentriq

This report summarizes our findings on the performance of **DIZK** and **Bellman** for the verifiable computation of a **Blake2s** hash on inputs, ranging between 192 and 786432 bytes, or equivalently of constraint systems, ranging between 64K and 512M constraints.

Setup of the experiment

DIZK is a distributed zero-knowledge framework, running on Spark. For the DIZK runs, we setup a cluster of 4 worker nodes and 1 master node on AWS in **Spark Standalone mode**. All nodes were r4.8xlarge instances, with 32 vCPUs and 244 GiB RAM. The total capacity of the worker nodes was thus 128 vCPUs and 976 GiB RAM. Each worker node had an EBS General Purpose SSD (gp2) volume, with a 600 GB capacity to provide sufficient disk space for data shuffling and caching of RDDs.

The Bellman runs were carried out on a personal server with 6 cores (Intel Core i5-8400) and 32 GB RAM.

For each run, the following metrics were collected:

- *NumConstraints*: the number of constraints in the circuit;
- *ByteInput*: the number of bytes on which the Blake2s computation is being run;
- *NumCoresPerMachine*: the number of cores (actual or vCPU) on each machine;
- *MinMemoryPerExecutorGB*: the amount of RAM memory consumed by an executor during the computation. In the case of DIZK, this number was determined by randomly choosing one executor and monitoring it's memory usage. For both DIZK and Bellman the tool used to collect this metric was **ps_mem**;
- *NumExecutors*: the number of Spark executors used in the DIZK run. This number is not applicable for the Bellman runs;
- *NumCoresPerExecutor*: the number of cores assigned to each Spark executor. This number is not applicable for the Bellman runs;
- *RunTimeParamGenMinutes*: the running time in minutes for computing the CRS (including both the proving and verification key);
- *RunTimeProverMinutes*: the running time in minutes for the prover

- *ParameterMB*: the size of the proving and verification key;
- *HourlyCostPerMachine*: the AWS hourly cost for ar4.8xlarge instance. For the Bellman runs and estimate of the hourly cost on AWS for a machine with 6 vCPUs and 32GB RAM was done;
- *MachineCount*: number of machines used in the setup. This is the number of worker nodes for DIZK and is always one for Bellman;
- *TotalCost*: the cost for running the prover, calculated as *MachineCount* * *HourlyCostPerMachine* * *RunTimeProverMinutes* / 60

The result of the experiments are summarized **here** and are elaborated upon in the *Discussion* Section.

Implementation

We provide two implementations, one for **DIZK** and one for **Bellman**. The Bellman runs can be carried out by using the **run.sh** script available in the repository. Internally, they are leveraging the existing **Blake2s** constraint system, used in Zcash.

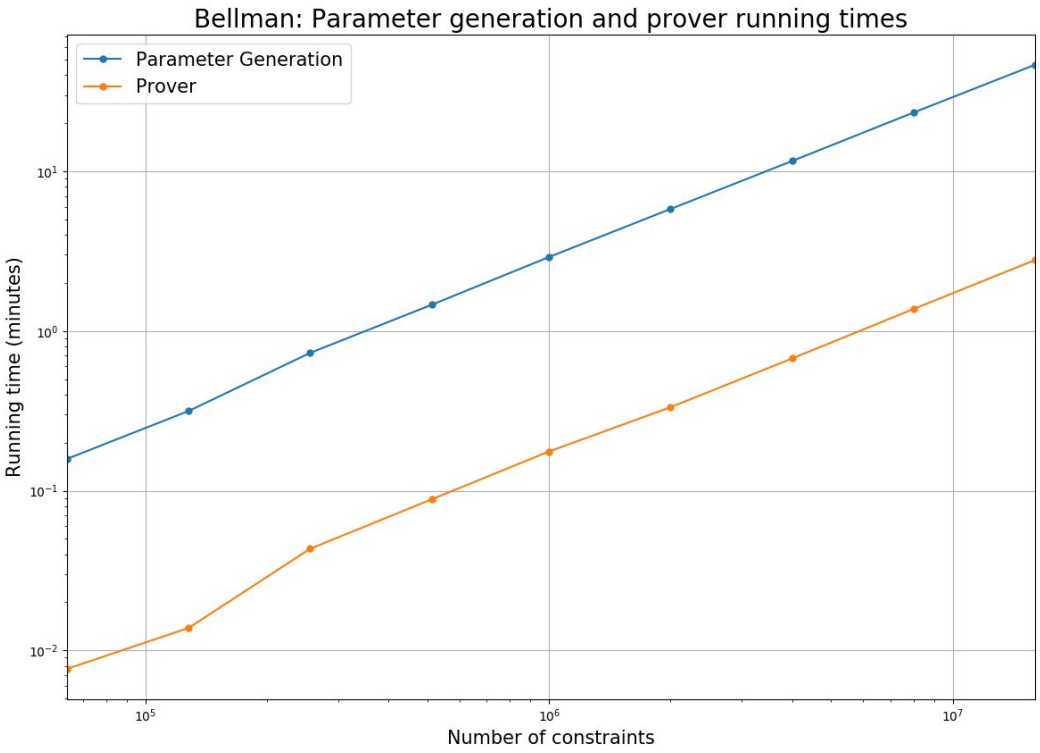
For the DIZK runs we rely on **xjsnark**, a library providing a high-level DSL for SNARKs, for producing the circuits in the Pinocchio format (a good overview of the format is available **here**). We implemented a **Pinocchio reader**, which takes a circuit in this format and constructs an equivalent DIZK R1CS. Detailed instructions about how to do an example run, are available **here**.

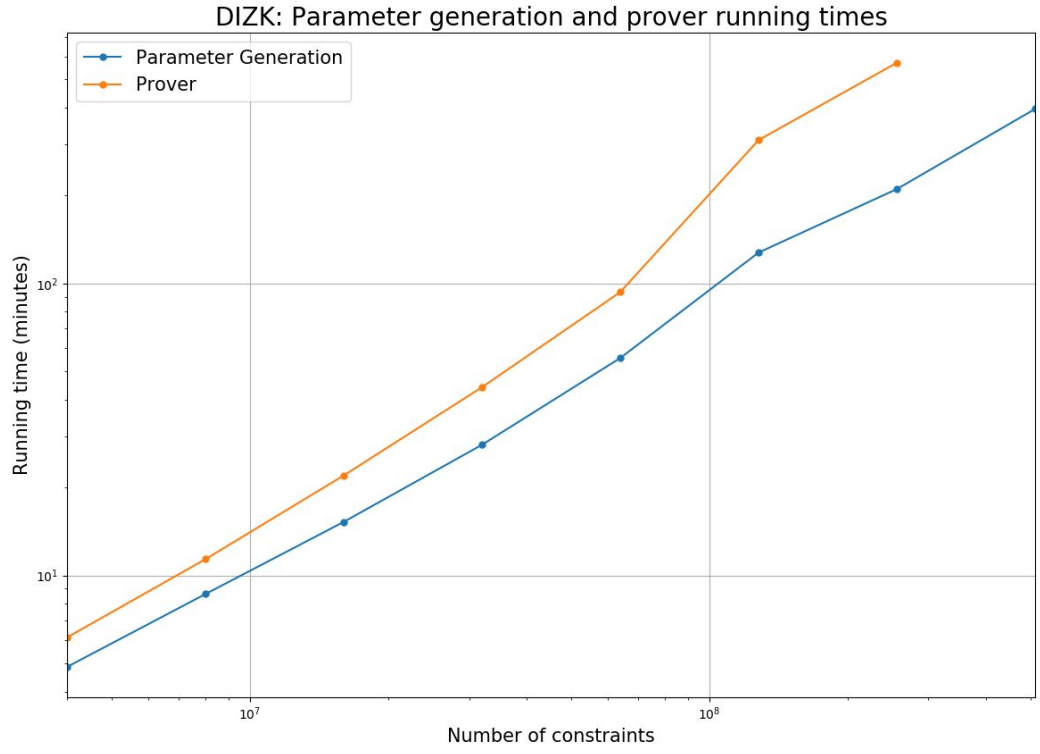
Results & Discussion

The maximum circuit size that we were able to successfully run was 16M constraints for Bellman and 256M constraints for DIZK. Runs for 32M constraints on Bellman and for 512M on DIZK failed with OOM errors during the setup and proving phase, respectively.

We had three overlapping runs between DIZK and Bellman in terms of the number of constraints: 4M, 8M and 16M. For these runs, the the total cost for running the prover in DIZK is **270-309x more expensive** than the Bellman version. Similarly, the running times for Bellman are **7.9-9.1x shorter**. We note that the multipliers are decreasing with increasing circuit size, i.e. it's 309x more expensive and 9.1x slower to run a 4M circuit and 270x more expensive and 7.9x slower to run a 16M circuit. However, given the magnitude of the differences and the (roughly) linear scaling of the running times in both DIZK and Bellman (see figures below), we believe it is fairly safe to assume that Bellman should be

the preferred choice even for considerably larger circuits and almost certainly should be used over DIZK for circuits with up to 2^{32} constraints, which is the limit for efficiently computing FFTs over the scalar field of BLS12-381. Preference should be placed on DIZK only for circuits larger than 2^{32} since the specially sampled **DIZK curve** has a scalar field, which supports efficient computation of FFTs for constraint systems of up to 2^{50} constraints.


Bellman: Parameter generation and prover running times
Parameter Generation
Prover
Running time (minutes)
10^{1}
10^{0}
10^{-1}
10^{-2}
10^{5}
10^{6}
10^{7}
Number of constraints


DIZK: Parameter generation and prover running times
Parameter Generation
Prover
Running time (minutes)
10^{2}
10^{1}
10^{7}
10^{8}
Number of constraints

Useful links

Below we provide a list of useful links for this study:

- **Bellman-related source code**
- **DIZK-related source code**
- **Summary of results**
- **Meeting notes**
- **xjsnark**
- **Pinocchio format**