On amortization techniques for FRI-based SNARKs

Albert Garreta[1], Hayk Hovhanissyan[2], Aram Jivanyan[2], Ignacio Manzur[1], Isaac Villalobos[1], and Michał Zając[1]

[1]Nethermind Research, {albert,ignacio,isaac,michal}@nethermind.io
[2]Yerevan State University, aram@skycryptor.com,hayk.9816@gmail.com

Abstract

We present two techniques to improve the computational and/or communication costs of STARK proofs: packing and modular split-and-pack.

Packing allows to generate a single proof of the satisfiability of several constraints. We achieve this by packing the evaluations of all relevant polynomials in the same Merkle leaves, and combining all DEEP FRI functions into a single randomized validity function. Our benchmarks show that packing reduces the verification time and proof size compared to individually proving the satisfiability of each witness, while only increasing the prover time moderately.

Modular split-and-pack is a proof acceleration technique where the prover divides a witness into smaller sub-witnesses. It then uses packing to prove the simultaneous satisfiability of each sub-witness. Compared to producing a proof of the original witness, splitting improves the prover time and memory usage, while increasing the verifier time and proof size. Ideas similar to modular split-and-pack seem to be used throughout the industry, but 1) generally execution traces are split by choosing the first k rows, then the next k rows, and so on; and 2) full recursion is used to prove the simultaneous satisfiability of the sub-witnesses, usually combined with a final wrapper proof (typically a Groth16 proof). We present a different way to split the witness that allows for an efficient re-writing of Plonkish-type constraints. Based on our benchmarks, we believe this approach (together with a wrapper proof) can improve upon existing splitting methods, resulting in a faster prover at essentially no cost in proof size and verification time.

Both techniques apply to popular FRI-based proof systems such as ethSTARK [Sta21], Plonky2/3 [Tea22], RISC Zero [BGT23], Boojum [zkS23], and others.

Contents

1 Introduction

STARKs are non-interactive proof systems known for their efficient prover and verifier characteristics. These systems are now crucial in decentralized environments, especially regarding blockchain scalability. Projects like StarkWare, Polygon, and RISC Zero use STARKs to scale Ethereum. At the same time, these companies have created frameworks that can be used to generate computational integrity proofs with multiple applications.

However, because these proof systems use Merkle tree commitments, they have not benefited from the exciting breakthrough of folding schemes [KST21], which require homomorphic commitments. Instead, STARKs have continued to rely on their sublinear verifier costs to implement recursion. This recursive proof generation method serves both as an IVC/PCD construction in the STARK setting, and as the preferred method for batch-proof generation. Batch-proof generation is now becoming crucial in scaling L2 solutions. For instance SHARP ("SHARed Prover") is a fundamental part of the Starkware tech stack that allows for the batch-generation and verification of multiple Cairo programs. This increases transaction throughput and reduces both latency and gas fees on Starknet, Starkware's decentralized zk-rollup.

Another use case of recursion comes from a technique that seems to be used throughout the industry: splitting long execution traces into smaller traces. When doing such a "splitting", one needs to carefully express constraints that span across trace chunks, so as to enforce the same exact constraints as before the splitting. Then, a proof is generated for each chunk, and the prover produces a recursive proof of the correctness of all the generated proofs. As we mentioned, this seems to be a widespread technique, and a concrete example of this technique being used by RISC Zero can be found in a zkSummit talk [Gaf]. As far as we are aware, this technique has not been formally analyzed yet, and further, it has not been fully exploited. Indeed, the common way teams are splitting execution traces is by selecting the first k rows, then the next k rows, and so on. We call this approach *sequential splitting*. In this work, we present a more efficient variation of this idea.

Overall, this outlines two interesting research directions for optimizing STARK systems: improving batch/recursive proof generation, and finding ways to improve the split-and-prove method.

1.1 Our contribution

We present two ideas to improve STARK proof generation and/or verification: *packing* and *modular split-and-pack*. We will describe the two techniques in the context of the *Plonkish* relation and what we call the *STARKish* IOP.

In this work, by *Plonkish relation* we refer to a relation which consists of an AIR instance [Sta21] with copy constraints [GWC19]. This relation is essentially the relation found in Plonky2/3 [Tea22] (see also [BGK$^+$23]). By *STARKish IOP* we refer to the Plonky2-based

IOP as described in [BGK$^+$23]. This is essentially the ethSTARK protocol [Sta21] with added support for permuatiton arguments. The two techniques described in this paper still apply in the simpler setting, where constraints are just AIRs, and the underlying IOP is the ethSTARK IOP. Similarly, the techniques still apply when we add lookups to the Plonkish relation and support for lookup arguments in the STARKish IOP.

At a high level, the two techniques utilize the fact that recent STARKs are constructed from a "δ-correlated IOP" [BGK$^+$23]. These are roughly IOPs in which the verifier's final decision procedure is as follows: 1) the verifier checks that a polynomial equation holds at a single point (DEEP QUERY procedure), and 2) the verifier checks that certain functions are polynomials of low degree by using FRI (DEEP FRI procedure).

1.1.1 Packing

Packing is a batch-proof generation method. With packing, a prover can create a proof for the simultaneous satisfiability of witnesses to different Plonkish (or AIR, or RAP, and so on) instances. These witnesses can be thought of as execution traces (matrices of dimension $\mathsf{n} \times \mathsf{r}$) representing the application of a certain program during n steps to r registers. The correctness of the trace representing a certain program is enforced by using polynomial constraints between its entries. As mentioned, one can reduce the integrity of a trace to two checks: that a certain polynomial equation holds at a random point, and that certain functions are low-degree polynomials. The second check is the most expensive of the two. For our purposes, the batched-FRI proximity test is used to carry out this check, which guarantees that a list of functions $f_1, \ldots, f_n : D \to \mathbb{F}$ are (close to) low degree polynomials[1]. Here D is certain subset of a finite field $\mathbb{F}$.

Batched-FRI simply works by applying the FRI low-degree test to a random linear combination of the maps $f_1, \ldots, f_n$. Hence if we need to test more functions we can simply linearly combine them with the functions that we already combined. This allows us to batch all functions to be tested for the low-degree property across different execution traces (representing different programs), and apply a single batched-FRI test. Our second observation was already presented in [MAGABMMT23]. During the batched-FRI protocol, we often have to reveal the evaluations of functions at the same point, and therefore when using Merkle tree commitments we may batch all the evaluations at the same point in the same Merkle leaf. We do this across different execution traces, while [MAGABMMT23] was limited to the same execution trace. These two improvements theoretically reduce the Prover work and the proof size, because the Prover performs essentially one FRI protocol and computes less Merkle commitment roots. The Verifier has a bit more work to do to hash Merkle leaves, but performs much less hash operations.

As we discuss later, we believe packing to be a technique that, in combination with recursion, can improve the cost of proving (multiple) large and complex statements. Further, packing is a used as a subroutine in the second technique we present in this paper, and which we call modular-split-and-pack.

Asymptotic and concrete improvements of packing. The theoretical improvement brought by packing is shown in Table 1.

The improvements concern mainly the communication complexity and the Verifier complexity. More precisely, for the proof size we see that the Prover sends less Merkle commitments to witness functions, as evaluations of witness functions are packed in the same leaves across instances. Additionally, the Prover sends less Merkle commitments and decommitment information during the FRI protocol, as it performs only one FRI protocol. The Prover sends exactly the same amount of evaluations when packing proofs. On the Verifier side, the improvement is contained in the hash operations it performs. When packing proofs, the

[1]In fact, batched-FRI shows the stronger property that the maps f_i have correlated agreement in a Reed-Solomon code

	Sequential, N instances	Packed, N instances
Proof size	$(4 + \sum_{i\in[N]} \log(\mathsf{n}_i))N \cdot \mu$ bits $+ \sum_{i\in[N]} (4\mathsf{r}_i + \nu_i + s_i + \ell_i + 1) \cdot \log(\mathbb{K})$ bits $\boldsymbol{+O\left(s_{FRI} \sum_{i\in[N]} \log(\rho^{-1}\mathsf{n}_i) \log(\mathsf{n}_i)\right) \cdot \mu}$ **bits**	$(4 + \log(\mathsf{n})) \cdot \mu$ bits $+ \sum_{i\in[N]} (4\mathsf{r}_i + \nu_i + s_i + \ell_i + 1) \cdot \log(\mathbb{K})$ bits $\boldsymbol{+O\left(s_{FRI} \log(\rho^{-1}\mathsf{n}) \log(\mathsf{n})\right) \cdot \mu}$ **bits**
Verifier work	$O\left(\sum_{i\in[N]} \left(s_i + \log(\mathsf{n}_i) + \sum_{k\in\lvert\mathcal{P}_i\rvert} \mathsf{fops}(P_{i,k})\right)\right) \mathbb{K}$ $+2s_{FRI}\mathbb{H}$ of size $\mathsf{r}_1 \log(\mathbb{K})$ $+2s_{FRI}\mathbb{H}$ of size $s_1 \log(\mathbb{K})$ $+2s_{FRI}\mathbb{H}$ of size $\nu_1 \log(\mathbb{K})$ $\vdots$ $+2s_{FRI}\mathbb{H}$ of size $\mathsf{r}_N \log(\mathbb{K})$ $+2s_{FRI}\mathbb{H}$ of size $s_N \log(\mathbb{K})$ $+2s_{FRI}\mathbb{H}$ of size $\nu_N \log(\mathbb{K})$ $\boldsymbol{+O(s_{FRI} \sum_{i\in[N]} \log(\rho^{-1}\mathsf{n}_i) \log(\mathsf{n}_i))}\mathbb{H}$	$O\left(\sum_{i\in[N]} \left(s_i + \log(\mathsf{n}_i) + \sum_{k\in\lvert\mathcal{P}_i\rvert} \mathsf{fops}(P_{i,k})\right)\right) \mathbb{K}$ $+2s_{FRI}\mathbb{H}$ of size $\sum_{i\in[N]} \mathsf{r}_i \log(\mathbb{K})$ $+2s_{FRI}\mathbb{H}$ of size $\sum_{i\in[N]} s_i \log(\mathbb{K})$ $+2s_{FRI}\mathbb{H}$ of size $\sum_{i\in[N]} \nu_i \log(\mathbb{K})$ $\boldsymbol{+O(s_{FRI} \log(\rho^{-1}\mathsf{n}) \log(\mathsf{n}))}\mathbb{H}$

Table 1: Impact of packing N proofs of the satisfiability of Plonkish instances $\mathsf{PL}_i = (\mathcal{P}_i, \mathcal{Q}_i, \sigma_i, \mathsf{PI}_i, \mathsf{r}_i, \mathsf{r}'_i, \ell_i, \mathsf{n}_i, \mathbb{F}, \mathbb{K})$. Here, n_i and r_i is the number of rows and columns in the execution trace, respectively; σ_i is the permutation specifying the permutation constraints; r'_i is the number of columns that are subject to permutation constraints; $\mathcal{P}_i$ are the constraint polynomials; PI are the public inputs; $\mathcal{Q}_i$ are "selector vectors" and $\ell_i = |\mathcal{Q}_i|$.
On the left, we specify the proof size and verifier work associated to applying the STARKish IOP to each witness separately, and on the right we apply the STARKPack IOP. We set $\mathsf{n} = \max_{i\in[N]}(\mathsf{n}_i)$, and recall that ρ is the rate used in the IOP (we may assume it is constant). The hash function is assumed to be a function $\{0,1\}^* \to \{0,1\}^\mu$. In the table, for the verifier work, $\mathbb{K}$ represents the cost of a field operation over $\mathbb{K}$, and $\mathbb{H}$ represents the cost of a hash function application. Whenever we write "of size" for a hash computation, this means that the hash function is applied to a bitstring of the corresponding size (e.g., $\mathsf{r}_1 \log(\mathbb{K})$). If no size is mentioned, it is assumed that the hash is applied to a bitstring of length μ. The quantity s_{FRI} is the number of times the QUERY phase of FRI is repeated. The FRI COMMIT phase is assumed to be carried out until polynomials are supposedly constant. The quantities s_i, ν_i are the quantities appearing in the STARKish IOP (packed and not packed), see for example Section 3.1. The quantity $\mathsf{fops}(P_{i,k})$ represents the maximum number of $\mathbb{K}$ field operations necessary to evaluate the $2\mathsf{r}_i + \ell_i$-variate polynomial $P_{i,k}$ at any point in $\mathsf{H}_i^{2\mathsf{r}_i+\ell_i}$ (H_i is the evaluation domain).

Verifier hashes bigger leaves for witness functions Merkle trees, but there are less of these. Finally when packing proofs, the Verifier only checks Merkle decommitment information for one application of FRI. In the case of proving multiple instances of the same program, more improvements are possible, e.g., we can make multiple proof instances share common parameters such as public input parameter columns, fixed selector columns, or lookup tables.

Our benchmarks can be found in Section 3.3 and Appendices A.1, A.2 and A.3. We use the Winterfell and RISC Zero libraries, and focus on proving execution traces of simple programs. Precisely, for the Winterfell library, the program checks that $x_{i+1} = x_i^3 + 42 \mod (p)$ for some prime p, for $i \in [0, 2^{16})$. The execution trace has 2^{16} rows. For the RISC Zero library, the program proves the knowledge of the prime factors of a number, yielding, again a trace of 2^{16} rows. The number of columns in RISC Zero is fixed to 275 (it cannot be customized because it is determined by the r0vm circuit[2]). We tested different column sizes in Winterfell.

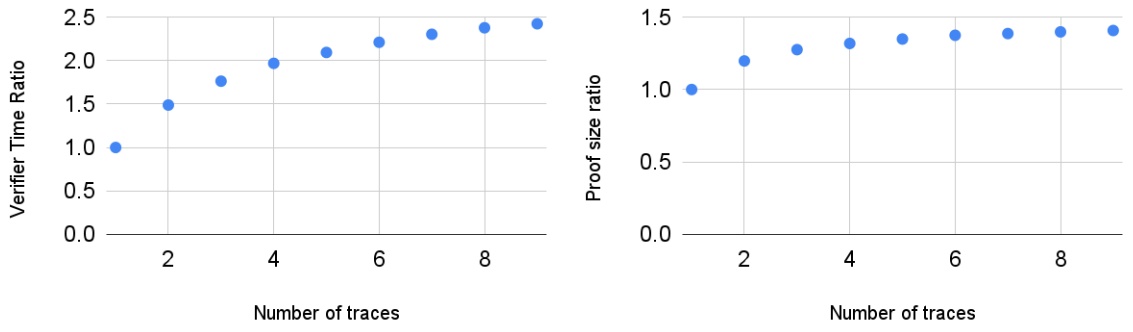
For 9 traces of 275 columns, the packed RISC Zero verifier is around 2.2 times faster than the unpacked RISC Zero verifier. Further, the packed Winterfell verifier is 2.4 times faster than the unpacked Winterfell verifier. Both the unpacked RISC Zero and Winterfell verifier checked each proof for the 9 traces sequentially.

The packed RISC Zero proof is around 2.5 times smaller than the proof produced by the unpacked RISC Zero scheme, and the packed Winterfell proof is 1.4 times smaller than the proof created by the unpacked Winterfell scheme[3]. We also note that as the number of traces packed together increases, the improvements in both proof size, shown in Appendix A.3, and verifier time, shown in Appendix A.2, become more significant as can be seen in Fig. 1 and Fig. 2.

In all our Winterfell benchmarks, the prover time was improved slightly, but remained

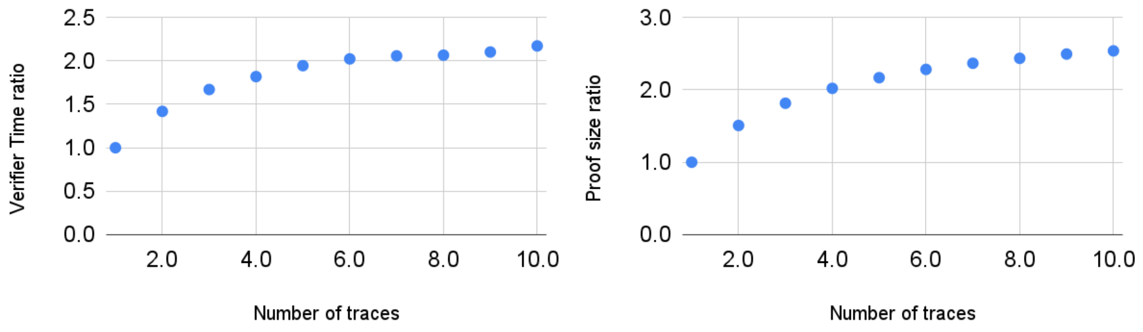
[2]https://docs.rs/risc0-circuit-rv32im-sys/latest/risc0_circuit_rv32im_sys/

[3]Both the RISC Zero and Winterfell proofs are the proofs for the 9 traces generated one after the other.



(a) Verification Time (b) Proof Sizes

Figure 1: Improvements for Winterfell with 275 columns



(a) Verification Time (b) Proof Sizes

Figure 2: Improvements for Risc Zero

comparable to the prover that individually proves the satisfiability of all witnesses. This was especially true when the number of columns got larger (for 275 columns, the improvement in prover time was around 5%). For our RISC Zero benchmark, the prover time was slightly worse as can be seen in Appendix A.1.

1.1.2 Modular Split-and-pack

Informally, trace splitting is a proof acceleration method where the prover informally splits the execution trace of size $\mathsf{n} \times \mathsf{r}$ into chunks of size $\mathsf{n}/k \times \mathsf{r}$ and then applies the original constraints to the entries in the split witnesses. Since the chunks have less rows, we can interpolate their columns with smaller degree polynomials over smaller domains. The constraints for the entries in the chunks need to be translated into polynomial equations involving the polynomials interpolating the new, smaller columns. The new constraints must stay consistent with the previous constraints. The way to do this translation will depend on the splitting, and in general it is hard to do efficiently.

As far as we understand, ideas similar to splitting are used throughout the industry. For instance, from [Gaf] we see that companies like RISC Zero split long execution traces into sequential chunks (with constraints that may overlap from one chunk to the other). This means that if the execution trace is represented as a matrix of size $\mathsf{n} \times \mathsf{r}$, they split it by choosing the first k rows, then the next k rows, and so on. Recursion is usually used to prove the simultaneous satisfiability of each chunk. Splittings other than sequential do not seem to be explored or considered. We present a new type of splitting modulo k, where rows are selected according to the residue of their index modulo k, where k divides n. We show that for Plonkish-type constraints that involve consecutive rows, our modular-split-and-pack technique allows to write k constraints for each original constraint (which is optimal). For typical constraints, a sequential splitting requires more, essentially because constraints can

Proving Action	**% of Overall Proof Time**
Execution trace low-degree extension (LDE) polynomial computation and commitment	$45\% - 50\%$
Constraint evaluation over LDE domain	$25\% - 30\%$
Constraint polynomial factoring into lower-degree polynomial components	$6\% - 10\%$
Constraint polynomial evaluation and commitment	$\approx 5\%$
DEEP composition polynomial construction and evaluation	$10\% - 12\%$
FRI folding and query proofs	$\approx 1\%$

(a) Modulus Labs benchmarks on Winterfell's prover time repartition

Proving Action	**% of Overall Proof Time**
Computing and committing to the wire polynomials, i.e. the intermediate computed values	$60\% - 70\%$
Computing and committing to permutation polynomials and permutation argument product polynomials	$5\% - 8\%$
Computing and committing to the vanishing/quotient polynomial	$6\% - 10\%$
Computing opening proofs for the verifier's challenge point (Plonky2 uses a multi-opening version of FRI)	10%

(b) Modulus Labs benchmarks on Plonky2's prover time repartition

Figure 3: Modulus Labs benchmarks on Prover time spread for the Winterfell (a) and Plonky2 (b) libraries.

involve the last row from a chunk and the first from the next one.

It is not immediately clear that splitting traces in other ways could lead to improvements in the number and degree of the resulting constraints. It just so happens that if constraints have some structure, then there are splittings which are more efficient in the rewriting of the constraints over the smaller chunks.

For instance, for Plonkish-type constraints (see Definition 2.1), we show that a splitting "modulo k" leads to better performance than the sequential splitting. Splitting modulo k splits rows in the execution trace according to the residue of their index modulo k. We show in Example 4.4 that this allows the Plonkish-type constraints to be efficiently translated into polynomial constraints. Once this splitting and constraint translation is done, the Prover can use a technique similar to packing to prove the simultaneous satisfiability of each chunk. This means that the evaluations of the witness polynomials at the same point in the evaluation domain are batched in the same Merkle leaf. It also means that all functions involved in all batched-FRI test are linearly combined into a single validity function.

We call the combination of the modular splitting with the packing technique *modular split-and-pack*.

Asymptotic and concrete improvements of modular split-and-pack. In Table 2, we see the impact of modular split-and-pack for a Plonkish instance. Precisely, the splitting is done modulo k. The operation is used on a Plonkish instance with n rows.

We see that modular split-and-pack reduces the cost of number-theoretic transforms from $O(\mathsf{n}\log(\mathsf{n}))$ to $O(\mathsf{n}\log(\mathsf{n}/k))$. The leaves of the Merkle trees corresponding to the witness functions are around k times larger, but the Prover performs $\sim k$ times less hash operations. Note the leaves of all other Merkle trees do not grow in size.

To understand the significance of these cost improvements, it is important to understand the different profile costs of creating STARKish proofs. For this, we follow the benchmarks (c.f. Fig. 3) run by the Modulus Labs team [Lab23]. We see that the NTTs and Merkle tree commitments to witness functions represent a significant part of the total computation. For the Winterfell library [KGL+20], these operations represent around 50% of the overall proving time. For Plonky2 [Tea22], they comprise around 70% of the proving cost. Therefore,

	No splitting	Split mod k
Prover work	$O((\mathsf{r} + \lceil \frac{\mathsf{r}}{u} \rceil)\mathsf{n}\log(\mathsf{n}))\mathbb{F}$ $\boldsymbol{+O((\mathsf{r} + \frac{\mathsf{r}}{u} + 1)\rho^{-1}\mathsf{n}\log(\rho^{-1}\mathsf{n}) + M'\rho^{-1}\mathsf{n})}\mathbb{K}$ $+O(\rho^{-1}\mathsf{n}\sum_{j\in[\lvert\mathcal{P}\rvert]}\mathsf{fops}(P_j))\mathbb{K}$ $+(\rho^{-1}\mathsf{n})\mathbb{H}$ of size $\mathsf{r}\cdot\log(\mathbb{K})$ $+(\rho^{-1}\mathsf{n})\mathbb{H}$ of size $(\lceil \mathsf{r}/u \rceil + 1)\cdot\log(\mathbb{K})$ $+(\rho^{-1}\mathsf{n})\mathbb{H}$ of size $\nu\cdot\log(\mathbb{K})$ $+O(\rho^{-1}\mathsf{n})\mathbb{H}$	$O((\mathsf{r} + \lceil \frac{\mathsf{r}}{u} \rceil)\mathsf{n}\log(\mathsf{n}/k))\mathbb{F}$ $\boldsymbol{+O((\mathsf{r} + \frac{\mathsf{r}}{u} + 1)\rho^{-1}\mathsf{n}\log(\rho^{-1}\mathsf{n}/k) + M\rho^{-1}\mathsf{n}/k)}\mathbb{K}$ $+O(\rho^{-1}\mathsf{n}\sum_{j\in[\lvert\mathcal{P}\rvert]}\mathsf{fops}(P_j))\mathbb{K}$ $+(\rho^{-1}\mathsf{n}/k)\mathbb{H}$ of size $k\mathsf{r}\cdot\log(\mathbb{K})$ $+(\rho^{-1}\mathsf{n}/k)\mathbb{H}$ of size $(\lceil k\mathsf{r}/u \rceil + 1)\cdot\log(\mathbb{K})$ $+(\rho^{-1}\mathsf{n}/k)\mathbb{H}$ of size $\nu\cdot\log(\mathbb{K})$ $+O(\rho^{-1}\mathsf{n}/k)\mathbb{H}$
Proof size	$(4 + \log(\mathsf{n}))\mu$ bits $+M'\cdot\log(\mathbb{K})$ bits $\boldsymbol{+O(\log(\rho^{-1}\mathsf{n})\log(\mathsf{n}))\mu}$ **bits**	$(4 + \log(\mathsf{n}/k))\mu$ bits $+M\cdot\log(\mathbb{K})$ bits $\boldsymbol{+O(\log(\rho^{-1}\mathsf{n}/k)\log(\mathsf{n}/k))\mu}$ **bits**
Verifier work	$O(s + \log(\mathsf{n}) + \sum_{j\in[\lvert\mathcal{P}\rvert]}\mathsf{fops}(P_j))\mathbb{K}$ $+2s_{FRI}\mathbb{H}$ of size $\mathsf{r}\cdot\log(\mathbb{K})$ $+2s_{FRI}\mathbb{H}$ of size $(\lceil \mathsf{r}/u \rceil + 1)\cdot\log(\mathbb{K})$ $+2s_{FRI}\mathbb{H}$ of size $\nu\cdot\log(\mathbb{K})$ $\boldsymbol{+O(\log(\rho^{-1}\mathsf{n})\log(\mathsf{n}))}\mathbb{H}$	$O(s + \log(\mathsf{n}/k) + k\cdot\sum_{j\in[\lvert\mathcal{P}\rvert]}\mathsf{fops}(P_j))\mathbb{K}$ $+2s_{FRI}\mathbb{H}$ of size $k\mathsf{r}\cdot\log(\mathbb{K})$ $+2s_{FRI}\mathbb{H}$ of size $(\lceil k\mathsf{r}/u \rceil + 1)\cdot\log(\mathbb{K})$ $+2s_{FRI}\mathbb{H}$ of size $\nu\cdot\log(\mathbb{K})$ $\boldsymbol{+O(\log(\rho^{-1}\mathsf{n}/k)\log(\mathsf{n}/k))}\mathbb{H}$

Table 2: Impact of modular split-and-pack for a Plonkish instance $\mathsf{PL} = (\mathcal{P}, \mathcal{Q}, \sigma, \mathsf{PI}, \mathsf{r}, \mathsf{r}', \ell, \mathsf{n}, \mathbb{F}, \mathbb{K})$ "modulo k" (c.f. Example 4.4) into a split Plonkish instance, when proving the satisfiability of witnesses to each instance Recall that ρ is the rate used in the IOP (we may assume it is constant). The hash function is assumed to be a function $\{0,1\}^* \to \{0,1\}^\mu$. In the table, for the prover and verifier work, $\mathbb{K}$ represents the cost of a field operation over $\mathbb{K}$ (similarly for $\mathbb{F}$), and $\mathbb{H}$ represents the cost of a hash function application. Whenever we write "of size" for a hash computation, this means that the hash function is applied to a bitstring of the corresponding size (e.g., $\mathsf{r}\log(\mathbb{K})$). If no size is mentioned, it is assumed that the hash is applied to a bitstring of length μ. The quantity s_{FRI} is the number of times the QUERY phase of FRI is repeated, we assume it is a small constant and remove it from asymptotic estimates for simplicity. The FRI COMMIT phase is assumed to be carried out until polynomials are supposedly constant. The quantities s, ν are the quantities appearing in the STARKish IOP (split and not split), see for example Section 4. The quantity M is equal to $(3k+1)\mathsf{r} + \lceil k\mathsf{r}/u \rceil + kl + \nu + 1$, and M' is equal to $4\mathsf{r} + \lceil \mathsf{r}/u \rceil + \nu + \ell + 1$. The quantity $\mathsf{fops}(P_j)$ represents the maximum number of $\mathbb{K}$ field operations necessary to evaluate the $2\mathsf{r} + \ell$-variate polynomial P_j at any point in $\mathsf{H}^{2\mathsf{r}+\ell}$ (H is the evaluation domain).

reducing the amount of prover computation in those operations is crucial to have an impact on the overall proving time. This is adressed by the modular-split-and-pack technique.

However, this should be nuanced by saying that the prover needs to send around k times more evaluations in the proof, and the verifier performs k times more field operations to compute the FRI validity funtions. This dependency on k means that one should generally split into a small number of chunks (see the benchmarks below), and use a wrapper proof (with, say, Groth16 [Gro16]) to decrease the proof size and verification time. We believe that modular-split-and pack could improve upon current methods for proving (multiple) long execution traces. We believe it could lead to a technique that is faster for the prover, while having a marginal impact on the proof size and verifier work.

Our benchmarks (cf. Section 4.3) compare modular split-and-pack to running a STARK on the whole execution trace without splitting. They were run with the Winterfell library [KGL+20] (which uses ethSTARK [Sta21]) for an execution trace representing the same simple program we used for the packing benchmarks. In particular, there were no copy constraints or lookups. For this program, and a trace of a typical dimensions ($2^{16} \times 275$) the reduction in the Prover time was around ***20%*** when splitting modulo 4 (into 4 chunks of dimension $2^{14} \times 275$). The Verifier was around 2.2 times slower, and the proof around 3 times larger. When splitting modulo 32, the prover time is reduced by around ***40%***, but the proof is $\sim$ 11 times larger, and the verifier $\sim$ 6 times slower as can be seen in Fig. 15 and Fig. 16. The prover's performance exhibits an exponential improvement when the number of splits is relatively low. However, as the number of splits increases, the prover's improvement quickly reaches an asymptotic limit. In contrast, the verifier time and proof size demonstrate rapid improvements for a small number of splits, followed by a linear growth as the number of splits continues to increase. This behavior suggests the existence of an optimal "sweet spot" in the lower range of the number of splits, where the trade-off between the prover's performance and the verifier time and proof size is most favorable.

1.2 Related work

Over the past few years there has been a rapid development of folding techniques [KST21] [KS22][KS23][ZGGX23][BC23][EG23]. These methods build upon the idea of post-processing for some predicates in SNARKs, which first appeared in [BGH19] and was formalized in [BCMS20]. Most rely on additively homomorphic commitments, which STARKs do not use (there is some recent work that explores IVC and PCD in the context of non-homomorphic vector commitments [BMNW24]). The packing technique can be seen as an attempt to transpose the idea of post-processing to the STARK setting, by batching FRI instances into a single execution. Of course, the resulting technique is weaker, as the witness size increases linearly in the number of packed instances. Furthermore, packing does not lead to primitives like IVC (and PCD).

The idea of batching relevant evaluations of witness functions at the same point in the same Merkle leaf was already present in [MAGABMMT23]. That work did not leverage the full power of the batched FRI protocol to extend this Merkle commitment optimization to multiple instances.

1.3 Structure of the paper

Section 2 will introduce all the necessary preliminary notions, such as the description of the Plonkish arithmetization and permutation arguments. In Section 3, we introduce the packing technique and the STARKPack IOP, and formally prove the security of the resulting proof system. In Section 4, we introduce the splitting technique, and show how to construct an IOP for a particular type of splitting "modulo k". The performance of both methods is discussed in Sections 3 and 4, and detailed benchmarks are presented in Appendix A.

2 Preliminaries

2.1 Plonkish arithmetization

We first recall the definition of Plonkish arithmetization. We then see that witnesses for the simultaneous satisfiability of Plonkish instances are equivalent to a witness for a *packed* Plonkish relation, which we describe. We closely follow [BGK$^+$23][4]. For a vector $x \in \mathbb{F}^n$, we denote by x_i its i-th entry. We can also write $x[i]$ for the i-th entry, typically if the vector already has an index in its notation.

Definition 2.1 (Plonkish relation)**.** A Plonkish instance $\mathsf{PL} = (\mathcal{P}, \mathcal{Q}, \sigma, \mathsf{PI}, \mathsf{r}, \mathsf{r}', \ell, \mathsf{n}, \mathbb{F}, \mathbb{K})$ consists of:

- A list $\mathcal{P}$ of $(2\mathsf{r} + \ell)$-variate polynomials $P_i(X_1, \dots, X_\mathsf{r}, Y_1, \dots, Y_\mathsf{r}, S_1, \dots, S_\ell)$ for all $i \in [|\mathcal{P}|]$.
- A list $\mathcal{Q}$ of ℓ vectors $(\mathsf{q}_1, \dots, \mathsf{q}_\ell) \in (\mathbb{F}^\mathsf{n})^\ell$. These are called selector vectors.
- $\mathsf{r}' \leq \mathsf{r}$. Intuitively, witnesses for Plonkish instances are traces of size $\mathsf{n} \times \mathsf{r}$, and r' is the number of columns whose entries are subject to a permutation constraint.
- $\sigma : [\mathsf{r}'\mathsf{n}] \to [\mathsf{r}'\mathsf{n}]$ is a permutation.
- $\mathsf{PI} \subset [\mathsf{rn}]$ indicates, intuitively speaking, the location of the public inputs in the witness traces.

An execution trace for a Plonkish instance is a vector $\mathtt{w} \in \mathbb{F}^{\mathsf{nr}}$. It is helpful to consider $\mathtt{w}$ as a matrix of size $\mathsf{n} \times \mathsf{r}$ whose entry (i, j) is given by $\mathtt{w}_{(i-1)\cdot\mathsf{r}+j}$. Such a trace is said to satisfy PL if the following conditions hold:

[4]Here we refer to the "Oplonky" relation [BGK$^+$23] as "Plonkish"

1. (Circuit constraints satifaction). For all $j \in [|\mathcal{P}|]$ and all $i \in [\mathsf{n}-1]$,

$$P_j(\mathbb{w}_{(i-1)\cdot\mathsf{r}+1}, \ldots, \mathbb{w}_{i\cdot\mathsf{r}}, \ldots, \mathbb{w}_{(i+1)\cdot\mathsf{r}}, \mathsf{q}_1[i], \ldots, \mathsf{q}_\ell[i]) = 0$$

2. (Copy constraints satisfaction). For all $i \in [\mathsf{r}'\mathsf{n}]$,

$$\mathbb{w}_i = \mathbb{w}_{\sigma(i)}$$

We denote by $\mathbb{w}^{\mathsf{PI}}$ the vector formed by the entries $\{\mathbb{w}_i \mid i \in \mathsf{PI}\}$. We call this vector the public part of $\mathbb{w}$. The Plonkish relation is then defined as:

$$\mathbf{R}_{\mathbf{Plonkish}} := \left\{ (\mathsf{PL}, \mathbb{x}, \mathbb{w}) \;\middle|\; \begin{array}{l} \mathsf{PL} = (\mathcal{P}, \mathcal{Q}, \sigma, \mathsf{PI}, \mathsf{r}, \mathsf{r}', \ell, \mathsf{n}, \mathbb{F}, \mathbb{K}), \\ \mathbb{x} \in \mathbb{F}^{|\mathsf{PI}|}, \\ \mathbb{x} = \mathbb{w}^{\mathsf{PI}}, \\ \mathbb{w} \in \mathbb{F}^{\mathsf{nr}} \text{ satisfies } \mathsf{PL} \end{array} \right\}$$

Remark 2.2. The original Plonkish relation in [Tea22, BGK+23] inlcudes further a so-called "repetition parameter". This parameter controls the number certain checks are performed. Repeating certain checks enables working over a small field, such as the Goldilocks field [Tea22]. However, recent trends in industry seem to have departed from this approach. Instead of repeating some checks, one simply samples challenges from a sufficiently large extension field [BGT23]. In this work, for simplicity, we also assume there are no repeated checks.

2.2 Grand product arguments

In the STARKish proof system, copy constraints and lookup checks are implemented by using what is called "grand product" arguments [GWC19, GW20, Gab]. In our techniques (splitting and modular-split-and-pack) these subprotocols are the ones that will undergo the most significant changes. Because of these, we describe them in detail here.

Protocol: multiset check. In a multiset check, we want to check that two vectors $a = (a_1, \ldots, a_n)$, $b = (b_1, \ldots, b_n) \in \mathbb{F}^n$ have the same elements counting repetition, possibly in a different order. With high probability over $\gamma \in \mathbb{F}$ chosen uniformly at random, this is true if and only if:

$$\prod_{i\in[n]} (a_i + \gamma) = \prod_{i\in[n]} (b_i + \gamma)$$

This fact allows one to construct the following IOP for multiset check.

Assumption: There is $D = \langle g \rangle \subset \mathbb{F}^*$ such that $|D| = n$.

Inputs: Two vectors $a = (a_1, \ldots, a_n)$, $b = (b_1, \ldots, b_n) \in \mathbb{F}^n$. We denote a, b the polynomial interpolants of a and b, respectively, over the set D. That is, for all $i \in [n]$, $\mathsf{a}(g^i) = a_i$, and similarly for b.

Protocol:

1. The verifier chooses $\gamma \in \mathbb{F}$ uniformly at random.

2. The prover gives oracle access to the verifier to the polynomial $Z \in \mathbb{F}_{<n}[X]$ such that $Z(g) = 1$, and for all $i \in \{2, \ldots, n\}$, $Z(g^i) = \prod_{1\leq j<i}(a_j + \gamma)/(b_j + \gamma)$.

3. The verifier checks that for all $x \in D$, $L_1(x) \cdot (Z(x) - 1) = 0$, and $Z(x) \cdot (\mathsf{a}(x) + \gamma) = (\mathsf{b}(x) + \gamma) \cdot Z(x \cdot g)$.

Here L_1 is the Lagrange polynomial for g, i.e. $L_1(g) = 1$ and for all $x \in D \setminus \{g\}$, $L_1(x) = 0$. Note in turn that the last two conditions can be expressed as polynomial constraints: the multiset check holds if and only if the polynomials $L_1(X) \cdot (Z(X) - 1)$ and $Z(X) \cdot (\mathsf{a}(X) + \gamma) - (\mathsf{b}(X) + \gamma) \cdot Z(X \cdot g)$ are divisible by $\prod_{y \in D}(X - y)$. The quantities $Z(g^i)$ are the partial products that are included in the accumulator column.

The multiset check is usually used to perform a permutation check between polynomials $f, h \in \mathbb{F}_{<n}[X]$, meaning that for a permutation $\sigma : [n] \to [n]$, we want to check that $h(g^i) = f(g^{\sigma(i)})$ for all i. The way this is done is by applying the multiset check to the vectors

$$(f(g) + \beta, \ldots, f(g^n) + \beta \cdot n),\ (h(g) + \beta \cdot \sigma(1), \ldots, h(g^n) + \beta \cdot \sigma(n))$$

for $\beta \in \mathbb{F}$ chosen uniformly at random. It can be extended further to a permutation check for $f_1, \ldots, f_k, h_1, \ldots, h_k \in \mathbb{F}_{<n}[X]$ given a permutation $\sigma : [nk] \to [nk]$. This is done by applying the permutation check to the vectors in $\mathbb{F}^{nk}$ constructed by setting

$$f_{(i-1)\cdot k+j} = f_j(g^i)$$

(and similarly for $h_{(i-1)\cdot k+j}$). In this case, for $j \in [k]$ one defines the polynomials

$$\mathsf{S}_{\mathsf{ID}_j}(\mathsf{g}^i) = (i-1) \cdot k + j$$
$$\mathsf{S}_{\sigma_j}(\mathsf{g}^i) = \sigma((i-1) \cdot k + j)$$

Then the polynomials $f'_j(X) = f_j(X) + \beta \mathsf{S}_{\mathsf{ID}_j}(X)$ and $h'_j(X) = h_j(X) + \beta \mathsf{S}_{\sigma_j}(X)$ correctly interpolate the value of the vectors $(f_j(\mathsf{g}) + \beta j, \ldots, f_j(\mathsf{g}^n) + \beta((n-1)k + j))$ and $(h_j(\mathsf{g}) + \beta\sigma(j), \ldots, h_j(\mathsf{g}^n) + \beta\sigma((n-1)k + j))$ respectively. Then one can apply the grand product argument to $f' = \prod f'_j$ and $h' = \prod h'_j$.

Finally, these extended permutation checks can be applied to enforce copy constraints. For a partition $\mathcal{T} = \{T_1, \ldots, T_s\}$ of $[kn]$, a set of polynomials $f_1, \ldots, f_k \in \mathbb{F}_{<n}[X]$ is said to copy-satisfy $\mathcal{T}$ if, for $f_{(j-1)\cdot n+i}$ as defined above, we have $f_\ell = f_{\ell'}$ whenever ℓ, ℓ' are in the same block T_i. This can be checked with an extended permutation check on $f_1, \ldots, f_k, f_1, \ldots, f_k$ (with polynomials repeated twice), by defining a permutation $\sigma(\mathcal{T}) : [kn] \to [kn]$ whose cycle decomposition is given by the blocks T_i.

Protocol: $\mathfrak{plookup}$. Lookups are a powerful primitive which allows to verify that a certain subset of witness values are contained in the values of predefined (lookup) tables. We present and use $\mathfrak{plookup}$ as it is the lookup argument used in STARKish, and it is also used in the RISC Zero proof system.

Suppose for simplicity that we want to check that the values taken by $f \in \mathbb{F}_{<n}[X]$ over $D = \{g, \ldots, g^n = 1\} \subset \mathbb{F}^*$ are included in the table $t \in \mathbb{F}^n$ (if the table is smaller, we pad it by repeating its last element until it has length n). Let s be the concatenation (f, t), and sort it by t (meaning that elements in s appear in the same order as they do in t). It is shown in [GW20] that it now suffices to do a multiset check between s' and $((1 + \beta)f, t')$, where:

- $\beta \in \mathbb{F}$ is chosen uniformly at random
- $s' \in \mathbb{F}^{2n}$ and $t' \in \mathbb{F}^n$ are vectors defined by $s'_i = s_i + \beta s_{i+1}$ and $t'_i = t_i + \beta t_{i+1}$

This also extends to vector lookups and multiple tables, as seen in [GW20].

Remark 2.3. Note again that the multiset check can be translated into polynomial constraints involving the polynomials representing the vectors of values we are checking, and the polynomial representing the partial product quantities. All permutation, copy constraint, and lookup checks are multiset checks. This means that if the prover commits to the partial product polynomials, all these checks can be translated into polynomial constraints involving the witness polynomials.

3 The packing technique

In this section we describe how the packing technique applies to the STARKish IOP. Recall that the STARKish IOP is the Plonky2 IOP [Tea22, BGK^{+}23], this is an IOP that can prove the satisfiability of a witness for a Plonkish instance which is essentially an AIR with copy constraints [GWC19]. In the literature, this arithmetization –with small variations in their formulations– is also referred to as RAPs or TurboPlonk. When adding lookups to the relation and support for lookup arguments in the IOP, it is also referred to as UltraPlonk. The packing technique applied to the STARKish IOP results in an IOP, which we call *STARKPack*, that can prove the simultaneous satisfiablity of witnesses for different Plonkish instances. We then explain how to extend that argument to support $\mathfrak{plookup}$, which we still call the STARKPack IOP, or STARKPack with lookups. Then, we provide soundness proofs for the STARKPack IOP.

Definition 3.1 (Packed Plonkish relation)**.** Informally, the packed Plonkish relation is a conjunction of Plonkish relations (cf Definition 2.1). Formally, it is defined as:

$$\mathbf{R}_{\mathbf{PPlonkish}} := \left\{ (N, \mathsf{PL}_1, \ldots, \mathsf{PL}_N, \vec{\mathbb{x}}, \vec{\mathbb{w}}) \;\middle|\; \begin{array}{l} N \text{ is an integer} \geq 1, \\ \mathsf{PL}_i = (\mathcal{P}_i, \mathcal{Q}_i, \sigma_i, \mathsf{PI}_i, \mathsf{r}_i, \mathsf{r}'_i, \ell_i, \mathsf{n}_i, \mathbb{F}, \mathbb{K}) \\ \qquad\quad \text{is a Plonkish relation } \forall i \in [N], \\ \vec{\mathbb{x}} = (\mathbb{x}^1, \ldots, \mathbb{x}^N) \in \prod\limits_{i \in [N]} (\mathbb{F}^{\mathsf{PI}_i}), \\ \vec{\mathbb{w}} = (\mathbb{w}^1, \ldots, \mathbb{w}^N) \in \prod\limits_{i \in [N]} (\mathbb{F}^{\mathsf{n}_i \mathsf{r}_i}), \\ \text{For all } i \in [N],\ (\mathbb{w}^i)^{\mathsf{PI}_i} = \mathbb{x}^i, \\ \text{For all } i \in [N],\ \mathbb{w}^i \text{ satisfies } \mathsf{PL}_i \end{array} \right\}$$

The STARKPack IOP is an IOP constructed from the STARKish IOP that can prove the simultaneous satisfiability of Plonkish instances, or equivalently, the satisfiability of a Packed Plonkish relation.

3.1 Description of the STARKPack IOP

The STARKPack IOP, denoted by $\Pi_{\mathsf{STARKPack}}$, mainly follows the STARKish protocol. We fix an integer $N \geq 1$, and a finite field extension $\mathbb{K}$ of $\mathbb{F}$. We fix N Plonkish instances, for each $i \in [N]$ their parameters will be denoted by $\mathsf{PL}_i = (\mathcal{P}_i, \mathcal{Q}_i, \sigma_i, \mathsf{PI}_i, \mathsf{r}_i, \mathsf{r}'_i, \ell_i, \mathsf{n}_i, \mathbb{F}, \mathbb{K})$. For each i, we assume that there is a multiplicative subgroup $\mathsf{D}_i = \langle \mathsf{g}_i \rangle \subset \mathbb{F}^{\times}$ of size n_i. Its vanishing polynomial is denoted by $Z_{\mathsf{D}_i}(X) \in \mathbb{F}_{<\mathsf{n}_i}[X]$. Further, we assume that if $\mathsf{n}_i \leq \mathsf{n}_{i'}$, then D_i is a multiplicative subgroup of $\mathsf{D}_{i'}$ (in which case we necessarily have $\mathsf{n}_i \mid \mathsf{n}_{i'}$).

We also fix $\mathsf{n} = \max_i \mathsf{n}_i$. The integer n is the largest trace length among all instances. It corresponds to a trace evaluation domain $\mathsf{D} = \langle \mathsf{g} \rangle \subset \mathbb{F}^{\times}$ of size n. We fix an evaluation domain $\mathsf{H} \subset \mathbb{K}^{\times}$ which contains a non-trivial coset of D. When we say that the prover gives oracle access to a function, we mean that the prover interpolates a vector with entries in $\mathbb{F}$ over a certain subgroup of $\mathbb{F}^{\times}$; and then provides oracle access to this interpolant over H. We fix a witness (or execution trace) $\mathbb{w} \in \prod_{i \in [N]} (\mathbb{F}^{\mathsf{n}_i \mathsf{r}_i})$ and public part $\mathbb{x}$ for the Packed Plonkish instance $(N, \mathsf{PL}_1, \ldots, \mathsf{PL}_N)$. Finally, we fix auxiliary parameters $\mathsf{aux} = (\vec{t}, s)$ for the Batched-FRI protocol. The parameter $\vec{t} = (t_1, \ldots, t_m) \in \mathbb{N}$ indicates that we will use t_i-to-1 maps in the i-th COMMIT phase of FRI, and s is the number of repetitions of the QUERY phase of FRI.

Preprocessing. The prover and verifier compute the selector polynomials as:

$$\forall i \in [N],\ \forall j \in [\ell_i],\ \mathsf{q_j}^i(X) = \sum_{k=1}^{\mathsf{n}_i} \mathsf{q_j}^i[k] \cdot L_k^i(X)$$

where the L_k^i are the Lagrange polynomials for the basis $\{\mathsf{g}_i, \ldots, \mathsf{g}_i^{\mathsf{n}_i} = 1\}$. Additionally, for all $i \in [N]$ and all $j \in [\mathsf{r}_i']$, the prover and verifier agree on the splitting parameter u_i for the permutation argument (see below), and compute the permutation polynomials $\mathsf{S}^i_{\mathsf{ID}_j}, (\mathsf{S}^i_{\sigma_i})_j$ described in Section 2.2.

Additionally, define the FRI degree correction terms $\mathsf{e}_i := (\mathsf{n}-1)-(\mathsf{n}_i-2)$ for all $i \in [N]$.

Round 1. The prover computes the wire polynomials as follows:

$$\forall i \in [N],\ \forall j \in [\mathsf{r}_i],\ \mathsf{a}^i_j(X) = \sum_{k=1}^{\mathsf{n}_i} \mathbb{w}_{(j-1)\cdot \mathsf{n}_i + k} \cdot L^i_k(X)$$

The prover then gives oracle access to the wire polynomials $\mathsf{a}^i_j : \mathsf{H} \to \mathbb{K}$ to the verifier. The verifier samples uniform randomness used in the permutation argument $\beta, \gamma \in \mathbb{K}$. The prover will use the same randomness for all copy constraint arguments and lookup arguments (see next section for lookups) across instances.

Round 2. The prover sends oracle access to all permutation and partial product polynomials. Here the terminology does not concord with what we have presented in Section 2.1, so we explain it. For simplicity fix $i \in [N]$. The prover is supposed to provide oracle access to a polynomial $z^i \in \mathbb{F}_{<\mathsf{n}_i}[X]$ such that:

$$z^i(\mathsf{g}X) \cdot g^i(X) = f^i(X) \cdot z^i(X) \tag{1}$$

where:

$$f^i(X) = \prod_{l=1}^{[\mathsf{r}_i]} (\mathsf{a}^i_l(X) + \beta \cdot \mathsf{S}^i_{\mathsf{ID}_l}(X) + \gamma) := \prod_{l=1}^{[\mathsf{r}_i]} f^i_l(\beta, \gamma, X)$$
$$g^i(X) = \prod_{l=1}^{[\mathsf{r}_i]} (\mathsf{a}^i_l(X) + \beta \cdot (\mathsf{S}^i_{\sigma_i})_l(X) + \gamma) := \prod_{l=1}^{[\mathsf{r}_i]} g^i_l(\beta, \gamma, X)$$

We have called z^i the partial product polynomial before. However, if r_i is large, then so is the degree of the polynomial constraint in Eq. (1). To reduce its degree, it is split into many polynomial constraints. For the integer u_i determined in preprocessing, the prover splits the products as follows. Define $s_i = \lceil \mathsf{r}_i / u_i \rceil$, then for each $k \in [s_i]$:

$$\overline{f^i_k}(\beta, \gamma, X) = \overline{f^i_k}(X) := \prod_{l=(u_i-1)\cdot k+1}^{u_i \cdot k} f^i_l(\beta, \gamma, X)$$
$$\overline{g^i_k}(\beta, \gamma, X) = \overline{g^i_k}(X) := \prod_{l=(u_i-1)\cdot k+1}^{u_i \cdot k} g^i_l(\beta, \gamma, X)$$

Now the prover will compute polynomials $\pi^i_1, \ldots, \pi^i_{s_i-1} \in \mathbb{F}_{<\mathsf{n}_i}[X]$ such that for all $m \in [\mathsf{n}_i]$:

$$\pi^i_1(\mathsf{g}^m) = z^i(\mathsf{g}^m)\overline{f^i_1}(\mathsf{g}^m)\overline{g^i_1}(\mathsf{g}^m)^{-1}$$
$$\pi^i_k(\mathsf{g}^m) = \pi^i_{k-1}(\mathsf{g}^m)\overline{f^i_k}(\mathsf{g}^m)\overline{g^i_k}(\mathsf{g}^m)^{-1},\ \text{for } k = 2, \ldots, s_i - 1$$

These equalities can then be translated into polynomial constraints of the following form:

$$\begin{aligned}
z^i(X)\overline{f^i_1}(X) - \pi^i_1(X)\overline{g^i_1}(X) &\equiv 0 \mod Z_{\mathsf{D}_i}(X) \\
\pi^i_{k-1}(X)\overline{f^i_k}(X) - \pi^i_k(X)\overline{g^i_k}(X) &\equiv 0 \mod Z_{\mathsf{D}_i}(X),\ \text{for } k = 2, \ldots, s_i - 1 \\
\pi^i_{s_i-1}(X)\overline{f^i_{s_i}}(X) - z^i(\mathsf{g}X)\overline{g^i_{s_i}}(X) &\equiv 0 \mod Z_{\mathsf{D}_i}(X) \\
L^i_1(X)(z^i(X) - 1) &\equiv 0 \mod Z_{\mathsf{D}_i}(X)
\end{aligned}$$

The π_k^i will now be called the partial product polynomials, while the polynomial z^i will be called the permutation polynomial. The prover sends oracle access to the permutation and partial product polynomials $(z^i)_{i\in[N]}, (\pi_k^i)_{i\in[N],k\in[s_i-1]}$.

The verifier samples uniform randomness $\alpha \in \mathbb{K}$.

Round 3. The prover computes polynomials $(\mathsf{u}^i, \mathsf{d}^i, \mathsf{quot}^i)_{i\in[N]}$ as follows:

$$\begin{aligned}\mathsf{u}^i(X) := &(z^i(X)\overline{f_1^i}(X) - \pi_1^i(X)\overline{g_1^i}(X))\alpha \\ &+ \sum_{k=2}^{s_i}(\pi_{k-1}^i(X)\overline{f_k^i}(X) - \pi_k^i(X)\overline{g_k^i}(X))\alpha^k \\ &+ (\pi_{s_i-1}^i(X)\overline{f_{s_i}^i}(X) - z^i(\mathsf{g}X)\overline{g_{s_i}^i}(X))\alpha^{s_i+1} \\ &+ L_1^i(X)(z^i(X)-1)\alpha^{s_i+2}\end{aligned}$$

$$\begin{aligned}\mathsf{d}^i(X) := &\sum_{k\in[|\mathcal{P}|]} \alpha^{k-1}P_k^i(\mathsf{a}_1^i(X),\ldots,\mathsf{a}_{\mathsf{r}_i}^i(X),\mathsf{q_1}^i,\ldots,\mathsf{q}_{\ell_i}{}^i) \\ &+ \alpha^{|\mathcal{P}|}\mathsf{u}^i(X)\end{aligned}$$

$$\mathsf{quot}^i(X) := \mathsf{d}^i(X)/Z_{\mathsf{D}_i}(X)$$

Then the prover splits each $\mathsf{quot}^i(X)$ into degree n_i polynomials $\mathsf{quot}_1^i,\ldots,\mathsf{quot}_\nu^i$ such that $\mathsf{quot}^i(X) = \sum_l X^{\mathsf{n}_i\cdot(l-1)}\mathsf{quot}_l^i(X)$.

The prover gives oracle access to the maps $\mathsf{quot}_1^i,\ldots,\mathsf{quot}_\nu^i : \mathsf{H} \to \mathbb{K}$. The verifier samples uniform randomness $\mathfrak{z} \in \mathbb{K} \setminus (\mathsf{H} \cup \mathsf{g}^{-1}\mathsf{H})$.

Round 4. The prover sends the evaluations

$$\mathsf{eval} = (\mathsf{a}_j^i(\mathfrak{z}), \mathsf{a}_j^i(\mathsf{g}\mathfrak{z}), \mathsf{quot}_l^i(\mathfrak{z}), \pi_m^i(\mathfrak{z}), z^i(\mathfrak{z}), z^i(\mathsf{g}\mathfrak{z}), \mathsf{q_c}{}^i(\mathfrak{z}), \mathsf{S}_{\mathsf{ID}_j}^i(\mathfrak{z}), (\mathsf{S}_{\sigma_i}^i)_j(\mathfrak{z}))$$

for all $i \in [N], j \in [\mathsf{r}_i], l \in [\nu], m \in [s_i-1], c \in [\ell_i]$. The verifier samples uniform FRI randomness $\eta \in \mathbb{K}$.

Batched-FRI. The prover and verifier engage in a Batched-FRI protocol with auxiliary parameters aux to check proximity to the Reed-Solomon code $\mathsf{RS}(\mathbb{K},\mathsf{H},\mathsf{n})$ of the function defined as follows. Consider the vector of functions defined as:

$$\left(\frac{\mathsf{a}_j^i(X)-\mathsf{a}_j^i(\mathfrak{z})}{X-\mathfrak{z}}, \frac{\mathsf{a}_j^i(X)-\mathsf{a}_j^i(\mathsf{g}\mathfrak{z})}{X-\mathsf{g}\mathfrak{z}}, \frac{\mathsf{quot}_l^i(X)-\mathsf{quot}_l^i(\mathfrak{z})}{X-\mathfrak{z}}, \ldots, \frac{(\mathsf{S}_{\sigma_i}^i)_j(X)-(\mathsf{S}_{\sigma_i}^i)_j(\mathfrak{z})}{X-\mathfrak{z}}\right) \tag{2}$$

for all $i \in [N], j \in [\mathsf{r}_i], l \in [\nu], m \in [s_i-1], c \in [\ell_i]$ (it contains all relevant quotients of functions whose evaluations appear in eval). For simplicity, relabel this vector as $\vec{F} = (F_1(X), F_2(X), \ldots, F_M(X))$ (for some integer M). The function we apply FRI to is the linear combination of the F_j's multiplied by the factors $(\eta^j + \eta^{j+2}X^{\mathsf{e}(F_j)})$. Here e is defined by the fact that if F_j is supposedly of degree at most $\mathsf{n}_i - 2$, then $\mathsf{e}(F_j) = \mathsf{e}_i$ (the FRI degree correction term defined in Preprocessing).

Decision.

1. **Circuit and copy constraint check.** The verifier uses eval to verify that for all $i \in [N]$, $\mathsf{d}^i(\mathfrak{z}) = \mathsf{quot}^i(\mathfrak{z})Z_{\mathsf{D}_i}(\mathfrak{z})$. If this check fails, the verifier rejects.
2. **Batched FRI check.** If the prover fails the Batched-FRI protocol with auxiliary parameters aux, the verifier rejects.

   If all checks pass, the verifier accepts the proof.

Commitments in the STARKPack IOP. When we instantiate this IOP protocol with Merkle tree commitments, in Round 1 the prover needs to provide a commitment to the evaluations of the wire polynomials $\mathsf{a}^i_j : \mathsf{H} \to \mathbb{K}$ coming from different instances. There is a way to compactly pack leaves so that they contain the necessary evaluations of all relevant functions. To do this, the prover performs an IFFT over D_i, and then an FFT over H taking $O(\mathsf{n}_i \log(\mathsf{n}_i) + |\mathsf{H}| \log(|\mathsf{H}|))$ field operations per function. Looking at Eq. (2), when querying the Batched-FRI function at $x_0 \in \mathsf{H}$, we need to provide all the evaluations of the wire polynomials at x_0. Therefore, the prover can pack all the evaluations at a point $x_0 \in \mathsf{H}$ of all wire polynomials (from all Plonkish instances) into one leaf of the Merkle tree, and compute one root for all functions.

The same holds for other functions computed by the prover during the protocol. This idea was also presented in [MAGABMMT23].

3.2 Adding $\mathfrak{plookup}$ to the STARKPack IOP

As we mentioned in Section 2.2, $\mathfrak{plookup}$ allows one to verify that $f \in \mathbb{F}^n$ is included in $t \in \mathbb{F}^d$ (meaning that $\{f\} \subset \{t\}$). Because we may pad t by repeating its last value, we may assume $d = n$ without loss of generality. The $\mathfrak{plookup}$ protocol applies a multiset check to $s', ((1+\beta)f, t') \in \mathbb{F}^{2n}$ for a uniformly randomly $\beta \in \mathbb{F}$, where:

- $s \in \mathbb{F}^{2n}$ is (f, t) sorted by t (elements in s appear in the same order as they do in t)
- $s'_i = s_i + \beta s_{i+1}$ and $t'_i = t_i + \beta t_{i+1}$

We make no difference in notation between the vectors and their polynomial interpolants over the domain $\mathsf{D} = \langle \mathsf{g} \rangle \subset \mathbb{F}^\times$ of size n, except that we denote by $h_1, h_2 \in \mathbb{F}_{<n}[X]$ the polynomials such that $h_1(\mathsf{g}^i) = s_i$ and $h_2(\mathsf{g}^i) = s_{n+i}$. For uniformly random $\beta, \gamma \in \mathbb{F}$, denote by $Z \in \mathbb{F}_{<n}[X]$ the polynomial such that:

$$\begin{aligned}
Z(\mathsf{g}) &= 1 \\
Z(\mathsf{g}^i) &= \frac{(1+\beta)^{i-1} \prod_{j<i}(f_j + \gamma) \cdot \prod_{1 \le j < i}(\gamma(1+\beta) + t_j + \beta t_{j+1})}{\prod_{1 \le j < i}(\gamma(1+\beta) + s_j + \beta s_{j+1})(\gamma(1+\beta) + s_{n+j} + \beta s_{n+j+1})}, \quad \text{for } 2 \le i < n \\
Z(\mathsf{g}^n) &= 1
\end{aligned}$$

The relevant multiset check translates into the polynomial constraints:

$$L_i(X)(Z(X) - 1) \equiv 0 \mod Z_{\mathsf{D}}(X), \text{ for } i = 1, n$$

$$\begin{aligned}
&Z(X) \cdot (1+\beta)(f(X) + \gamma)(\gamma(1+\beta) + t(X) + \beta t(\mathsf{g}X)) \equiv \\
&\qquad Z(\mathsf{g}X) \cdot (\gamma(1+\beta) + h_1(X) + \beta h_1(\mathsf{g}X))(\gamma(1+\beta) + h_2(X) + \beta h_2(\mathsf{g}X)) \mod Z_{\mathsf{D}}(X)
\end{aligned}$$

This means that if the lookup table t is encoded as one of the wire polynomials, then the verifier can us the same randomness sampled in Round 1 to implement $\mathfrak{plookup}$. The prover will need to additionally commit to h_1, h_2, Z over the evaluation domain. If the degree of the polynomial constraints gets too large, we can also split it into smaller degree constraints exactly as for the permutation argument. Then the polynomial constraints will be included in the d polynomials in Round 3. The rest of the protocol remains the same. We can modify the Plonkish instances slightly to include lookup tables in the public data, we still refer to these instances as Plonkish instances. By implementing $\mathfrak{plookup}$ in the STARKPack IOP as we described, we obtain an IOP to prove the simultaneous satisfiability of Plonkish instances with lookups, still denoted by $\Pi_{\mathsf{STARKPack}}$.

3.3 Benchmarks

In this section we present benchmarks comparing the performance of STARKPack and the protocol whereby each proof is created separately. For a detailed description of the costs of STARKPack we refer to Table 1 in the introduction of the paper.

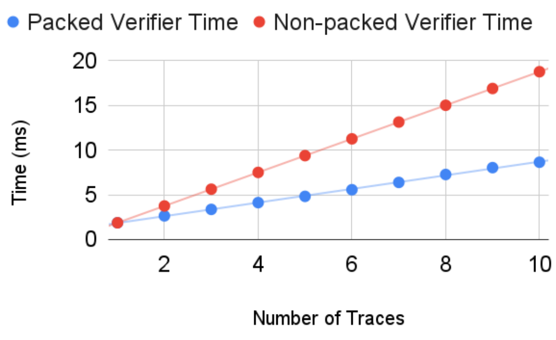
In general, STARKPack optimizes both the computational and communication complexities of STARK proof generation. For N programs, there is a single FRI proof. This reduces the joint proof size and the verification time. All N proofs share a single Merkle commitment and decommitment step, and rely on a single composition polynomial evaluation. This also helps reduce both the proving and verification time, and shrinks the proof size. Furthermore, the verifier samples the same randomness for all N instances, which reduces the communication complexity.

We recall the simple programs used for benchmarks: for Winterfell, checking that $x_{i+1} = x_i^3 + 42 \mod (p)$ for some prime p until the rows of the trace were 2^{16} ; and for RISC Zero, the program proved the knowledge of the prime factors of a number over a trace with 2^{16} rows.

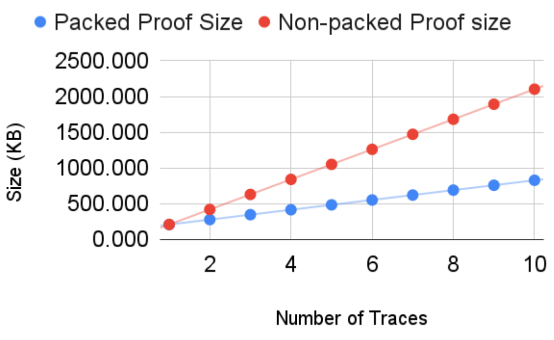
A note about the RISC Zero and Winterfell proof systems. The arithmetization used in RISC Zero [BGT23] is a RAP. This is another way to formalize the STARK-ish/TurboPlonk/UltraPlonk arithmetization. There are selector columns (which act as the selector polynomials), and the prover and verifier engage in a randomized preprocessing phase to compute the values of the relevant permutation/partial product polynomials in additional trace columns called accumulator columns. Therefore, this arithmetization is also amenable to permutation arguments and lookups. The IOP used in RISC Zero is similar to ethSTARK [Sta21] with support for permutation and lookup arguments, and closely follows the blueprint we described in Section 3.1 when the number of traces is $N = 1$. For this reason, the same ideas that we applied to obtain the STARKPack IOP can be applied in this setting. In this way, we obtain a proof system that can show the simultaneous satisfiability of multiple RAP instances.

Winterfell uses ethSTARK for AIRs, and the ideas apply in the same way (AIRs are RAPs without selector or accumulator columns).

Benchmarks. For our benchmarks, we applied the STARKPack ideas for the same program run many times with different inputs. Fig. 1 shows the STARKPack performance improvement when implemented in RISC Zero. The acceleration for the proving time has been small, around 5% for the benchmarked instances: this is due to the large number of columns. The Winterfell benchmarks presented below for traces with fewer columns show how the proving time optimization depends on the number of columns (Fig. 5).



(a) Verification Time

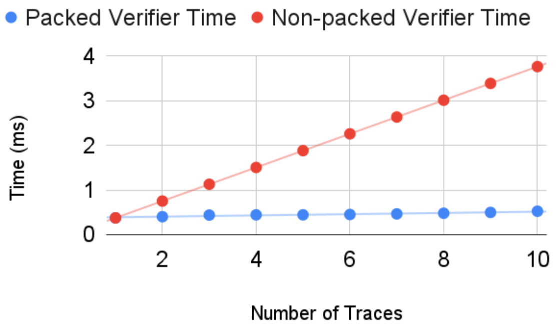


(b) Proof Sizes

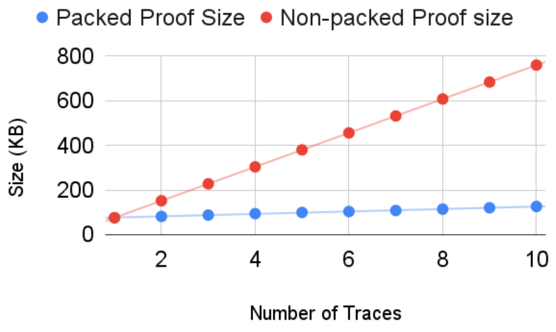
Figure 4: RISC Zero STARKPack Implementation Performance

The diagrams below show the STARKPack performance improvement implemented in Winterfell for AIR traces comprised of 10, 100 and 275 columns.

Remark 3.2. When we run our benchmarks on the same program with different inputs, further optimizations are possible. For instance, all programs will use the same selector columns and lookup tables. We can use these columns only once for all RAPs. In this extension, only

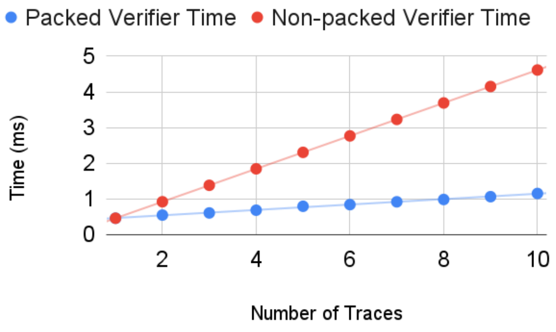


(a) Verification Time

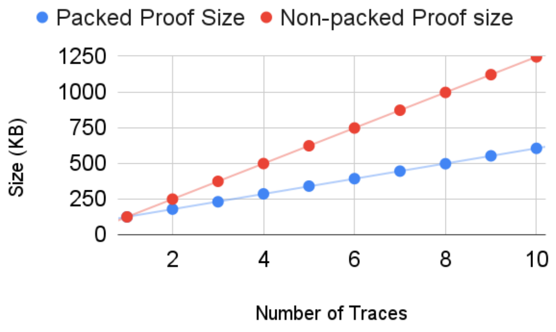


(b) Proof Sizes

Figure 5: Winterfell STARKPack Implementation Performance for AIR traces of 10 columns

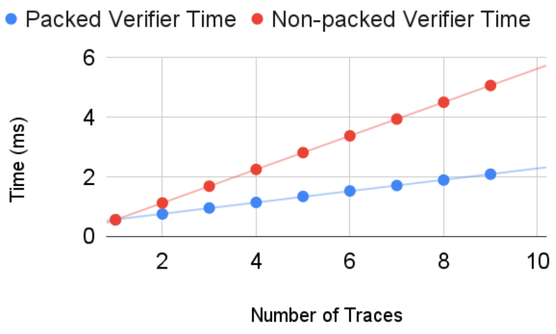


(a) Verification Time

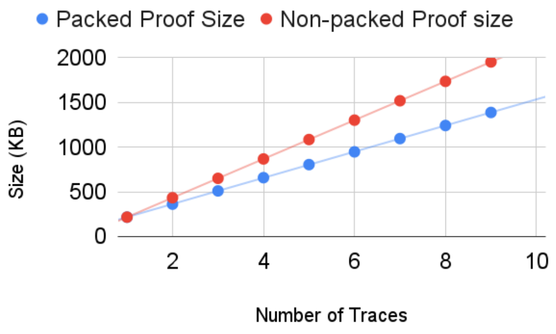


(b) Proof Sizes

Figure 6: Winterfell STARKPack Implementation Performance for AIR traces of 100 columns



(a) Verification Time



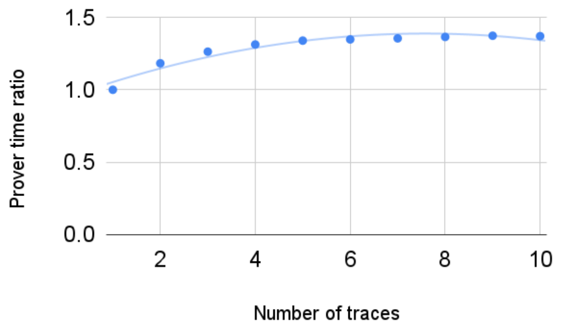
(b) Proof Sizes

Figure 7: Winterfell STARKPack Implementation Performance for AIR traces of 275 columns

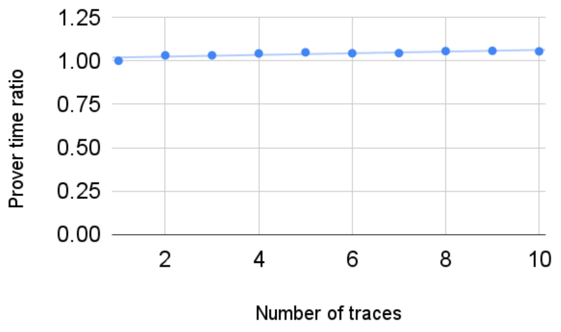
one RAP has selector and lookup columns, while all others only have accumulator columns. Constraints can span across different RAPs to enforce the selector constraints across RAPs.

3.4 Soundness Analysis

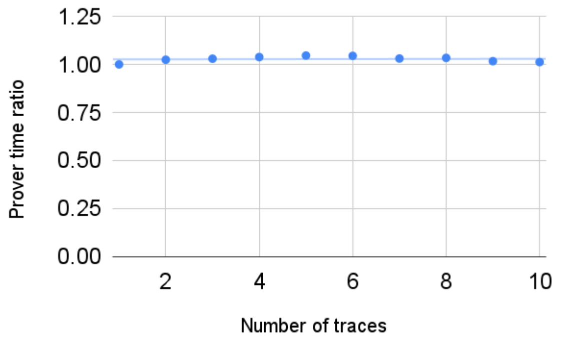
In [BGK+23, Sta21] it was proved that both the ethSTARK and the STARKish (called OPlonky in [BGK+23]) IOPs are round-by-round knowledge sound, and, consequently, special sound [BGTZ23]. These are strong soundness properties that imply knowledge soundness after compiling the IOP via Merkle tree vector commitments and the Fiat-Shamir transformation. As a consequence, the ethSTARK and Starkish protocols, in their final



(a) Number of Columns = 10



(b) Number of Columns = 100



(c) Number of Columns = 275

Figure 8: Winterfell: Proving Time Optimizations for various AIR traces

non-interactive and succinct form, are knowledge sound.

We argue that the same properties hold for the STARKPack IOP $\Pi_{\mathsf{STARKPack}}$ and its Merkle-tree+Fiat-Shamir transformed final form. To show this, we first prove the following result, which states, essentially, that an instance of the Packed Plonkish relation is equivalent to an instance of the standard Plonkish relation. We use this fact to later derive the knowledge soundness of the STARKpack IOP from the knowledge soundness of STARKish IOP.

Lemma 3.3. *Let $N \geq 1$ and let*

$$\mathsf{PACK} = (N, \mathsf{PL}_1, \dots, \mathsf{PL}_N, \mathbb{x}_1, \dots, \mathbb{x}_N) \tag{3}$$

be an instance of the Packed Plonkish relation $\mathbf{R_{PPlonkish}}$, with

$$\mathsf{PL}_i = (\mathcal{P}_i, \mathcal{Q}_i, \sigma_i, \mathsf{PI}_i, \mathsf{r}_i, \mathsf{r}'_i, \ell_i, \mathsf{n}_i, \mathbb{F}, \mathbb{K})$$

for $i \in [N]$. Then there exists a (deterministcally and polynomially computable) instance $(\mathsf{PL}_{\mathsf{PACK}}, \mathbb{x}_{\mathsf{PACK}})$ for the Plonkish relation $\mathbf{R_{Plonkish}}$, with

$$\mathsf{PL}_{\mathsf{PACK}} = (\mathcal{P}, \mathcal{Q}, \sigma, \mathsf{PI}, \mathsf{r}, \mathsf{r}', \ell, \mathsf{n}, \mathbb{F}, \mathbb{K}),$$

with the following properties: 1) $\mathsf{r} = \sum_{i \in [N]} \mathsf{r}_i$; 2) there is an efficient deterministic algorithm that, given a a valid witness $(\mathbb{w}_1, \dots, \mathbb{w}_N) \in \prod_{i \in [N]} \mathbb{F}^{\mathsf{n}_i \mathsf{r}_i}$ for PACK, outputs a valid witness $\mathbb{w}$ for $\mathsf{PL}_{\mathsf{PACK}}$.

Proof. Assume first that $\mathsf{r}'_i = \mathsf{r}_i$ for all $i \in [N]$ (i.e. there are no advice wires) and $\mathsf{n}_i = \mathsf{n}_j$ for all $i, j \in [N]$. The idea in this case is that the Packed Plonkish instance is equivalent to a Plonkish instance where N execution traces are "glued together horizontally", and the permutations in each of the Plonkish instances are combined into a single permutation of the glued trace. Formally, we define $\mathsf{PL}_{\mathsf{PACK}} = (\mathcal{P}, \mathcal{Q}, \sigma, \mathsf{PI}, \mathsf{r}, \mathsf{r}', \ell, \mathsf{n}, \mathbb{F}, \mathbb{K})$ as follows. First, for each $i \in [N]$, let

$$\mathbf{X}_i := (X_{i1}, \dots, X_{i\mathsf{r}_i}), \mathbf{Y}_i := (Y_{i1}, \dots, Y_{i\mathsf{r}_i}), \mathbf{S}_i := (S_{i1}, \dots, S_{i\ell_i})$$

be tuples of variables. Intuitively, $\mathbf{X}_i, \mathbf{Y}_i$ are variables reserved for the values of the "i-th chunk” of r_i columns of the glued execution trace, and $\mathbf{S}_i$ are variables that will take the values in the selector vectors $\mathcal{Q}_i$.

Next we let $\mathsf{r} = \sum_{i\in[N]} \mathsf{r}_i$, $\mathsf{n} = \mathsf{n}_i$ for all i (as we are assuming all n_i are the same), $\mathsf{r}' = \mathsf{r}$, and $\ell := \sum_{i\in[N]} \ell_i$. Further, we define $\mathcal{P}$ to be the set of polynomials

$$\mathcal{P} := \{P(\mathbf{X}_i, \mathbf{Y}_i, \mathbf{S}_i) \mid i \in [N], P \in \mathcal{P}_i\}.$$

Formally, we view each of these as a polynomial on the $2\mathsf{r} + \ell$ variables

$$(\mathbf{X}_1, \dots, \mathbf{X}_N, \mathbf{Y}_1, \dots, \mathbf{Y}_N, \mathbf{S}_1, \dots, \mathbf{S}_N).$$

We further let

$$\mathcal{Q} := (\mathcal{Q}_1, \dots, \mathcal{Q}_N).$$

To make the upcoming definitions clearer, we identify the sets $[\mathsf{n}] \times [\mathsf{r}_i]$ and $[\mathsf{n}\mathsf{r}_i]$ ($i \in [N]$) via the bijection $(j-1)\cdot \mathsf{n} + k \mapsto (j, k+1)$, for $j \geq 1$ and $0 \leq k < \mathsf{n}$. Further, we identify $[\mathsf{n}] \times [\mathsf{r}]$ and $[\mathsf{nr}]$ via an analogous bijection. For the rest of the proof, we look at all sets $[\mathsf{nr_i}]$ (and similarly for $[\mathsf{nr}]$) as $[\mathsf{n}] \times [\mathsf{r_i}]$, unless otherwise stated.

Now, we let

$$\mathsf{PI} := \left(\mathsf{PI}_1^{\mathsf{r}_1}, \mathsf{PI}_2^{\mathsf{r}_1,\mathsf{r}_2}, \dots, \mathsf{PI}_N^{\mathsf{r}_1,\dots,\mathsf{r}_N}\right),$$

where $\mathsf{PI}_1^{\mathsf{r}_1} := \mathsf{PI}$ and, for $i > 1$, $\mathsf{PI}_i^{\mathsf{r}_1,\dots,\mathsf{r}_i}$ denotes PI_i after translating the first component of the indices in PI_i by $\sum_{j<i} \mathsf{r}_j$. Further, we define the input $\mathbb{x}$ for $\mathsf{PL}_{\mathsf{PACK}}$ to be $\mathbb{x} = (\mathbb{x}_1, \dots, \mathbb{x}_N)$.

Next, we define $\sigma : [\mathsf{n}] \times [\mathsf{r}] \to [\mathsf{n}] \times [\mathsf{r}]$ as the result of composing the permutations $\sigma_1, \dots, \sigma_N$ in the following way:

$$\sigma := \sigma_1^{\mathsf{Ext}} \cdots \sigma_N^{\mathsf{Ext}} \tag{4}$$

where $\sigma_i^{\mathsf{Ext}} : [\mathsf{n}] \times [\mathsf{r}] \to [\mathsf{n}] \times [\mathsf{r}]$ is the permutation that acts as σ_i on the set $[\mathsf{n}] \times \{1 + \sum_{j<i} \mathsf{r}_j, \dots, \sum_{j<(i+1)} \mathsf{r}_j\}$ (which is in bijective correspondence with $[\mathsf{n}] \times [\mathsf{r}_i]$), and fixes the rest of the elements in $[\mathsf{n}] \times [\mathsf{r}]$. In (4), concatenation stands for permutation composition.

Next, we describe how to, given a valid witness $(\mathbb{w}_1, \dots, \mathbb{w}_N)$ for PACK, construct a witness $\mathbb{w}$ for $\mathsf{PL}_{\mathsf{PACK}}$. Following our previous convention, we look at each $\mathbb{w}_i$ as a $\mathsf{n} \times \mathsf{r}_i$ matrix, and let $\mathbb{w}$ be the $\mathsf{n} \times \mathsf{r}$ matrix obtained by concatenating the matrices $\mathbb{w}_i$. Note that, by construction, $\mathbb{w}$ satisfies the copy constraints of $\mathsf{PL}_{\mathsf{PACK}}$. Further, since $\mathbb{w}_i^{\mathsf{PI}_i} = \mathbb{x}_i$ for all $i \in [N]$, we have $\mathbb{w}^{\mathsf{PI}} = \mathbb{x}$. Further, $\mathbb{w}$ satisfies the circuit constraints of $\mathsf{PL}_{\mathsf{PACK}}$, since, given $i \in [N]$ and

$$P = P(\mathbf{X}, \mathbf{Y}, \mathbf{Z}) = P(\mathbf{X}_i, \mathbf{Y}_i, \mathbf{Z}_i) = P(X_{i1}, \dots, X_{ir i}; Y_{i1}, \dots, Y_{iri}; S_{i1}, \dots, S_{i\ell_i}) \in \mathcal{P}_i,$$

we have that for all row index $j \in [\mathsf{n}]$ (below, we look at $\mathbb{w}$ again as a vector from $\mathbb{F}^{\mathsf{nr}}$, rather than a matrix from $\mathbb{F}^{\mathsf{n}\times\mathsf{r}}$),

$$\begin{aligned} &P(\mathbb{w}_{(j-1)\mathsf{r}+1}, \dots, \mathbb{w}_{(j-1)\mathsf{r}+\mathsf{r}-1}; \mathbb{w}_{j\mathsf{r}+1}, \dots, \mathbb{w}_{j\mathsf{r}+\mathsf{r}-1}; (\mathsf{q}[j] \mid \mathsf{q} \in \mathcal{Q})) \\ =&P(\mathbb{w}_{(j-1)\mathsf{r}+i+1}, \dots, \mathbb{w}_{(j-1)\mathsf{r}+\mathsf{r}_i-1}; \mathbb{w}_{j\mathsf{r}+i+1}, \dots, \mathbb{w}_{j\mathsf{r}+\mathsf{r}_i+i-1}; (\mathsf{q}[j] \mid \mathsf{q} \in \mathcal{Q}_i)). \end{aligned}$$

This expression is zero by construction of $\mathbb{w}$.

This completes the proof of the first part of the lemma under the assumption that $\mathsf{r}_i' = \mathsf{r}_i$ for all $i \in [N]$ and $\mathsf{n}_i = \mathsf{n}_j$ for all $i, j \in [N]$. Assume now that not all n_i’s are the same. The key claim we make in this scenario is that for any Plonkish instance of the form $\mathsf{PL} = (\mathcal{P}, \mathcal{Q}, \sigma, \mathsf{PI}, \mathsf{r}, \mathsf{r}', \ell, \mathsf{n}, \mathbb{F}, \mathbb{K})$, and any power of two $\widetilde{\mathsf{n}} > \mathsf{n}$, there is a Plonkish instance of the form $\widetilde{\mathsf{PL}} = (\mathcal{P}, \widetilde{\mathcal{Q}}, \widetilde{\sigma}, \mathsf{PI}, \mathsf{r}, \mathsf{r}', \ell, \widetilde{\mathsf{n}}, \mathbb{F}, \mathbb{K})$ an an efficient algorithm that transforms valid witness for PL to valid witnesses for $\widetilde{\mathsf{PL}}$, and viceversa. Once this claim is proved, we can assume without loss of generality that $\mathsf{n}_i = \max_{j\in[N]} \mathsf{n}_j$ for all $i \in [N]$, by replacing PL_i with $\widetilde{\mathsf{PL}_i}$ for all $i \in [N]$ such that $\mathsf{n}_i < \max_{j\in[N]} \mathsf{n}_j$.

Finally, the case where $\mathsf{r}'_i \neq \mathsf{r}_i$ for some $i \in [N]$ can be dealt with with analogous arguments as the ones given so far. In a bit more detail, the proof proceeds in the same way, except that, formally speaking, we cannot have $\mathbb{w}$ just be the concatenation of the witness matrices $\mathbb{w}_1, \dots, \mathbb{w}_N$. This is because, again formally, the witness (or execution trace) of a Plonkish relation has all its advice wires (i.e. columns) on the rightmost part of the trace. Accordingly, in the presence of advice wires, we construct $\mathbb{w}$ by concatenating $\mathbb{w}_1, \dots, \mathbb{w}_N$ and then moving the advice wires of each submatrix $\mathbb{w}_i$ to the right-most columns of $\mathbb{w}$. Then, $\mathsf{PI}, \mathbb{x}, \sigma, \mathbf{X}, \mathbf{Y}, \mathbf{Z}$ need to be modified in order to accomoddate this rearrangement of the columns in $\mathbb{w}$. □

Now Lemma 3.3 and [BGK+23, Sta21] yield:

Corollary 3.4. *The STARKPack IOP* $\Pi_{\mathsf{STARKPack}}$ *has knowledge soundness error, given an instance* PACK *as in* (3),

$$\varepsilon_{\mathsf{ks}} = \frac{1}{2\eta\sqrt{\rho}} \cdot \max\left\{ \left(\frac{3\mathsf{n}(\mathsf{r}'+\mathsf{u})}{|\mathbb{F}|}\right), \left(\frac{|\mathcal{P}|+s+1}{|\mathbb{F}|}\right), \frac{\mathsf{n}\cdot\max\{u+1, \mathsf{d}_{\mathsf{max}}\}}{|\mathbb{K}\setminus \mathsf{D}|}, 2\eta\sqrt{\rho}\cdot\varepsilon_{\mathsf{FRI}} \right\},$$

where $\mathsf{n} = \max_{i\in[N]} \mathsf{n}_i$*,* D *is a subgroup of size* n*,* $\mathsf{r}' = \sum_{i\in[N]} \mathsf{r}'_i$*,* $\mathcal{P} = (\mathcal{P}_1, \dots, \mathcal{P}_N)$*,* $\mathsf{d}_{\mathsf{max}} = \max_{P\in\mathcal{P}} \deg(P)$*, u is a parameter used during the permutation argument of the STARKish IOP (analogous to the splitting parameters u_i in Section 3.1),* $s = \lceil \mathsf{r}'/\mathsf{u} \rceil$ *,* $\rho = (\mathsf{d}_{\mathsf{max}}+1)/\mathsf{n}$*, η is a parameter used in the batched FRI protocol (cf. [BGK+23]), and $\varepsilon_{\mathsf{FRI}}$ is the round-by-round soundness error of the batched FRI protocol used at the end of the STARKish IOP.*

Furthermore, the Succinct Non-interactive Argument of Knowledge obtained by compiling the $\Pi_{\mathsf{STARKPack}}$ *IOP with Merkle tree commitments and the Fiat-Shamir transform is knowledge sound with error*

$$Q\cdot\varepsilon_{\mathsf{ks}} + O(Q^2/2^\kappa),$$

where Q is the number of random oracle queries with κ-bits of output that an adversary is capable of making.

Proof. Let $\mathsf{P}^*_{\mathsf{PACK}}$ be a malicious prover for the StarkPack IOP (respectively, $\mathsf{P}^*_{\mathsf{PACK}}$ is a malicious prover for the SNARK compiled from the StarkPack IOP using Merkle trees and the FS transform, which we call compiled StarkPack). We build a prover P^*_0 for the STARKish IOP (resp. compiled STARKish) as follows. The prover P^*_0 only accepts Plonkish instances of the form $\mathsf{PL}_{\mathsf{PACK}}$. In its execution, it first recovers the packed instance PACK, and then it runs an extractor $\mathsf{Ext}_{\mathsf{PACK}}$ for the Starkpack IOP (resp. compiled IOP) on the instance PACK and the prover $\mathsf{P}^*_{\mathsf{PACK}}$. If the extractor outputs a valid witness $(\mathbb{w}_1, \dots, \mathbb{w}_N)$ for the instance PACK, then P^*_0 uses Lemma 3.3 to efficiently construct a valid witness $\mathbb{w}$ for $\mathsf{PL}_{\mathsf{PACK}}$. This shows that the knowledge error of the StarkPack IOP (resp. compiled SNARK) is at most the knowledge error of the STARKish IOP (resp. compiled IOP) for instances of the form $\mathsf{PL}_{\mathsf{PACK}}$. Now the corollary follows by applying the results from [BGK+23] in order to bound the knowledge error of the STARKish IOP (resp. compiled IOP) for an instance of the form $\mathsf{PL}_{\mathsf{PACK}}$. □

4 Modular split-and-pack

The idea of splitting is quite simple, in essence it is about breaking execution traces into smaller pieces. We describe any type of splitting first, and then move on to modular splitting. When interpolating and evaluating the relevant polynomials, the prover performs more number theoretic transforms but over smaller domains. As we will see this is cheaper overall.

The cost of committing to these polynomials is amortized by batching the evaluations at the same point in the same Merkle leaf. There will be less leaves for committing to witness polynomials, but they will be bigger. Computing the combined constraint for the splitting (the polynomial we have called quot) is comparable —if not cheaper— than when there is no splitting. Finally, the cost of FRI will be asymptotically comparable (and even a bit cheaper) to when there is no splitting. We expect that the splitting operation therefore leads to an improvement in the proof time. The trade-off is at the level of the proof size (as the Prover is forced to send more evaluations) and the Verifier time (as the Verifier now needs to check more evaluations and hash bigger leaves).

Formalizing the splitting operation can be cumbersome in its full generality. In this paper, we focus on a simpler and natural case where we split witnesses into chunks with the same column size, by selecting entire rows out of the original witness. Let $\mathsf{PL} = (\mathcal{P}, \mathcal{Q}, \sigma, \mathsf{PI}, \mathsf{r}, \mathsf{r}', \ell, \mathsf{n}, \mathbb{F}, \mathbb{K})$ be a Plonkish instance. Choose a divisor k of n. instead of considering a witness $\mathbb{w} \in \mathbb{F}^{\mathsf{nr}}$ for PL which we consider as a matrix whose (i, j)-th entry is $\mathbb{w}[(i-1)\mathsf{r} + j]$, we split it into vectors $\mathbb{w}^0, \dots, \mathbb{w}^{k-1} \in \mathbb{F}^{(\mathsf{n}/k)\times \mathsf{r}}$ by selecting rows out of $\mathbb{w}$. We consider these vectors as matrices in the same way. It should hold that for any $(i, j) \in [\mathsf{n}] \times [\mathsf{r}]$ there are unique $(s, i') \in \{0, \dots, k-1\} \times [\mathsf{n}/k]$ such that:

$$\mathbb{w}[(i-1)\mathsf{r} + j] = \mathbb{w}^s[(i'-1)\mathsf{r} + j] \tag{5}$$

We should still ask that the constraints in $\mathcal{P}$ hold when evaluated at any of the corresponding entries in the newly created vectors $\mathbb{w}^s$. More specifically, for any $(i, j) \in [\mathsf{n}] \times [\mathsf{r}]$, let $(s(i), i'(i))$ be the indices such that (5) holds, emphasizing the dependence on i. The condition:

$$\forall P \in \mathcal{P},\ \forall i \in [\mathsf{n}-1],\ P(\mathbb{w}[(i-1)\mathsf{r}+1], \dots, \mathbb{w}[i\mathsf{r}], \dots, \mathbb{w}[(i+1)\mathsf{r}], \mathsf{q}_1[i], \dots, \mathsf{q}_\ell[i]) = 0$$

must translate into the condition:

$$\begin{aligned}\forall P \in \mathcal{P},\ \forall i \in [\mathsf{n}-1],\ P(\mathbb{w}^{s(i-1)}[(i'(i-1)\mathsf{r}+1], \dots, \mathbb{w}^{s(i-1)}[i'(i-1)\mathsf{r}+\mathsf{r}],& \\ \dots, \mathbb{w}^{s(i)}[i'(i)\mathsf{r}+\mathsf{r}], \mathsf{q}_1[i], \dots, \mathsf{q}_\ell[i]) &= 0\end{aligned}$$

And a similar condition must hold for copy constraints. The way this translates into polynomial constraints is hard to systematize when the splitting is general. We will see that in some special cases, like when splitting over cosets of a multiplicative subgroup, the constraints can be expressed very naturally and efficiently. Note from the previous equation that the constraints may now involve much more than two consecutive rows of the same $\mathbb{w}^s$, and may even involve values from different vectors $\mathbb{w}^s$. For this reason, after we formally define what a splitting of a Plonkish instance is, we need to define a new *split Plonkish relation.* We do this next.

4.1 The split Plonkish relation

We can think informally of the $\mathbb{w}^s$ as buckets in which we place rows of the original witness $\mathbb{w}$ successively without replacement. Once we have chosen n/k rows of each $\mathbb{w}^s$, we are done. A natural way to formalize this is by using a permutation $\mathfrak{S} : \{0, \dots, \mathsf{n}-1\} \to \{0, \dots, \mathsf{n}-1\}$. The first[5] row of $\mathbb{w}^0$ will be the row with index $\mathfrak{S}^{-1}(0)$ in $\mathbb{w}$, and so on. We formalize this next.

Definition 4.1 (Splitting of a Plonkish instance)**.** Let $\mathsf{PL} = (\mathcal{P}, \mathcal{Q}, \sigma, \mathsf{PI}, \mathsf{r}, \mathsf{r}', \ell, \mathsf{n}, \mathbb{F}, \mathbb{K})$ be a Plonkish instance. A splitting of PL is the additional data of a strict divisor k of n, together with a permutation $\mathfrak{S} : \{0, \dots, \mathsf{n}-1\} \to \{0, \dots, \mathsf{n}-1\}$.

[5]Note that the entries in the first row of $\mathbb{w}$, that is, entries with the form $(1, j)$ are given by $\mathbb{w}[j]$. This means that rows are indexed from 0.

We also define the following auxiliary functions $\mathfrak{T} : \{0, \ldots, \mathsf{n}-1\} \to \{0, \ldots, k-1\}$ and $\gamma : \{0, \ldots, \mathsf{n}-1\} \to \{0, \ldots, \mathsf{n}/k - 1\}$ by the following relations:

$$\mathfrak{T}(i) = \left\lfloor \frac{k\mathfrak{S}(i)}{n} \right\rfloor$$
$$\gamma(i) = \mathfrak{S}(i) - \mathfrak{T}(i)\frac{n}{k}$$

Informally, $\mathfrak{T}(i)$ tells us the s such that row i in $\mathbb{w}$ will be sent to $\mathbb{w}^s$, and $\gamma(i)$ tells us the index that row will have in $\mathbb{w}^s$.

We now define the split Plonkish relation.

Definition 4.2 (Split Plonkish Relation)**.** A split Plonkish instance

$$\mathsf{PL}_{\mathsf{split}} = (\mathcal{P}, \mathcal{Q}, \sigma, \mathsf{PI}, \mathsf{r}, \mathsf{r}', \ell, \mathsf{n}, k, \mathfrak{S}, \mathbb{F}, \mathbb{K})$$

is a Plonkish instance with the additional data of a splitting. Recall from Definition 4.1 that this is given by a strict divisor k of n and a permutation $\mathfrak{S} : \{0, \ldots, \mathsf{n}-1\} \to \{0, \ldots, \mathsf{n}-1\}$.

A set of execution traces for $\mathsf{PL}_{\mathsf{split}}$ is a set of k vectors $\mathbb{w}^0, \ldots, \mathbb{w}^{k-1} \in \mathbb{F}^{(\mathsf{n}/k)\times\mathsf{r}}$. It will be helpful to consider these as matrices whose entry $(i, j) \in [\mathsf{n}/k] \times [\mathsf{r}]$ is given by $\mathbb{w}^s[(i-1)\mathsf{r}+j]$ for $s = 0, \ldots, k-1$. Such a set of traces is said to satisfy $\mathsf{PL}_{\mathsf{split}}$ if the following conditions hold:

1. (Circuit constraint satisfaction). For all $P \in \mathcal{P}$, and all $i \in [n-1]$,

   $$P(\mathbb{w}^{\mathfrak{T}(i-1)}[\gamma(i-1)\mathsf{r}+1], \ldots, \mathbb{w}^{\mathfrak{T}(i-1)}[\gamma(i-1)\mathsf{r}+\mathsf{r}], \ldots, \mathbb{w}^{\mathfrak{T}(i)}[\gamma(i)\mathsf{r}+\mathsf{r}], \mathsf{q}_1[i], \ldots, \mathsf{q}_\ell[i]) = 0$$

   where $\mathfrak{T}$ and γ are defined as in Definition 4.1.

2. (Copy constraint satisfaction). Let $S \subset [\mathsf{r}]$ be the subset of column indices subject to a copy constraint. By definition $|S| = \mathsf{r}'$. For all $(i, j) \in [\mathsf{n}] \times S$, let $(\zeta(i, j), \xi(i, j)) \in [\mathsf{n}] \times S$ be such that $\sigma((i-1)\mathsf{r}+j) = (\zeta(i, j)-1)\mathsf{r}+\xi(i, j)$. Then the following should hold for all $(i, j) \in [\mathsf{n}] \times S$:

   $$\mathbb{w}^{\mathfrak{T}(i-1)}[(i-1)\mathsf{r}+j] = \mathbb{w}^{\mathfrak{T}(\zeta(i,j)-1)}[(\zeta(i, j)-1)\mathsf{r}+\xi(i, j)]$$

We denote by $(\mathbb{w}^0, \ldots, \mathbb{w}^{k-1})^{\mathsf{PI}}$ the vector formed by the entries $\{\mathbb{w}^{\mathfrak{T}(i-1)}[\gamma(i-1)\mathsf{r}+j] \mid (i-1)\mathsf{r}+j \in \mathsf{PI}\}$. We call this vector the public part of $\mathbb{w}^0, \ldots, \mathbb{w}^{k-1}$. The split Plonkish relation is then defined as:

$$\mathbf{R}_{\mathbf{SPlonkish}} := \left\{ (\mathsf{PL}_{\mathsf{split}}, \mathbb{x}, \mathbb{w}^0, \ldots, \mathbb{w}^{k-1}) \;\middle|\; \begin{array}{l} \mathsf{PL}_{\mathsf{split}} = (\mathcal{P}, \mathcal{Q}, \sigma, \mathsf{PI}, \mathsf{r}, \mathsf{r}', \ell, \mathsf{n}, k, \mathfrak{S}, \mathbb{F}, \mathbb{K}), \\ \mathbb{x} \in \mathbb{F}^{|\mathsf{PI}|}, \\ \mathbb{x} = (\mathbb{w}^0, \ldots, \mathbb{w}^{k-1})^{\mathsf{PI}}, \\ \mathbb{w}^0, \ldots, \mathbb{w}^{k-1} \in \mathbb{F}^{(\mathsf{n}/k)\times\mathsf{r}} \text{ satisfy } \mathsf{PL}_{\mathsf{split}} \end{array} \right\}$$

Remark 4.3. We have explained what a witness for the split Plonkish relation is, and how it satisfies a split instance. However, we have not explained how to turn the conditions in Point 1 and 2 in Definition 4.2 into polynomial constraints. As we mentioned, for circuit constraint satisfaction this might depend on the particular splitting. There are cases in which it translates seamlessly into polynomial constraints (see the next Example 4.4). However, for copy constraints there is a natural way to express the permutation argument, which we explain next.

For simplicity, assume that all wires are routable (i.e. $\mathsf{r} = \mathsf{r}'$). We first want to find a permutation $\tilde{\sigma} : \{0, \ldots, k-1\} \times \{0, \ldots, n/k-1\} \times [\mathsf{r}] \to \{0, \ldots, k-1\} \times \{0, \ldots, n/k-1\} \times [\mathsf{r}]$ that expresses the action of σ on the $\mathbb{w}^s[(i'-1)\mathsf{r}+j']$. Let us temporarily use the notation

$E = \{0, \dots, k-1\} \times \{0, \dots, n/k - 1\} \times [\mathsf{r}]$ and $F = \{0, \dots, n-1\} \times [\mathsf{r}]$. The permutation $\tilde{\sigma}$ is given by the composition of the following three maps:

$$\begin{array}{ccc} E & \longrightarrow & F \\ (s, i', j') & \longmapsto & (\mathfrak{S}^{-1}(i' + s\frac{n}{k}), j') \end{array}$$

$$\begin{array}{ccc} F & \xrightarrow{\sigma} & F \\ (i, j) & \longmapsto & (\zeta(i+1, j) - 1, \xi(i+1, j)) \end{array}$$

$$\begin{array}{ccc} F & \longrightarrow & E \\ (i, j) & \longmapsto & (\tau(i), \gamma(i), j) \end{array}$$

where ζ, ξ are defined as in Definition 4.2. All in all, this means that for all $(s, i', j') \in E$:

$$\tilde{\sigma}(s, i', j') = (\tau(\zeta(\mathfrak{S}^{-1}(i' + s\frac{n}{k}) + 1, j') - 1), \gamma(\zeta(\mathfrak{S}^{-1}(i' + s\frac{n}{k}) + 1, j') - 1), \xi(\mathfrak{S}^{-1}(i' + s\frac{n}{k}) + 1, j'))$$

Define a vector $W \in \mathbb{F}^{\mathsf{nr}}$ by setting, for all $(s, i, j) \in \{0, \dots, k-1\} \times [\mathsf{n}/k] \times [\mathsf{r}]$:

$$W[(k(i-1) + s)\mathsf{r} + j] = \mathbb{w}^s[(i-1)\mathsf{r} + j]$$

The vector W is constructed by putting the matrices $\mathbb{w}^s$ side-by-side. We can make the permutation $\tilde{\sigma}$ act on $[\mathsf{nr}]$ in the same way we can make the permutation σ act on $[\mathsf{nr}]$. For all $(s, i, j) \in \{0, \dots, k-1\} \times [\mathsf{n}/k] \times [\mathsf{r}]$, if $(\bar{s}, \bar{i}, \bar{j}) = \tilde{\sigma}(s, i-1, j)$, then

$$\tilde{\sigma}((k(i-1) + s)\mathsf{r} + j) = (k\bar{i} + \bar{s})\mathsf{r} + \bar{j}$$

One checks that with these definitions, Point 2 in in Definition 4.2 corresponds precisely to the requirement that for all $i \in [\mathsf{nr}]$:

$$W_{\tilde{\sigma}(i)} = W_i$$

So now define the polynomials $\mathsf{S}^s_{ID_j}$ and $(\mathsf{S}^s_{\tilde{\sigma}})_j$ for all $s \in \{0, \dots, k-1\}$ and $j \in [\mathsf{r}]$ by the following:

$$\forall i \in \{0, \dots, \mathsf{n}/k - 1\},\ \mathsf{S}^s_{ID_j}((\mathsf{g}^k)^i) := (k(i-1) + s)\mathsf{r} + j$$
$$(\mathsf{S}^s_{\tilde{\sigma}})_j((\mathsf{g}^k)^i) := \tilde{\sigma}((k(i-1) + s)\mathsf{r} + j)$$

If we set a^s_j to be the polynomial such that for all $i \in \{0, \dots, \mathsf{n}/k - 1\}$, $\mathsf{a}^s_j((\mathsf{g}^k)^i) = \mathbb{w}^s[(i-1)\mathsf{r} + j]$, then in order to check that copy constraints hold it suffices to exhibit a polynomial $z(X) \in \mathbb{F}_{<\mathsf{n}/k}[X]$ such that:

$$z(\mathsf{g}^k X) \cdot f(X) = g(X) \cdot z(X)$$

where:

$$f(X) := \prod_{s=0}^{k-1} \prod_{j=1}^{\mathsf{r}} (\mathsf{a}^s_j(X) + \delta \cdot \mathsf{S}^s_{ID_j}(X) + \eta)$$
$$g(X) := \prod_{s=0}^{k-1} \prod_{j=1}^{\mathsf{r}} (\mathsf{a}^s_j(X) + \delta \cdot (\mathsf{S}^s_{\tilde{\sigma}})_j(X) + \eta)$$

and δ, η are randomly chosen by the verifier. This can also be split into constraints of smaller degree as we have seen previously. Note that the degree in the variable X of the polynomials f and g is smaller or equal to $\mathsf{r}(\mathsf{n} - k)$ (a product of $\mathsf{r}k$ polynomials of degree smaller or equal to $\mathsf{n}/k - 1$). However, the degree in δ and η (considered as formal variables) increases with respect to the permutation argument without splitting, which results in a small soundness loss.

The computation of $\tilde{\sigma}$ and the polynomials $\mathsf{S}^s_{ID_j}, (\mathsf{S}^s_{\tilde{\sigma}})_j$ will be done in the preprocessing phase.

Example 4.4 (Splitting along cosets modulo k)**.** We can perform a splitting by selecting rows depending on the residue of their index modulo k. Choose k a strict divisor of n, and set $\mathfrak{S}(i) = \lfloor i/k \rfloor + n/k \cdot (i \mod k)$. One checks that $\mathfrak{S}$ is a permutation, and that:

$$\mathfrak{T}(i) := \left\lfloor \frac{k\mathfrak{S}(i)}{n} \right\rfloor = (i \mod k)$$
$$\gamma(i) := \mathfrak{S}(i) - \mathfrak{T}(i)\frac{n}{k} = \lfloor i/k \rfloor$$

Here, the circuit constraints can be translated easily to polynomial constraints. Let $((\mathcal{P}, \mathcal{Q}, \sigma, \mathsf{PI}, \mathsf{r}, \mathsf{r}', \ell, \mathsf{n}, k, \mathfrak{S}, \mathbb{F}, \mathbb{K}), \mathbb{x})$ be a split Plonkish instance where $\mathfrak{S}$ is given by the permutation we defined above, and let $\mathbb{w}^0, \dots, \mathbb{w}^{k-1}$ be a set of execution traces for it. For simplicity focus on one circuit constraint P. We ask that for all $i \in [\mathsf{n}-1]$,

$$P(\mathbb{w}^{i-1\,[k]}[\lfloor (i-1)/k \rfloor \mathsf{r} + 1], \dots, \mathbb{w}^{i\,[k]}[\lfloor i/k \rfloor \mathsf{r} + \mathsf{r}], \mathsf{q}_1[i], \dots, \mathsf{q}_\ell[i]) = 0$$

where we have used the notation $i\,[k] = (i \mod k)$. We notice that the constraints always involve $\mathbb{w}^s$ and $\mathbb{w}^{s+1}$, unless $s = k-1$ in which case it involves $\mathbb{w}^{k-1}$ and $\mathbb{w}^0$. Further, the constraints always involve the same rows of $\mathbb{w}^s$ and $\mathbb{w}^{s+1}$ if $(i-1)\,[k] = s < k-1$, namely the row with index $\lfloor (i-1)/k \rfloor = \lfloor i/k \rfloor$. And if $(i-1)\,[k] = s = k-1$, the constraint involves row $\lfloor (i-1)/k \rfloor$ of $\mathbb{w}^{k-1}$ and row $\lfloor i/k \rfloor = \lfloor (i-1)/k \rfloor + 1$ of $\mathbb{w}^0$. This means that we can translate the circuit constraint into:

$$\begin{aligned}
&\forall s \in \{0, \dots, k-2\},\ \forall i \in \{0, \dots, \mathsf{n}/k - 1\},\\
&\qquad P(\mathbb{w}^s[i\mathsf{r}+1], \dots, \mathbb{w}^{s+1}[i\mathsf{r}+\mathsf{r}], \mathsf{q}_1[\mathfrak{S}^{-1}(i + n(s+1)/k)], \dots, \mathsf{q}_\ell[\mathfrak{S}^{-1}(i + n(s+1)/k)]) = 0\\
&\forall i \in \{0, \dots, \mathsf{n}/k - 2\},\\
&\qquad P(\mathbb{w}^{k-1}[i\mathsf{r}+1], \dots, \mathbb{w}^0[(i+1)\mathsf{r}+\mathsf{r}], \mathsf{q}_1[\mathfrak{S}^{-1}(i+1)], \dots, \mathsf{q}_\ell[\mathfrak{S}^{-1}(i+1)]) = 0
\end{aligned}$$

Set a^s_j to be the polynomials such that $\mathsf{a}^s_j((\mathsf{g}^k)^i) = \mathbb{w}^s[(i-1)\mathsf{r} + j]$ for all $(s, i, j) \in \{0, \dots, k-1\} \times \{0, \dots, \mathsf{n}/k - 1\} \times [\mathsf{r}]$. For $0 \le s < k-1$ and $1 \le l \le \ell$ set q^s_l to be the polynomial such that $\mathsf{q}^s_l((\mathsf{g}^k)^i) = \mathsf{q}_l[\mathfrak{S}^{-1}(i + n(s+1)/k)]$, and set $\mathsf{q}^{k-1}_l((\mathsf{g}^k)^i) = \mathsf{q}_l[\mathfrak{S}^{-1}(i+1)]$. Then in order to check the circuit constraint satisfaction, we may check that all polynomials:

$$\begin{aligned}
&P(\mathsf{a}^0_1(X), \dots, \mathsf{a}^0_\mathsf{r}(X), \dots, \mathsf{a}^1_\mathsf{r}(X), \mathsf{q}^0_1(X), \dots, \mathsf{q}^0_\ell(X))\\
&\vdots\\
&P(\mathsf{a}^{k-1}_1(X), \dots, \mathsf{a}^{k-1}_\mathsf{r}(X), \dots, \mathsf{a}^0_\mathsf{r}(\mathsf{g}^k X), \mathsf{q}^{k-1}_1(X), \dots, \mathsf{q}^{k-1}_\ell(X))
\end{aligned}$$

are divisible by the vanishing polynomial of $\mathsf{D}^k = \langle \mathsf{g}^k \rangle$. Note that the degree of these polynomial constraints is less than $\mathsf{n}\deg(P)/k - \mathsf{n}/k$. This is k times less than the degree of the circuit constraints had we not performed the splitting. This may look like it increases the soundness of the protocol, but the fact that we now have k times more constraints will balance out the gain from having smaller degree constraints. The newly created selector polynomials should also be computed during preprocessing.

It is clear that a similar phenomenon happens for any splitting which has the property that the circuit constraint involves the same pair of rows of the same k pairs of traces $w^0, \dots, w^{k-1}$. The splitting modulo k happens to be a splitting with this property.

4.2 An IOP for split-and-pack modulo k

In the previous section, we have seen that copy constraints over the trace evaluation domain D can always be expressed as a polynomial permutation argument over the domain D^k, provided that we perform the computation of the induced permutation $\tilde{\sigma}$ and the polynomials $S^s_{ID_j}, (S^s_{\tilde{\sigma}})_j$ correctly (see Remark 4.3). We have also seen that when splitting along cosets modulo k (see Example 4.4), circuit constraints over the trace evaluation domain D can be expressed efficiently as k polynomial constraints over the domain D^k, provided that we perform the computation of the selectors $\mathsf{q}_1^s, \dots, \mathsf{q}_\ell^s$ correctly. In this section, we present an IOP for proving the satisfiability of a witness to a split Plonkish instance where the splitting is along cosets modulo k, which we denote by Π_{split}^k. Of course, one can extrapolate the ideas to more general splittings.

We fix a split Plonkish instance $(\mathcal{P}, \mathcal{Q}, \sigma, \mathsf{PI}, \mathsf{r}, \mathsf{r}', \ell, \mathsf{n}, k, \mathfrak{S}, \mathbb{F}, \mathbb{K})$, where $\mathfrak{S}(i) = \lfloor i/k \rfloor + n/k \cdot (i \mod k)$. For simplicity, we assume that $\mathsf{r} = \mathsf{r}'$ (all wires are routable). The trace evaluation domain is denoted by $\mathsf{D} = \langle \mathsf{g} \rangle$, and the subgroup generated by $\mathsf{g}_k := \mathsf{g}^k$ is denoted by D^k. Its vanishing polynomial is denoted by $Z_k(X) := \prod_{x \in \mathsf{D}^k}(X - x) = \prod_{i=0}^{\mathsf{n}/k-1}(X - \mathsf{g}_k^i)$. We fix an evaluation domain $\mathsf{H} \subset \mathbb{K}^\times$, a multiplicative subgroup containing a non-trivial coset of D^k. When we say that the prover gives oracle access to a function, we mean that the prover interpolates a vector with entries in $\mathbb{F}$ over D^k ; and then provides oracle access to this interpolant over H. We fix a set of execution traces $\mathbb{w}^0, \dots, \mathbb{w}^{k-1} \in \mathbb{F}^{(\mathsf{n}/k)\mathsf{r}}$ and public part $\mathbb{x}$ for the split Plonkish instance. Finally, we fix auxiliary parameters $\mathsf{aux} = (\vec{t}, s)$ for the Batched-FRI protocol. The parameter $\vec{t} = (t_1, \dots, t_m) \in \mathbb{N}$ indicates that we will use t_i-to-1 maps in the i-th COMMIT phase of FRI, and s is the number of repetitions of the QUERY phase of FRI.

Preprocessing. The prover and verifier compute the induced permutation $\tilde{\sigma} : [\mathsf{nr}] \to [\mathsf{nr}]$ as explained in Remark 4.3. They compute the polynomials $S_{ID_j}^s, (S_{\tilde{\sigma}}^s)_j \in \mathbb{F}_{<\mathsf{n}/k}[X]$ for all $s \in \{0, \dots, k-1\}$ and $j \in [\mathsf{r}]$ as:

$$\forall i \in \{0, \dots, \mathsf{n}/k - 1\},\ \mathsf{S}_{ID_j}^s(\mathsf{g}_k^i) := (k(i-1) + s)\mathsf{r} + j$$
$$(\mathsf{S}_{\tilde{\sigma}}^s)_j(\mathsf{g}_k^i) := \tilde{\sigma}((k(i-1) + s)\mathsf{r} + j)$$

They also compute the selectors $\mathsf{q}_1^s, \dots, \mathsf{q}_\ell^s \in \mathbb{F}_{<\mathsf{n}/k}[X]$ for all $s \in \{0, \dots, k-1\}$ as:

$$\forall i \in \{0, \dots, \mathsf{n}/k - 1\},\ \mathsf{q}_l^s(\mathsf{g}_k^i) := \mathsf{q}_l[\mathfrak{S}^{-1}(i + n(s+1)/k)]$$
$$\mathsf{q}_l^{k-1}(\mathsf{g}_k^i) := \mathsf{q}_l[\mathfrak{S}^{-1}(i+1)]$$

Finally, they agree on the splitting parameter u for the permutation argument.

Round 1. The prover computes the polynomials $\mathsf{a}_j^s \in \mathbb{F}_{<\mathsf{n}/k}[X]$ such that $\mathsf{a}_j^s(\mathsf{g}_k^i) = \mathbb{w}^s[(i-1)\mathsf{r} + j]$ for all $(s, i, j) \in \{0, \dots, k-1\} \times \{0, \dots, \mathsf{n}/k - 1\} \times [\mathsf{r}]$. The prover gives oracle access to the wire polynomials $\mathsf{a}_j^s : \mathsf{H} \to \mathbb{K}$ to the verifier. The verifier samples uniform randomness $\delta, \eta \in \mathbb{K}$ to be used in the permutation argument.

Round 2. The prover sends oracle access to the permutation and partial product polynomials $z, (\pi_l)_{l \in [s-1]} \in \mathbb{F}_{<\mathsf{n}/k}[X]$, where $s = \lceil k\mathsf{r}/u \rceil$. We recall their definition quickly. First recall the definition of the f and g polynomials:

$$f(X) := \prod_{s=0}^{k-1} \prod_{j=1}^{\mathsf{r}} (a_j^s(X) + \delta \cdot S_{ID_j}^s(X) + \eta)$$
$$g(X) := \prod_{s=0}^{k-1} \prod_{j=1}^{\mathsf{r}} (a_j^s(X) + \delta \cdot (S_{\tilde{\sigma}}^s)_j(X) + \eta)$$

Then $z(\mathsf{g}_k) = 1$, and for all $i \in \{2, \dots, \mathsf{n}/k\}$:

$$z(\mathsf{g}_k^i) := \prod_{1 \le j < i} f(\mathsf{g}_k^j)/g(\mathsf{g}_k^j)$$

We can arrange each of the $k\mathsf{r}$ factors appearing in the definition of f and g above in lexicographic order with respect to the indices $(s, j) \in \{0, \ldots, k-1\} \times [\mathsf{r}]$. For each $l \in [s]$, one defines the polynomials $\overline{f_l}(\delta, \eta, X) = \overline{f_l}(X)$ (resp. $\overline{g_l}(\delta, \eta, X) = \overline{g_l}(X)$) as the product of the factors indexed from $(\mathsf{u}-1)l + 1$ to $\min(\mathsf{u}l, k\mathsf{r})$ appearing in the definition of f (resp. g), according to that lexicographic order. Then for all $i \in [\mathsf{n}/k]$:

$$\begin{aligned}\pi_1(\mathsf{g}_k^i) &:= z(\mathsf{g}_k^i)\overline{f_1}(\mathsf{g}_k^i)\overline{g_1}(\mathsf{g}_k^i)^{-1}\\ \pi_l(\mathsf{g}_k^i) &:= \pi_{l-1}(\mathsf{g}_k^i)\overline{f_l}(\mathsf{g}_k^i)\overline{g_l}(\mathsf{g}_k^i)^{-1}, \text{ for } l = 2, \ldots, s-1\end{aligned}$$

The verifier samples uniform randomness $\alpha \in \mathbb{K}$.

Round 3. The prover computes the polynomials $\mathsf{u}, \mathsf{d}, \mathsf{quot}$ as follows:

$$\begin{aligned}\mathsf{u}(X) := &\,(z(X)\overline{f_1}(X) - \pi_1(X)\overline{g_1}(X))\alpha\\ &+ \sum_{l=2}^{s}(\pi_{l-1}(X)\overline{f_l}(X) - \pi_l(X)\overline{g_l}(X))\alpha^l\\ &+ (\pi_{s-1}(X)\overline{f_s}(X) - z(\mathsf{g}_k X)\overline{g_s}(X))\alpha^{s+1}\\ &+ L_1(X)(z(X)-1)\alpha^{s+2}\end{aligned}$$

$$\begin{aligned}\mathsf{d}(X) := &\sum_{j\in[|\mathcal{P}|]}\sum_{l=0}^{k-2}\alpha^{(j-1)k+l}P_j(\mathsf{a}_1^l(X), \ldots, \mathsf{a}_\mathsf{r}^{l+1}(X), \mathsf{q}_1^l(X), \ldots, \mathsf{q}_\ell^l(X))\\ &+ \sum_{j\in[|\mathcal{P}|]}\alpha^{(j-1)k+k-1}P_j(\mathsf{a}_1^{k-1}(X), \ldots, \mathsf{a}_\mathsf{r}^0(\mathsf{g}_k X), \mathsf{q}_1^{k-1}(X), \ldots, \mathsf{q}_\ell^{k-1}(X))\\ &+ \alpha^{(|\mathcal{P}|k-1)}\mathsf{u}(X)\end{aligned}$$

$$\mathsf{quot}(X) := \mathsf{d}(X)/Z_k(X)$$

where L_1 is the first Lagrange basis polynomial with respect to the set $\{\mathsf{g}_k, \ldots, \mathsf{g}_k^{\mathsf{n}/k}\}$. Then the prover splits each $\mathsf{quot}(X)$ into degree n/k polynomials $\mathsf{quot}_1, \ldots, \mathsf{quot}_\nu$ such that $\mathsf{quot}(X) = \sum_l X^{\mathsf{n}\cdot(l-1)/k}\mathsf{quot}_l(X)$. The prover gives oracle access to the maps $\mathsf{quot}_1, \ldots, \mathsf{quot}_\nu : \mathsf{H} \to \mathbb{K}$. The verifier samples uniform randomness $\mathfrak{z} \in \mathbb{K} \setminus (\mathsf{H} \cup \mathsf{g}^{-1}\mathsf{H})$.

Round 4. The prover sends the evaluations

$$\mathsf{eval} = (\mathsf{a}_j^s(\mathfrak{z}), \mathsf{a}_j^0(\mathsf{g}_k\mathfrak{z}), \mathsf{quot}_l(\mathfrak{z}), \pi_m(\mathfrak{z}), z(\mathfrak{z}), z(\mathsf{g}_k\mathfrak{z}), \mathsf{q}_c^s(\mathfrak{z}), \mathsf{S}_{\mathsf{ID}_j}^s(\mathfrak{z}), (\mathsf{S}_{\tilde{\sigma}}^s)_j(\mathfrak{z}))$$

for all $s \in \{0, \ldots, k-1\}, j \in [\mathsf{r}], l \in [\nu], m \in [s-1], c \in [\ell]$. The verifier samples uniform FRI randomness $\xi \in \mathbb{K}$.

Batched-FRI. The prover and verifier engage in a Batched-FRI protocol with auxiliary parameters aux to check proximity to the Reed-Solomon code $\mathsf{RS}(\mathbb{K}, \mathsf{H}, \mathsf{n})$ of the function defined as follows. Consider the vector of functions defined as:

$$\left(\frac{\mathsf{a}_j^s(X) - \mathsf{a}_j^s(\mathfrak{z})}{X - \mathfrak{z}}, \frac{\mathsf{a}_j^0(X) - \mathsf{a}_j^0(\mathsf{g}_k\mathfrak{z})}{X - \mathsf{g}_k\mathfrak{z}}, \frac{\mathsf{quot}_l(X) - \mathsf{quot}_l(\mathfrak{z})}{X - \mathfrak{z}}, \ldots, \frac{(\mathsf{S}_{\tilde{\sigma}}^s)_j(X) - (\mathsf{S}_{\tilde{\sigma}}^s)_j(\mathfrak{z})}{X - \mathfrak{z}}\right) \tag{6}$$

for all $s \in \{0, \ldots, k-1\}, j \in [\mathsf{r}], l \in [\nu], m \in [s-1], c \in [\ell]$ (it contains all relevant quotients of functions whose evaluations appear in eval). For simplicity, relabel this vector as $\vec{F} = (F_1(X), F_2(X), \ldots, F_M(X))$ (for some integer M). The function we apply FRI to is the function $\sum_{i=1}^{M}\xi^{i-1}F_i(X)$.

Decision.

1. **Circuit and copy constraint check.** The verifier uses eval to verify that $\mathsf{d}(\mathfrak{z}) = \mathsf{quot}(\mathfrak{z})Z_k(\mathfrak{z})$. If this check fails, the verifier rejects.

2. **Batched FRI check.** If the prover fails the Batched-FRI protocol with auxiliary parameters aux, the verifier rejects.

   If all checks pass, the verifier accepts the proof.

Remark 4.5. Our remark about Merkle commitments still applies in this case, where we can pack the evaluations of all relevant polynomials at the same evaluation point at each commitment step.

Remark 4.6 (Adding $\mathfrak{plookup}$ support for splitting)**.** For the kind of splittings we have been considering, i.e., those where we split the execution trace by selecting entire rows, one can see that a lookup argument involving a column of the execution trace can be expressed equivalently as k lookup arguments involving the same column of a set of execution traces for a splitting. This is due to the fact that if $f, t \in \mathbb{F}^n$, and $f_0, \dots, f_{k-1}$ is a partition of f, then $\{f\} \subset \{t\} \Leftrightarrow (\{f_0\} \subset \{t\}) \wedge \cdots \wedge (\{f_{k-1}\} \subset \{t\})$.

This translates into polynomial constraints that we include in the d polynomial in Round 3. of the IOP. Even though there are k times as much polynomial constraints, their degree is divided at least by k (because all of the polynomials involved are of smaller degree except the t polynomial, but the vanishing polynomials Z of D^k are now of degree n/k).

This still holds for more general splittings, however in this case we might be forced to perform more than k permutation arguments.

4.3 Performance

Benchmarks. To evaluate the performance implications of splitting, we conducted benchmarks on the Winterfell library. Our analysis focuses on the effects of splitting the trace into more columns and fewer rows via the $mod\,k$ approach. Figures 9-16 illustrate the performance trends observed for a trace with an increasing number of columns. Both Prover time and RAM peak usage decrease upon applying splitting techniques. However, the proof size and Verifier time exhibit initial reductions before increasing linearly as the number of splits grows.

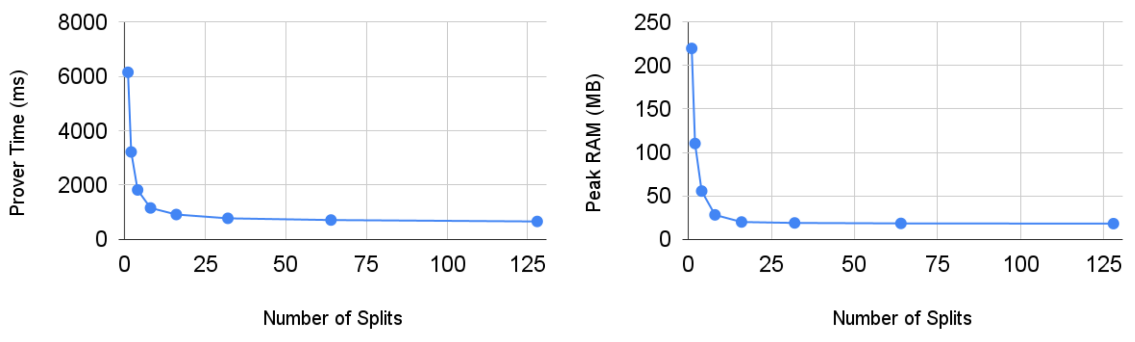


Figure 9: Prover time and Peak RAM splitting performance for 1 column

These findings suggest that splitting offers substantial gains in Prover time and RAM peak usage, particularly for scenarios involving a limited number of columns and a reasonable number of splittings. The improvement is less sharp whenever any of these two parameters is large.

The proof sizes and Verifier times are asymptotically linear in the number of chunks the original trace is split into. The Verifier time is still less than a few hundred ms at worst, however the proof gets large. This is especially true as the number of columns and

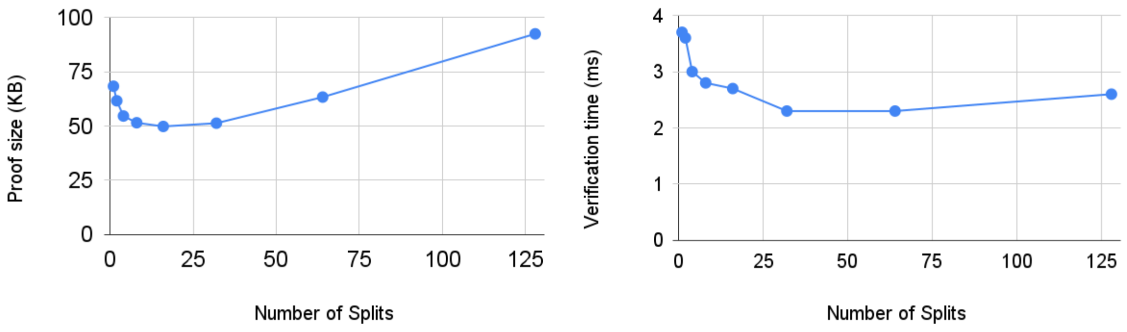


Figure 10: Proof size and Verifying time splitting performance for 1 column

split chunks gets large (the proof is around $10000kB$ for 275 columns and 64 chunks). This means that one should limit the splitting into a small number of chunks before the proof gets prohibitively large. However, because the Prover time decreases so sharply before attaining a sort of plateau, it means that we can get significant Prover time improvements without making the proof too large, or the verifier too slow.

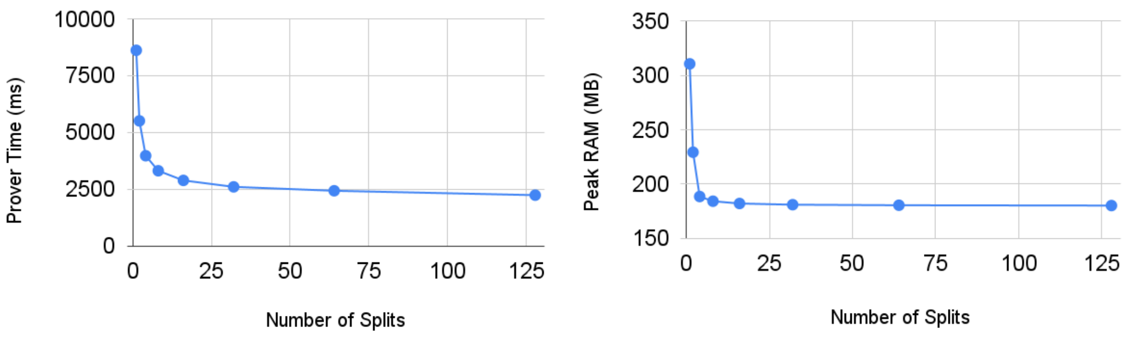


Figure 11: Prover time and Peak RAM splitting performance for 10 column

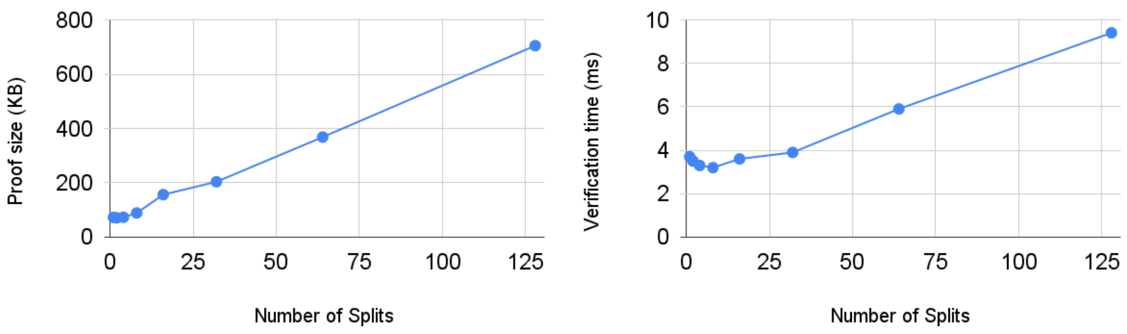


Figure 12: Verifier time and Proof size splitting performance for 10 columns

4.4 Security analysis

In this section we obtain knowledge soundness results for the IOP Π^k_{split}. Following a strategy we employed when proving knowledge soundness of the STARKPack IOP, we show that witnesses to a splitting of a Plonkish instance are in 1-to-1 correspondende with witnesses to the Plonkish instance. Because of this general correspondence, we have no doubt that these ideas extend to IOPs for more general splittings.

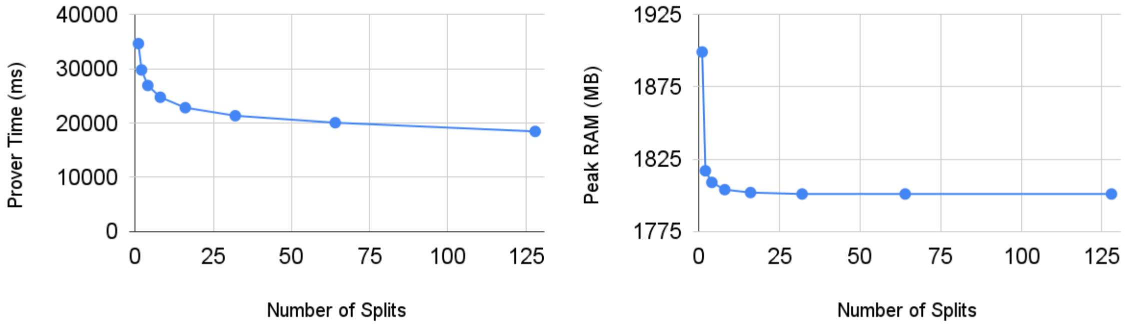


Figure 13: Prover time and Peak RAM splitting performance

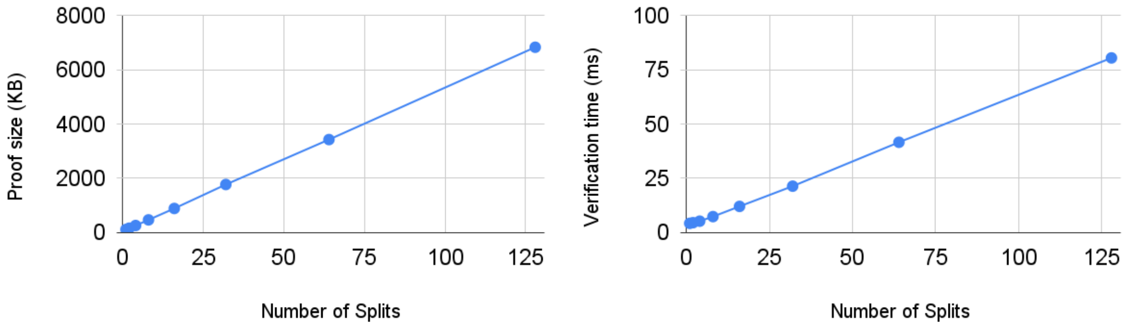


Figure 14: Verifier time and Proof size splitting performance for 100 columns

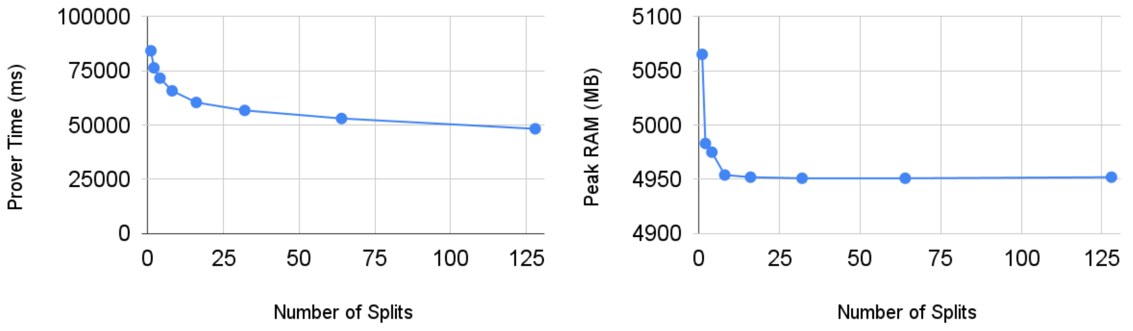


Figure 15: Prover time and Peak RAM splitting performance

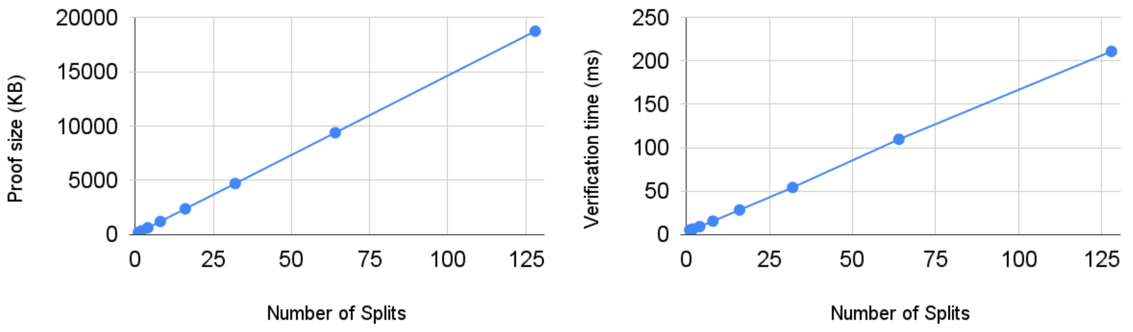


Figure 16: Verifier time and Proof size splitting performance for 275 columns

Lemma 4.7. *Let* $\mathsf{PL}_{\mathsf{split}} = (\mathcal{P}, \mathcal{Q}, \sigma, \mathsf{PI}, \mathsf{r}, \mathsf{r}', \ell, \mathsf{n}, k, \mathfrak{S}, \mathbb{F}, \mathbb{K})$ *be a split Plonkish instance. Let* $\mathsf{PL} = (\mathcal{P}, \mathcal{Q}, \sigma, \mathsf{PI}, \mathsf{r}, \mathsf{r}', \ell, \mathsf{n}, \mathbb{F}, \mathbb{K})$ *be the corresponding Plonkish instance. Then there is a bijective (polynomially computable) correspondence between satisfying execution traces* $\mathbb{w} \in$

$\mathbb{F}^{\mathsf{nr}}$ *for* PL*, and satisfying sets of execution traces* $\mathtt{w}^0, \ldots, \mathtt{w}^{k-1} \in \mathbb{F}^{(\mathsf{n}/k)\mathsf{r}}$ *for* $\mathsf{PL}_{\mathsf{split}}$.

Proof. This is almost by construction. Given a satisfying execution trace $\mathtt{w} \in \mathbb{F}^{\mathsf{nr}}$ for PL, set for all $(i, j) \in \{0, \ldots, \mathsf{n} - 1\} \times [\mathsf{r}]$:

$$\mathtt{w}[i\mathsf{r} + j] = \mathtt{w}^{\mathfrak{T}(i)}[\gamma(i)\mathsf{r} + j]$$

Recall the definition of $\mathfrak{T}, \gamma$ from Definition 4.1. This gives us k vectors $\mathtt{w}^0, \ldots, \mathtt{w}^{k-1} \in \mathbb{F}^{(\mathsf{n}/k)\mathsf{r}}$. The assumption that $\mathtt{w}$ is satisfying means that:

$$\forall i \in [n-1],\ \forall P \in \mathcal{P},\ P(\mathtt{w}[(i-1)\mathsf{r}+1], \ldots, \mathtt{w}[i\mathsf{r}+\mathsf{r}], \mathsf{q}_1[i], \ldots, \mathsf{q}_\ell[i]) = 0$$

and for all $i \in [\mathsf{nr}]$, $\mathtt{w}_i = \mathtt{w}_{\sigma(i)}$. The former condition translates to:

$$\forall i \in [n-1],\ \forall P \in \mathcal{P},\ P(\mathtt{w}^{\mathfrak{T}(i-1)}[\gamma(i-1)\mathsf{r}+1], \ldots, \mathtt{w}^{\mathfrak{T}(i)}[\gamma(i)\mathsf{r}+\mathsf{r}], \mathsf{q}_1[i], \ldots, \mathsf{q}_\ell[i]) = 0$$

We set again $(\zeta(i,j), \xi(i,j)) \in [\mathsf{n}] \times S$ to be such that $\sigma((i-1)\mathsf{r}+j) = (\zeta(i,j)-1)\mathsf{r}+\xi(i,j)$ for all $(i,j) \in [\mathsf{n}] \times [\mathsf{r}]$. The latter condition becomes $\mathtt{w}[(i-1)\mathsf{r}+j] = \mathtt{w}[(\zeta(i,j)-1)\mathsf{r}+\xi(i,j)]$ for all i, j. In turn, this becomes:

$$\forall (i,j) \in [\mathsf{n}] \times [\mathsf{r}],\ \mathtt{w}^{\mathfrak{T}(i-1)}[\gamma(i-1)\mathsf{r}+j] = \mathtt{w}^{\mathfrak{T}(\zeta(i,j)-1)}[\gamma(\zeta(i,j)-1)\mathsf{r}+\xi(i,j)]$$

So that $\mathtt{w}^0, \ldots, \mathtt{w}^{k-1}$ are satisfying for $\mathsf{PL}_{\mathsf{split}}$.

Conversely, let $\mathtt{w}^0, \ldots, \mathtt{w}^{k-1} \in \mathbb{F}^{(\mathsf{n}/k)\mathsf{r}}$ be satisfying for $\mathsf{PL}_{\mathsf{split}}$. For all $(s, i, j) \in \{0, \ldots, k-1\} \times \{0, \ldots, \mathsf{n}/k - 1\} \times [\mathsf{r}]$, let

$$\mathtt{w}^s[i\mathsf{r}+j] = \mathtt{w}[\mathfrak{S}^{-1}(i + s\mathsf{n}/k)\mathsf{r} + j]$$

This gives a vector $\mathtt{w} \in \mathbb{F}^{\mathsf{nr}}$. The relation $\gamma(i) = \mathfrak{S}(i) - \mathsf{n}\mathfrak{T}(i)/k$ implies that $\mathfrak{S}^{-1}(\gamma(i) + \mathsf{n}\mathfrak{T}(i)/k) = i$. Therefore, the circuit constraint satisfaction conditions

$$\forall i \in [n-1],\ \forall P \in \mathcal{P},\ P(\mathtt{w}^{\mathfrak{T}(i-1)}[\gamma(i-1)\mathsf{r}+1], \ldots, \mathtt{w}^{\mathfrak{T}(i)}[\gamma(i)\mathsf{r}+\mathsf{r}], \mathsf{q}_1[i], \ldots, \mathsf{q}_\ell[i]) = 0$$

immediately give:

$$\forall i \in [n-1],\ \forall P \in \mathcal{P},\ P(\mathtt{w}[(i-1)\mathsf{r}+1], \ldots, \mathtt{w}[i\mathsf{r}+\mathsf{r}], \mathsf{q}_1[i], \ldots, \mathsf{q}_\ell[i]) = 0$$

It is clear by construction that the copy constraints on $\mathtt{w}^0, \ldots, \mathtt{w}^{k-1}$ are equivalent to the condition that $\mathtt{w}_i = \mathtt{w}_{\sigma(i)}$, for all $i \in [\mathsf{nr}]$. $\square$

Corollary 4.8. *The IOP for split-and-pack modulo k from Section 4.2 has knowledge soundness error, given an instance* $\mathbf{R}_{\mathbf{SPlonkish}} = (\mathcal{P}, \mathcal{Q}, \sigma, \mathsf{PI}, \mathsf{r}, \mathsf{r}', \ell, \mathsf{n}, k, \mathfrak{S}, \mathbb{F}, \mathbb{K})$,

$$\varepsilon_{\mathsf{ks}} = \frac{1}{2\eta\sqrt{\rho}} \cdot \max\left\{ \left(\frac{3\mathsf{n}(\mathsf{r}'+\mathsf{u})}{|\mathbb{F}|} \right), \left(\frac{|\mathcal{P}|+s+1}{|\mathbb{F}|} \right), \frac{\mathsf{n} \cdot \max\{u+1, \mathsf{d}_{\mathsf{max}}\}}{|\mathbb{K} \setminus \mathsf{D}|}, 2\eta\sqrt{\rho} \cdot \varepsilon_{\mathsf{FRI}} \right\},$$

where D *is a subgroup of size* n*, u is a parameter used during the permutation argument of the STARKish IOP (analogous to the splitting parameters u_i in Section 3.1), $s = \lceil \mathsf{r}'/\mathsf{u} \rceil$, $\rho = (\mathsf{d}_{\mathsf{max}}+1)/\mathsf{n}$, η is a parameter used in the batched FRI protocol (cf. [BGK+23]), and $\varepsilon_{\mathsf{FRI}}$ is the round-by-round soundness error of the batched FRI protocol used at the end of the STARKish IOP.*

Furthermore, the Succinct Non-interactive Argument of Knowledge obtained by compiling the IOP with Merkle tree commitments and the Fiat-Shamir transform is knowledge sound with error

$$Q \cdot \varepsilon_{\mathsf{ks}} + O(Q^2/2^\kappa),$$

where Q is the number of random oracle queries with κ-bits of output that an adversary is capable of making.

Proof. The proof is analogous to that of Corollary 3.4, using Lemma 4.7 instead of Lemma 3.3. $\square$

5 References

[BC23] Benedikt Bünz and Binyi Chen. Protostar: Generic efficient accumulation/folding for special sound protocols. Cryptology ePrint Archive, Paper 2023/620, 2023. https://eprint.iacr.org/2023/620.

[BCMS20] Benedikt Bünz, Alessandro Chiesa, Pratyush Mishra, and Nicholas Spooner. Proof-carrying data from accumulation schemes. Cryptology ePrint Archive, Paper 2020/499, 2020. https://eprint.iacr.org/2020/499.

[BGH19] Sean Bowe, Jack Grigg, and Daira Hopwood. Recursive proof composition without a trusted setup. Cryptology ePrint Archive, Paper 2019/1021, 2019. https://eprint.iacr.org/2019/1021.

[BGK$^+$23] Alexander R. Block, Albert Garreta, Jonathan Katz, Justin Thaler, Pratyush Ranjan Tiwari, and Michał Zając. Fiat-shamir security of fri and related snarks. Cryptology ePrint Archive, Paper 2023/1071, 2023. https://eprint.iacr.org/2023/1071.

[BGT23] Jeremy Bruestle, Paul Gafni, and RISC Zero Team. Risc zero zkvm: Scalable, transparent arguments of risc-v integrity, 2023. https://dev.risczero.com/proof-system-in-detail.pdf.

[BGTZ23] Alexander R. Block, Albert Garreta, Pratyush Ranjan Tiwari, and Michał Zając. On soundness notions for interactive oracle proofs. Cryptology ePrint Archive, Paper 2023/1256, 2023. https://eprint.iacr.org/2023/1256.

[BMNW24] Benedikt Bünz, Pratyush Mishra, Wilson Nguyen, and William Wang. Accumulation without homomorphism. Cryptology ePrint Archive, Paper 2024/474, 2024. https://eprint.iacr.org/2024/474.

[EG23] Liam Eagen and Ariel Gabizon. Protogalaxy: Efficient protostar-style folding of multiple instances. Cryptology ePrint Archive, Paper 2023/1106, 2023. https://eprint.iacr.org/2023/1106.

[Gab] Ariel Gabizon. Multiset checks in plonk and plookup. https://hackmd.io/@arielg/ByFgSDA7D. Accessed: 2023-11-27.

[Gaf] Paul Gafni. Zk10: Fri-based recursion with a groth16 wrapper. https://www.youtube.com/watch?v=wkIBN2CGJdc&list=PLj80z0cJm8QFnY6VLVa84nr-21DNvjWH7&index=17&ab_channel=ZeroKnowledge. Accessed: 2024-3-20.

[Gro16] Jens Groth. On the size of pairing-based non-interactive arguments. Cryptology ePrint Archive, Paper 2016/260, 2016. https://eprint.iacr.org/2016/260.

[GW20] Ariel Gabizon and Zachary J. Williamson. plookup: A simplified polynomial protocol for lookup tables. Cryptology ePrint Archive, Paper 2020/315, 2020. https://eprint.iacr.org/2020/315.

[GWC19] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Paper 2019/953, 2019. https://eprint.iacr.org/2019/953.

[KGL$^+$20] Irakliy Khaburzaniya, François Garillot, Kevin Lewi, Konstantinos Chalkias, and Jasleen Malvai. Winterfell, 2020. https://github.com/facebook/winterfell.

[KS22] Abhiram Kothapalli and Srinath Setty. Supernova: Proving universal machine executions without universal circuits. Cryptology ePrint Archive, Paper 2022/1758, 2022. https://eprint.iacr.org/2022/1758.

[KS23] Abhiram Kothapalli and Srinath Setty. Hypernova: Recursive arguments for customizable constraint systems. Cryptology ePrint Archive, Paper 2023/573, 2023. https://eprint.iacr.org/2023/573.

[KST21] Abhiram Kothapalli, Srinath Setty, and Ioanna Tzialla. Nova: Recursive zero-knowledge arguments from folding schemes. Cryptology ePrint Archive, Paper 2021/370, 2021. https://eprint.iacr.org/2021/370.

[Lab23] Modulus Labs. The cost of intelligence: Proving machine learning inference with zero-knowledge, 2023. https://drive.google.com/file/d/1tylpowpaqcOhKQtYolPlqvx6R2Gv4IzE/view.

[MAGABMMT23] Héctor Masip-Ardevol, Marc Guzmán-Albiol, Jordi Baylina-Melé, and Jose Luis Muñoz-Tapia. estark: Extending starks with arguments. Cryptology ePrint Archive, Paper 2023/474, 2023. https://eprint.iacr.org/2023/474.

[Sta21] StarkWare. ethSTARK documentation. Cryptology ePrint Archive, Paper 2021/582, 2021. https://eprint.iacr.org/2021/582.

[Tea22] PolygonZero Team. Plonky2: Fast recursive arguments with plonk and fri, 2022. https://github.com/0xPolygonZero/plonky2/blob/main/plonky2/plonky2.pdf.

[ZGGX23] Tianyu Zheng, Shang Gao, Yu Guo, and Bin Xiao. Kilonova: Non-uniform pcd with zero-knowledge property from generic folding schemes. Cryptology ePrint Archive, Paper 2023/1579, 2023. https://eprint.iacr.org/2023/1579.

[zkS23] zkSync. Boojum upgrade: zkSync era's new high-performance proof system for radical decentralization, 2023. Accessed: 2023-7-17.

A Detailed benchmarks

A.1 Packed prover times

Table 3: Comparison of Packed and Non-packed(Winterfell) Prover Times with 10 columns traces

Number of Traces	Packed Prover Time (s)	Non-packed Prover Time (s)	Prover Time Ratio
1	1.0173	1.0173	1
2	1.6527	2.0346	1.23107642
3	2.3065	3.0519	1.32317364
4	2.9295	4.0692	1.389042499
5	3.5535	5.0865	1.431405656
6	4.2125	6.1038	1.448973294
7	4.8912	7.1211	1.455900393
8	5.5078	8.1384	1.477613566
9	6.1971	9.1557	1.477416856
10	6.7718	10.173	1.50225937

Table 4: Comparison of Packed and Non-packed(Winterfell) Prover Times with 100 columns traces

Number of Traces	Packed Prover Time (s)	Non-packed Prover Time (s)	Prover Time Ratio
1	5.0106	5.0106	1
2	9.1251	10.0212	1.098201664
3	13.141	15.0318	1.143885549
4	17.242	20.0424	1.162417353
5	21.843	25.053	1.146957835
6	25.453	30.0636	1.181141712
7	29.344	35.0742	1.195276718
8	33.329	40.0848	1.202700351
9	37.469	45.0954	1.203538926
10	42.099	50.106	1.190194541

Table 5: Comparison of Packed and Non-packed(Winterfell) Prover Times with 275 columns traces

Number of Traces	Packed Prover Time (s)	Non-packed Prover Time (s)	Prover Time Ratio
1	11.568	11.568	1
2	22.292	23.136	1.037861116
3	33.271	34.704	1.043070542
4	44.044	46.272	1.050585778
5	54.995	57.84	1.051731976
6	66.506	69.408	1.043635161
7	76.914	80.976	1.052812232
8	88.239	92.544	1.048787951
9	99.898	104.112	1.042183027

Table 6: Comparison of Packed and Non-packed(RISC Zero) Prover Times

Number of Traces	Packed Prover Time (s)	Non-packed Prover Time (s)	Prover Time Ratio
1	4.0001	4.0001	1
2	8.2615	8.0002	0.9683713611
3	12.404	12.0003	0.9674540471
4	16.621	16.0004	0.962661693
5	21.523	20.0005	0.92926172
6	25.012	24.0006	0.9595634096
7	29.996	28.0007	0.9334811308
8	34.47	32.0008	0.9283666957
9	39.646	36.0009	0.9080588206
10	44.005	40.001	0.9090103397

A.2 Packed verifier times

Table 7: Comparison of Packed and Non-packed(Winterfell) Verifier Times(ms) with 10 columns traces

No. Traces	Packed Verifier Time	Non-packed Verifier Time	Verifier Time Ratio
1	0.37661	0.37661	1
2	0.40327	0.75322	1.867780891
3	0.43987	1.12983	2.568554346
4	0.4445	1.50644	3.389066367
5	0.4449	1.88305	4.232524163
6	0.45203	2.25966	4.998916001
7	0.46565	2.63627	5.661483947
8	0.48044	3.01288	6.271084839
9	0.49884	3.38949	6.794743806
10	0.52587	3.7661	7.161655923

Table 8: Comparison of Packed and Non-packed(Winterfell) Verifier Times(ms) with 100 columns traces

Number of Traces	Packed Verifier Time	Non-packed Verifier Time	Verifier Time Ratio
1	0.46116	0.46116	1
2	0.54923	0.92232	1.67929647
3	0.60912	1.38348	2.271276596
4	0.68608	1.84464	2.688666045
5	0.7959	2.3058	2.897097625
6	0.84213	2.76696	3.285668483
7	0.92047	3.22812	3.50703445
8	0.98587	3.68928	3.742156674
9	1.0649	4.15044	3.897492722
10	1.154	4.6116	3.996187175

Table 9: Comparison of Packed and Non-packed(Winterfell) Verifier Times(ms) with 275 columns traces

Number of Traces	Packed Verifier Time	Non-packed Verifier Time	Verifier Time Ratio
1	0.5633	0.5633	1
2	0.75701	1.1266	1.488223405
3	0.95869	1.6899	1.762717875
4	1.145	2.2532	1.967860262
5	1.3456	2.8165	2.093118312
6	1.5293	3.3798	2.210030733
7	1.7129	3.9431	2.302002452
8	1.896	4.5064	2.376793249
9	2.0925	5.0697	2.422795699

Table 10: Comparison of Packed and Non-packed(RISC Zero) Verifier Times(ms)

Number of Traces	Packed Verifier Time	Non-packed Verifier Time	Verifier Time Ratio
1	1.8774	1.8774	1
2	2.6476	3.7548	1.418190059
3	3.3735	5.6322	1.669542019
4	4.1306	7.5096	1.818040963
5	4.8293	9.387	1.943759965
6	5.5719	11.2644	2.021644322
7	6.3896	13.1418	2.056748466
8	7.2754	15.0192	2.06438134
9	8.0432	16.8966	2.100731052
10	8.6492	18.774	2.170605374

A.3 Packed proof size

Table 11: Comparison of Packed and Non-packed(Winterfell) Proof Sizes in KB for 10 columns traces

Number of Traces	Packed Proof Size	Non-packed Proof Size	Proof Size Ratio
1	76	76	1
2	81.9	152	1.855921856
3	87.9	228	2.593856655
4	94.3	304	3.223753977
5	99.6	380	3.815261044
6	103.9	456	4.388835419
7	108.3	532	4.912280702
8	114.5	608	5.310043668
9	120.6	684	5.671641791
10	126.4	760	6.012658228

Table 12: Comparison of Packed and Non-packed(Winterfell) Proof Sizes in KB for 100 columns traces

Number of Traces	**Packed Proof Size**	**Non-packed Proof Size**	**Proof Size Ratio**
1	124.5	124.5	1
2	179	249	1.391061453
3	230.5	373.5	1.620390456
4	285.8	498	1.742477257
5	340.2	622.5	1.829805996
6	392.3	747	1.904154983
7	445	871.5	1.958426966
8	496.8	996	2.004830918
9	552.1	1120.5	2.029523637
10	604.6	1245	2.059212703

Table 13: Comparison of Packed and Non-packed(Winterfell) Proof Sizes in KB for 275 columns traces

Number of Traces	**Packed Proof Size**	**Non-packed Proof Size**	**Proof Size Ratio**
1	217	217	1
2	362.5	434	1.197241379
3	510.5	651	1.275220372
4	658.6	868	1.317947161
5	804.6	1085	1.348496147
6	947.4	1302	1.374287524
7	1095.8	1519	1.386201862
8	1242	1736	1.397745572
9	1387.9	1953	1.407161899

Table 14: Comparison of Packed and Non-packed(RISC Zero) Proof Sizes in KB

Number of Traces	**Packed Proof Size**	**Non-packed Proof Size**	**Proof Size Ratio**
1	210.223	210.223	1
2	278.941	420.445	1.507
3	347.660	630.668	1.814
4	416.378	840.891	2.020
5	485.098	1051.113	2.167
6	553.816	1261.336	2.278
7	622.535	1471.559	2.364
8	691.254	1681.781	2.433
9	759.973	1892.004	2.490
10	828.691	2102.227	2.537

A.4 Splitting in Winterfell

Table 15: Splitting benchmarks for a trace of 2^{16} rows with 1 column.

Number of Splits	Prover Time (ms)	Proof Size (KB)	Verification Time (ms)
1	6147	68.3	3.7
2	3219	61.6	3.6
4	1823	54.6	3
8	1160	51.5	2.8
16	919	49.8	2.7
32	780	51.3	2.3
64	717	63.3	2.3
128	665	92.4	2.6

Table 16: Splitting benchmarks for a trace of 2^{16} rows with 10 columns.

Number of Splits	Prover Time (ms)	Proof Size (KB)	Verification Time (ms)
1	8618	72.8	3.7
2	5514	71.1	3.5
4	3981	73.3	3.3
8	3320	89.1	3.2
16	2896	156.6	3.6
32	2612	203.9	3.9
64	2440	368.4	5.9
128	2247	704.8	9.4

Table 17: Splitting benchmarks for a trace of 2^{16} rows with 100 columns.

Number of Splits	Prover Time (ms)	Proof Size (KB)	Verification Time (ms)
1	34605	118.5	4.3
2	29766	166	4.6
4	26906	264.3	5.3
8	24738	471.5	7.4
16	22816	891.6	12.1
32	21337	1773.3	21.4
64	20055	3427.3	41.6
128	18451	6825	80.4

Table 18: Splitting benchmarks for a trace of 2^{16} rows with 275 columns.

Number of Splits	Prover Time (ms)	Proof Size (KB)	Verification Time (ms)
1	84143	212.2	5.4
2	76286	353.2	6.6
4	71546	636.2	9.3
8	65711	1215.1	15.6
16	60372	2378.6	28.4
32	56712	4708.8	54.3
64	52987	9379.5	109.7
128	48255	18724.3	210.8