A Unified Framework for Contract-Validated Benchmarking of Zero-Knowledge Proving Systems

Matěj Hůlek*, Martin Perešíni*† and Ivan Homoliak*‡
*Brno University of Technology, Faculty of Information Technology, Brno, Czech Republic
†Communication Systems Group CSG, Department of Informatics IfI, University of Zürich UZH, Zürich, Switzerland
‡Slovak University of Technology, Faculty of Informatics and Information Technologies, Bratislava, Slovak Republic

***Abstract*—Zero-knowledge proofs are an operational infrastructure for rollups, private payments, verifiable execution, and zkVM applications. We treat them as succinct proofs of validity whose privacy properties depend on the system and the mode of use. Published benchmarks often compare different workloads, proof modes, and hardware paths, which makes their results hard to interpret. This work presents a unified host-agent framework for benchmarking heterogeneous proving systems under a common execution and reporting method. Each prover runs in an isolated containerized agent and is selected through a scheme-adapter layer. The agent reports canonical JSON artifacts linked to a hashed run manifest and workload-specific semantic contracts. The host checks the public semantic fields and the agent's verification flag to ensure backends are compared on the same declared public claim, even when their circuits, traces, or guest pipelines differ. We evaluate circuit-based, transparent, GPU-accelerated, folding, and zkVM systems: Groth16, ICICLE, Halo2, Plonky3, Winterfell, Nova, RISC Zero with CUDA and Groth16 wrapping, and SP1 with and without AVX. The suite uses six primary workloads with comparable semantics, together with secondary hash workloads that expose specialization. No prover family dominates all deployment criteria. At 8 CPU cores, Plonky3 and Winterfell have the smallest across-case proving-time and RAM figures. Groth16 and ICICLE produce sub-kilobyte proofs. Winterfell has the lowest host verification time. zkVM systems trade larger proofs and higher memory for programmability. On the Merkle-path micro-benchmark, pairing artifacts remain sub-kilobyte, whereas wrapping a zkVM receipt yields a Groth16-sized artifact at a high proving cost. GPU acceleration helps most after fixed setup costs are amortized. The main contribution is methodological: validate the statement being proved, disclose the security profile, and report resource limits in addition to proving time.**

***Index Terms*—Zero-knowledge proofs, blockchain, Ethereum, Rollups, zkVM, zk-SNARK, zk-STARK, GPU acceleration, Benchmarking, Verifiable computation**

I. Introduction

Zero-knowledge proof (ZKP) systems now sit on the critical path of blockchain infrastructure. Ethereum rollups rely on succinct verification to move computation off Layer 1. Privacy and DeFi applications use ZKPs to hide witnesses while they keep public auditability. zkVM and zkEVM systems promise a programming model closer to ordinary software execution [1], [2]. The result is a systems problem. A proof system is no longer judged only by asymptotic complexity or by the elegance of its security argument. A deployment must generate a proof within a slot, verify it on-chain or in a light client, remain within available RAM and VRAM, and maintain an acceptable trust model. Choosing a proof system is hard for both cryptographic and operational reasons. Pairing-based zk-SNARKs (Groth16) provide compact proofs and low-cost verification but require a trusted setup [3]. Plonkish systems trade that for arithmetization- and backend-dependent performance [4]. STARK-like systems avoid a trusted setup through hash-based assumptions, at the cost of larger proofs [5]. zkVMs move circuit complexity off developers but add guest-execution, recursion, and wrapping overheads [6]–[8]. GPU provers are CPU-bound at small sizes and become accelerator-bound once work is batched [9]–[11]. Published benchmarks are meant to help, but many published numbers are hard to compare. Studies mix native circuits with zkVM execution [1], [12] or hand-written circuits with optimized gadgets [13], [14]. Some report end-to-end wall time, others only the proving kernel [9], [10]. Many differ in security profiles, curves, or proof modes under the same framework name [15]–[17]. Even when names match, the statement being proved can differ in padding, arithmetic model, or input binding. A single "lowest-latency" ranking then hides *why* a framework is fast and *when* that speed matters for a network.

This work treats ZKP benchmarking as a reproducible systems problem. We design and evaluate a *host-agent* framework in which each proving system is isolated in a containerized agent, selected through a common adapter interface, and required to return canonical results linked to an explicit run manifest. The suite separates *primary* workloads, implemented with comparable semantics across frameworks, from *secondary* workloads that expose specialization such as hash gadgets or GPU paths. The goal is not to force heterogeneous systems into identical internal representations. It is to make the comparison *level* explicit: systems prove the same contract-level claim, while their security profile, execution mode, hardware path, and caveats are reported beside the numbers.

Research questions. We ask three practical questions. **(RQ1)** What performance and artifact-size trade-offs emerge when representative ZKP systems are measured with the same host, workload contracts, and result schema? **(RQ2)** How much do CPU scaling and GPU acceleration change the interpretation of prover performance? **(RQ3)** Which systems fit verifier-constrained (on-chain or light-client), prover-constrained, transparent, memory-constrained, and general-purpose-program deployments, under a common run policy

and a disclosed measurement budget?

Contributions. (1) A *contract-validated* method: each workload is fixed by a host-side output contract so public fields are comparable across backends. **(2)** A host-agent implementation with containerized agents, scheme adapters, hashed manifests, and a canonical result schema. **(3)** A cross-family campaign on nine proving systems (pairing, Plonkish, STARK, folding, zkVM, including GPU and SIMD variants) on one workstation. **(4)** Illustrative deployment takeaways, including a Merkle-path reading for light-client and rollup inclusion proofs.

Main finding. The contract-validated data identify a deployment decision boundary rather than a single winner. In this campaign, pairing backends produced the smallest artifacts; Plonky3 and Winterfell had the lowest proving times and RAM usage, and zkVMs paid extra for memory and proof size in a guest programming model.

II. Background and Benchmarking Dimensions

A zero-knowledge proof allows a prover to convince a verifier that a statement is true without revealing any information beyond the statement's truth. This paper uses the blockchain sense of "succinct validity argument". Witness hiding holds only when the measured profile enables it. Modern systems produce non-interactive arguments of knowledge: the prover turns a computation into algebraic constraints, commits to witness-dependent objects, and answers checks derived through a transcript such as Fiat-Shamir [18]–[20]. The key point for benchmarking is that the same high-level statement can pass through substantially different internal pipelines. A handwritten R1CS circuit, a Plonkish custom-gate circuit, an AIR trace, a folding scheme, or a zkVM guest can all prove contract-level equivalent claims while performing very different internal work.

A. Arithmetization and Commitment Choices

The first benchmark-relevant choice is *arithmetization*. Rank-1 Constraint Systems (R1CS) and Quadratic Arithmetic Programs (QAPs) are natural for many pairing-based SNARKs [21], [22]. Plonkish systems use polynomial constraints over evaluation domains with custom gates, lookups, and permutation arguments. STARK-style systems express computation as an Algebraic Intermediate Representation (AIR), where the prover commits to an execution trace and the verifier checks low-degree consistency through interactive oracle proof machinery [5]. zkVMs add a further layer: the user writes a program, and the system proves the correct execution of an instruction set or a virtual machine trace.

The second choice is the *commitment-and-verification backend*. KZG commitments over pairings give compact proofs and low verification latency but require structured reference strings and pairing-friendly curves [23]. Inner-product commitments avoid pairings but trade proof size and verifier work. FRI-style commitments are transparent and hash-based. They usually lead to larger proofs but to post-quantum-attractive assumptions. These choices are not implementation details. They determine whether a system fits on-chain verification, mobile verification, high-throughput proving, or long-term transparency requirements. For a rollup or a light client, they also price calldata or data availability, because the proof size is posted with the statement.

B. Why ZKP Benchmarks Are Hard to Compare

Efforts such as zk-Bench and the Pantheon of ZKP frameworks make the field more empirical, but they also illustrate the difficulty of fair comparison when frameworks expose different abstractions [12], [13]. Four issues recur. **(1) Different semantic claims:** two benchmarks called `merkle` can differ in hash schedule, path ordering, or root binding, so identical names can hide different statements. **(2) Different specialization paths:** SHA-256 and Poseidon are exactly where custom gadgets, precompiles, or GPU kernels win for reasons that do not generalize. **(3) Different artifact accounting:** per-circuit setup, keys, SRS, trace compilation, guest builds, recursion, or wrapping may or may not be counted, which changes rankings at small sizes. **(4) Different security and hardware profiles:** curves, query counts, blowup factors, and receipt modes do not share security margins, and CPU, GPU, and SIMD builds diverge even when the proof system is unchanged. A sound benchmark need not force every system to a single profile, but it must disclose the profile it measured.

C. Metrics That Matter

We report metrics a deployment planner can act on: proving time, verification time, proof size, peak RAM and VRAM, CPU scaling, GPU acceleration behavior, and *reliability* (timeouts and infeasible runs, because omitting failed runs overstates feasibility near a resource boundary). We report them *together*. A 10 ms, 400 KB proof is neither strictly better nor worse than a 30 ms, 700-byte proof. The first can suit a transparent proving service. The second can suit an EVM verifier. A benchmark must make both readings visible.

III. Related Work

Prior empirical work on ZKP systems falls into three families: cross-system SNARK benchmarks, hardware and accelerator studies, and application-level rollup or zkVM benchmarks.

A. Cross-system SNARK benchmarks

zk-Bench [12] times field and curve kernels across several libraries and then runs SHA-256 and exponentiation on Bellman, Gnark, Circom, Halo2, and Starky. It is useful when researchers compare near-equivalent SNARK and STARK toolchains. The Pantheon of ZKP Frameworks [13] gives a practitioner-oriented comparison of higher-level frameworks (Circom/SnarkJS, Halo2, gnark, Plonky2, and others) anchored to fixed example circuits, with less emphasis on contract-level semantics. Both keep zkVM execution outside their main ranking, because the cost shape of zkVM proving (program execution, recursion, optional wrapping) is hard to place on the same axis as a hand-written circuit. El-Hajj and Oude Roelink [14] compare zk-SNARK, zk-STARK, and

Bulletproof libraries on a MiMC application and recover the familiar ordering (smallest proofs for SNARKs, fastest proving for STARKs). Their campaign stays inside three protocol families and does not validate a shared output contract across arithmetizations.

The community has since converged on shared harnesses. zk-Harness [24] is a unified, config-driven benchmark, and the ZKProof community proposal [25] specifies a common driver and standardized measurements. We operationalize and extend that program. Rather than standardizing only the *input* (for example, a shared R1CS file), we add a host-side *output contract* that validates that the same declared claim was proved across differing circuit, trace, or guest pipelines. The distinction is not cosmetic. Two `merkle` circuits can differ in path ordering or root binding (Table I) yet consume the same input, a divergence only an output contract catches. Compiler testing (Circuzz [26], MTZK [27]) checks that one toolchain matches its own spec, not that Groth16, Halo2, Plonky3, and a zkVM proved the same public claim. Liang et al. [28] dockerize 11 SNARK libraries. Kobelt et al. [29] share our selection framing. Sheybani et al. [30] survey 25 frameworks and run SHA-256 and matrix workloads. None of these applies a host-side output contract or treats failed runs as a reliability axis.

B. Hardware- and Accelerator-Focused Studies

ZKProphet [9] characterizes GPU performance for ZKP kernels and isolates MSM and NTT bottlenecks. Samardzic et al. [10], the survey by Hu et al. [11], and the NTT accelerator of Bao et al. [31] ask how to speed up a kernel given a fixed algorithm. Our framework is workload- and system-level: GPU and SIMD paths share an input contract and are observed through canonicalized artifacts. Our results echo the finding that GPU paths need setup amortization before throughput dominates total time.

C. Application-level rollup and zkVM benchmarks

Chaliasos et al. [1] measure zk-rollups at the deployment-system level (proof time, USD per transaction, L1 gas, and data posting). Velička [6] surveys zkVM designs. Yang et al. [32] give an SoK of zkVM ISA, VM, and prover layers. Gassmann et al. [33] isolate compiler effects on RISC Zero and SP1. Jolt [34] is the lookup-centric zkVM we do not integrate. Fenbushi [35], Ethproofs [36], and the Ethereum Foundation ere campaign [37] add live zkVM numbers. zkBridge [38] is the Groth16-wrapped light-client pattern. Prooϕ [39] studies prover-market incentives that need reliability accounting. These zkVM studies remain within a single family and do not apply a cross-arithmetization output contract or treat failed runs as a reliability axis. Open-source community benchmarks (`polybase/zk-benchmarks` [40] and `brevis-network/zk-benchmark` [41]) provide reproducible scripts but enforce no cross-framework semantic checks and tend to report a single hardware path.

We sit between these efforts. We benchmark the proving systems themselves, with explicit reliability and artifact accounting, and we read the Merkle-path workload as an inclusion-proof micro-benchmark. The contribution is the combination that prior work does not offer: host-side semantic contract validation across pairing, Plonkish, STARK, folding, and zkVM families at once, under a single controlled host.

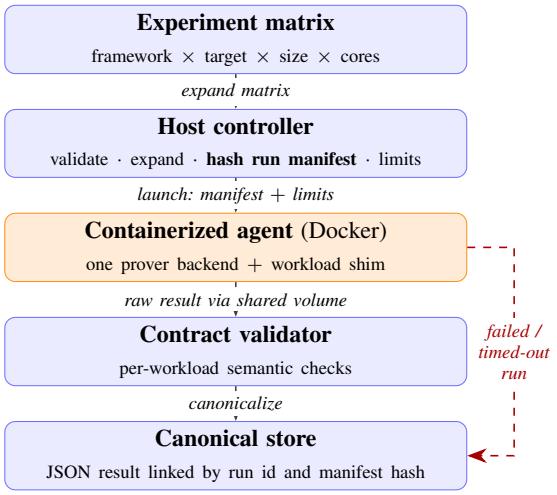

Figure 1. Host-agent benchmark workflow. Successful runs become canonical JSON artifacts.

IV. Unified Host-Agent Framework

The framework is built around one constraint: each proving system can use its natural toolchain, but all systems are launched, measured, validated, and archived through the same outer protocol. A Python *host controller* pairs with containerized *agents*, one per proving backend or variant (Figure 1). The host selects an agent through a scheme adapter, prepares a run manifest, launches the container with the requested workload and resource limits, validates the returned result, and stores the canonical artifact.

A. Host Controller and Scheme Adapters

The host controller owns all cross-framework behavior. It validates user inputs, expands experiment matrices, applies CPU limits, requests GPU access where required, monitors container logs and exit codes, and writes artifacts under a common layout. Every run receives a unique identifier and a manifest that records the scheme, target, size, CPU limit, selected device, capability metadata, and output paths. The host hashes the canonical manifest object (SHA-256) and copies that hash into each successful result, so a benchmark row can be tied back to the exact run configuration. Adapters hide launch differences (environment-variable versus command-line agents) and declare target availability, default sizes, proof modes, device and nested-Docker requirements, and result validators. We do not reimplement the proving systems. We implement only the shared benchmarking layer, so framework-specific launch details do not leak into the experiment matrix.

B. Canonical Result Schema

Each successful run emits a canonical JSON result with three parts: a `core` section of cross-framework metrics

(proving, verification, setup, and total time, peak memory, proof size, CPU count, and a verification flag), a `target_core` section of workload-specific semantic fields (for example Merkle root, MiMC output, matrix checksum), and an `extensions` section of optional backend fields (GPU peak memory, key sizes, constraint and wire counts). Only core metrics drive comparisons. Target fields validate that the declared contract claim was proved. Extension fields are diagnostic. The host does not re-verify the cryptographic proof object. It trusts the agent's reported verification flag after the container exits successfully, then applies the semantic checks in Table I.

C. Workload Contracts

The suite uses contract-level semantics rather than identical internal representations (Table I). For each primary workload, a written contract specifies deterministic input generation, the statement to prove, the public output to report, the arithmetic model, and validation checks. This fixes the level of abstraction. Systems are comparable if they prove the same contract claim and report a result that validates against it. They need not share trace layout, circuit shape, witness serialization, padding policy, or public-input encoding. SHA-256 and Poseidon are classified as secondary workloads: they are practically important, but they are exactly where optimized gadgets, precompiles, or GPU libraries dominate, so we use them to study specialization and crossover behavior, not to rank general proving efficiency.

D. Isolation, Reproducibility, and Reliability

Each agent is built from a Docker context that pins tool versions. The host shares a data volume with the container: the manifest is written before launch, the agent writes a raw result into the shared directory, and the host validates and canonicalizes it. The framework records *failed and timed-out runs* in the experiment index, with the run identifier and leftover manifest. The practical boundary of a prover is often not the median of its successful runs but the point at which memory, VRAM, or the timeout makes a workload infeasible. Reliability is therefore part of the output, not a footnote. The host does not emit a ranking. It emits contract-validated rows that a planner can filter by binding constraint. We apply that procedure in Section VII-A.

V. Experimental Setup

The campaign evaluates a representative set of proving systems and operational variants spanning pairing-based SNARKs, Plonkish proving, transparent STARK-like proving, folding, GPU acceleration, and zkVM execution. This selection isolates proving-system primitives that recur inside Ethereum-style workloads, which mix compact on-chain proofs, transparent off-chain proofs, and zkVM program proving. We do not evaluate end-to-end application contracts. That scoping limitation is stated in Section VII-C.

A. Evaluated Frameworks and Variants

Table II summarizes the integrations. Several entries are variants rather than independent systems: `icicle` and `icicle-cpu` expose the GPU and CPU paths of the ICICLE Groth16 stack; `risc0`, `risc0-cuda`, and `risc0-groth16` expose different RISC Zero proving modes; and `sp1` and `sp1-noavx` isolate the effect of CPU SIMD within the same zkVM family.

The comparison measures frameworks as configured and disclosed, not as forced to identical cryptographic settings. BN254 paths use the usual BN254 pairing-security class [15]. Halo2 uses its Pasta-curve profile. Winterfell uses its explicit STARK options. RISC Zero and SP1 use their documented receipt profiles [16], [17]. Changing proof mode can change assumptions. For example, `risc0-groth16` is bounded by its outer Groth16 wrapping.

Privacy caveat. In blockchain engineering, "zero-knowledge proof" often means a succinct *validity* proof. Several measured systems run in default validity-proving configurations and can omit witness blinding (`plonky3`, `winterfell`). RISC Zero warns that non-recursed proofs leak execution length [16]. SP1 states that its core STARK proofs are not zero-knowledge, while its compression layers are [17]. Our claims concern performance and deployment trade-offs, not privacy leakage.

Parameter disclosure. Each Dockerfile pins the upstream commit or release tag for its prover stack and toolchain. Proof-mode parameters are each project's documented native defaults (for example, Winterfell's upstream `ProofOptions` [50]; SP1-noAVX differs only in a disabled host AVX flag). "Upstream default" is not a perfect normalization across systems with different security profiles, which is why Section VII-C lists security heterogeneity as a threat to validity. The *Sec.* column of Table III reports the disclosed classical bits of the *measured* profile: BN254-class systems ≈100, Halo2/Pasta ≈126, Winterfell's explicit preset 96, Plonky3 at least 116, RISC Zero 96, and SP1 ≈100. These values are not a shared target.

B. Hardware, Matrix, and Measurement Budget

All experiments were run on one workstation: an AMD Ryzen 7 7800X3D (8 cores, SMT disabled), an NVIDIA GeForce RTX 5070 Ti with 16 GB of VRAM, 32 GB of DDR5 RAM, Ubuntu 24.04.4 LTS, Docker, and the NVIDIA container runtime (Python 3.12.3, Docker 28.2.2, driver 590.48.01, driver-CUDA 13.1). The container timeout is 7200 s. The SP1 CUDA track was excluded because 16 GB VRAM is below SP1's documented 24 GB minimum for the CUDA prover [54]. The single-machine design is deliberate but limiting: it makes the comparison controlled but not portable, so we read hardware-sensitive results as evidence for the tested environment and for the existence of bottleneck classes, not as universal constants.

Each experiment case is a tuple (*framework*, *target*, *size*, *cpus*) with CPU caps of 1, 2, 4, and 8 cores for scaling. Primary sizes were: `fibonacci` ∈ {1k, 10k, 25k, 50k,

Table I
PRIMARY WORKLOADS AND HOST-SIDE SEMANTIC VALIDATION.

Workload	Canonical claim	Validated by host	Main caveat
`fibonacci`	Fixed-start recurrence for requested size.	Arithmetic model, output value, index, transitions, requested and effective size (padded backends).	Padding and transition grouping differ across backends.
`multiplier`	Deterministic factorial-style product.	Recomputed output value, arithmetic model, and effective size.	Inputs can be generated in-prover or serialized by the host.
`grandproduct`	Closure of a deterministic grand-product relation.	Closure flag, arithmetic model, effective size, selected trace checks.	Accumulator encodings and zkVM executions express the claim differently.
`matrix`	Deterministic matrix multiplication $C = A \cdot B$.	Recomputed matrix checksum and match flag, arithmetic model.	Some systems expose a checksum, others more public values.
`merkle`	Canonical Merkle-path verification under a fixed MiMC schedule.	Root field and match flag on Halo2, Plonky3, and Winterfell. Other backends pass on the schema and the agent flag. A conformance suite recomputes the root.	Root binding differs across circuits, traces, and zkVM journals.
`mimc`	Fixed-key, fixed-message MiMC permutation with a canonical constant schedule.	Recomputed output value, optional match flag, and arithmetic model.	Constant binding and padding affect internal cost.

Table II
EVALUATED SCHEMES AND VARIANTS. EACH INTEGRATION IS PINNED BY A DOCKERFILE IN THE ARTIFACT.

Integration	Family	Setup / security shape	Mode and caveat
`groth16`	pairing zk-SNARK	circuit-specific trusted setup over BN254 class	`snarkjs` setup, `rapidsnark` proving [42]
`icicle`	Groth16 path, GPU-accelerated	trusted setup, BN254 class	CUDA path using ICICLE / ICICLE-SNARK tooling [43]–[45]
`icicle-cpu`	Groth16 path	trusted setup, BN254 class	CPU path of the same integration
`halo2`	Plonkish zk-SNARK	transparent IPA over Pasta (no toxic waste)	CPU custom-circuit path [46], [47]
`plonky3`	STARK-like, transparent	transparent-style profile	CPU path in this campaign [48]
`winterfell`	zk-STARK	transparent, explicit proof options	CPU path [49], [50]
`nova`	folding / recursive	depends on instantiation, BN254-class compression here	CPU recursion-oriented integration [51]
`risc0`	zkVM	transparent STARK receipt mode	CPU baseline [7], [52]
`risc0-cuda`	zkVM	transparent STARK receipt mode	GPU path; high VRAM demand
`risc0-groth16`	zkVM + SNARK wrapping	inner STARK + outer Groth16 assumptions	wrapped proof mode; needs nested Docker
`sp1`	zkVM	transparent STARK-family profile	CPU build with AVX [8], [53]
`sp1-noavx`	zkVM	same proof family as SP1	CPU build with AVX disabled

100k}; `matrix` $\in \{5, 10, 20, 30, 40\}$; `multiplier` $\in \{100,$ 1k, 5k, 10k, 25k}; `mimc` $\in \{500,$ 2k, 5k, 10k, 25k}; `merkle` $\in \{10, 20, 50, 100, 200, 400\}$; and `grandproduct` $\in \{10,$ 100, 500, 1k, 2.5k}. Secondary sizes were `sha256` $\in \{16,$ 64, 256, 1k, 4k, 8k} and `poseidon` $\in \{16, 64, 256, 512,$ 1k, 2k}. Plonky3 has no SHA-256 target (140 scheduled runs). Winterfell has neither SHA-256 nor Poseidon (120 scheduled runs). The other main schemes have 160 scheduled runs.

Single-run measurement budget. The campaign collected one successful run per case for the large majority of cases. The framework supports repeated runs, but budget and scheduling windows constrained this campaign to a mostly single-run design, which is the limiting factor for the strength of any quantitative claim. Reported "medians" are across distinct *cases* (different sizes and sub-workloads), not repeated runs of one case. We present no confidence intervals because the data do not support them, and we tone headlines down accordingly. Timeouts are reported separately from successful aggregates.

Aggregation rules. We aggregate only successful runs whose outputs validate against the target contract. Failed runs feed reliability summaries but are excluded from latency and size medians. Missing metrics are omitted rather than imputed. The analysis uses CPU=8 as the primary workload (the most comparable setting and the machine's maximum capacity). In Table III, the proving-time column isolates the cryptographic step, whereas total time adds setup, build, and validation overhead. Total time is an upper bound on user-facing wall time, not steady-state throughput once setup amortizes.

VI. RESULTS

We present results in four views: reliability, CPU scaling, the proving, verification, and size trade-off, and focused hardware and proof-mode comparisons. The main comparison uses primary workloads. Secondary workloads appear only where they explain specialization. All quantitative claims must be read in light of the measurement budget in Section V-B: in most cases, each contributes one successful observation, and aggregates are medians across distinct cases, not over repeated runs. We therefore phrase the findings as "observed" medians or "in this campaign," rather than as statistical rankings.

A. Reliability

Most frameworks completed their scheduled full-coverage campaign. The exceptions occurred at feasibility boundaries rather than due to semantic validation failure. In the main set, `groth16`, `icicle`, `halo2`, `risc0-cuda`, and `sp1` completed 160 of 160 scheduled runs; `plonky3` completed 140 of 140 (no SHA-256 target); `winterfell` completed 120 of 120 (no SHA-256 or Poseidon). `nova` completed 158

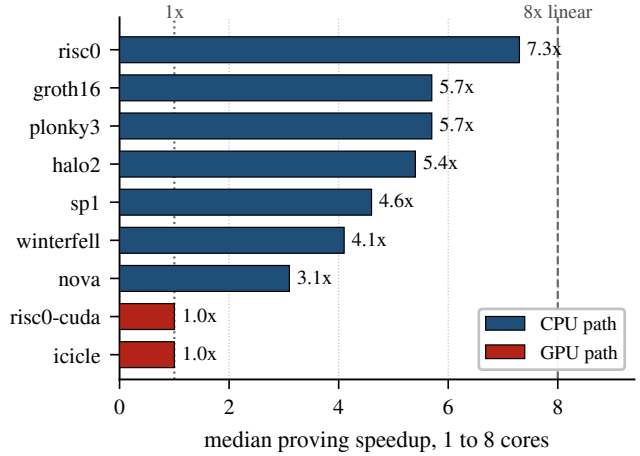

Figure 2. Median proving-time speedup t_1/t_N from 1 to 8 CPU cores per system, aggregated across matched targets. CPU-oriented paths scale toward the 8-core cap. GPU paths (`icicle`, `risc0-cuda`) stay near 1.0×.

of 160 and `risc0` completed 147 of 160. The unfinished cases were timeouts. A table of successful latencies hides cases where a framework is already near the test machine's limit. One consequence deserves explicit statement: because timeouts cluster at the *largest* sizes, the medians reported for the least-reliable systems (`risc0`, `nova`) are taken over the subset that completed (the smaller, easier cases). The cross-system medians are therefore not strictly like-for-like and, if anything, flatter those systems. We keep reliability as a separate axis rather than folding it into latency.

B. CPU Scaling

Figure 2 summarizes the median proving-time speedup t_1/t_N from 1 to 8 cores per system. The figure separates CPU-oriented provers from accelerator-bound paths. `risc0` shows the largest observed median speedup at 7.3×, with `plonky3` and `groth16` at 5.7×, `halo2` at 5.4×, `sp1` at 4.6×, `winterfell` at 4.1×, and `nova` at 3.1×. By contrast, `icicle` and `risc0-cuda` stay at 1.0×, consistent with the host CPU cap no longer being the dominant lever once the measured path is GPU-oriented. Operationally, CPU parallelism is a real lever up to 8 cores for the circuit and CPU zkVM provers. For the GPU paths, the levers are GPU initialization, VRAM, and kernel efficiency.

C. The Proving, Verification, and Size Trade-off

The headline comparison is the CPU=8 primary-workload aggregate in Table III, visualized as trade-off frontiers in Figure 3. Three observations structure the picture.

Proving time and memory favor transparent circuit-oriented systems. Plonky3 (10 ms) and Winterfell (20 ms) have the lowest median proving time and the smallest median RAM (11 and 8.3 MB). Groth16 follows at 26 ms. Among zkVMs, `risc0-cuda` improves dramatically over CPU RISC Zero (410 ms versus 31,000 ms).

Proof size favors pairing-based SNARKs. *Host verification is mixed.* Groth16 (707 B) and ICICLE (723 B) are smallest by a wide margin. Halo2 is still compact (2.3 KB). Transparent and zkVM systems range from tens of kilobytes (Winterfell) to megabytes (Nova, SP1). Host verification is ordered differently: Winterfell (0.36 ms) is the cheapest here, followed by Groth16 and ICICLE (1.4 ms to 1.6 ms), and Plonky3 (5.2 ms). Halo2, Nova, and especially SP1 (360 ms) are costly. These verify times are native host wall time, not EVM gas. Nova's table entry is the raw folding artifact (1.66 MB). The same campaign also records a Nova-specific compressed proof of about 11.5 KB, at a median compression cost of about 659 ms and a compressed verification time of about 29 ms. We keep that compression separate because it is not a common field. Proof sizes are deterministic artifact sizes. Timing and memory are rounded to two significant figures, consistent with the single-run budget, so orderings within an order of magnitude are not claimed.

The combined picture. Figure 3 shows that no system occupies the preferred (fast and compact, fast to prove and verify) corner of both panels. Groth16 and ICICLE occupy the compact-proof corner. Plonky3 and Winterfell occupy the fast-prover corner. The zkVMs sit in the upper region, trading artifact size and memory for programmability. The `Total` column must be read with care. For Groth16, it pairs low proving and verification time with a large total runtime because setup and pipeline costs dominate the small primary workloads. The setup time (total minus proving) is about 3.8 s for Groth16, 4.0 s for ICICLE, and 7.9 s for Halo2, compared to 7.5 ms for Plonky3 and 0.6 ms for Winterfell. The table is most useful when read across columns rather than sorted by a single metric.

The two accelerated paths differ in GPU memory. `icicle` has a median GPU peak of 920 MB (up to 3.5 GB), so it is limited by whether the GPU path amortizes its fixed overhead. `risc0-cuda` has a median of 6.0 GB and reaches 10.5 GB, so it is more directly VRAM-constrained (within the 16 GB device).

Table III
PRIMARY-WORKLOAD RESULTS AT CPU=8. MEDIANS ACROSS DISTINCT CASES (ONE RUN PER CASE). SEC. IS THE DISCLOSED CLASSICAL BITS OF THE MEASURED PROFILE, NOT A SHARED TARGET.

Scheme	Total ms	Proving ms	Verify ms	Peak MB	Sec. bits	Proof bytes
`plonky3`	18	10	5.2	11	116	443 250
`winterfell`	21	20	0.36	8.3	96	44 999
`groth16`	3 800	26	1.4	280	100	707
`icicle`	4 200	200	1.6	160	100	723
`halo2`	9 200	1 300	35	140	126	2 304
`nova`	2 800	97	54	100	100	1 661 400
`risc0-cuda`	430	410	10	980	96	260 810
`sp1`	10 000	9 300	360	4 000	100	5 674 577
`risc0`	31 000	31 000	14	3 300	96	260 810

D. Per-Workload and Variant Comparisons

The primary workloads are contract-equivalent yet diverse enough to expose different cost shapes (overhead-dominated `fibonacci`, arithmetic-heavy `matrix` and `multiplier`,

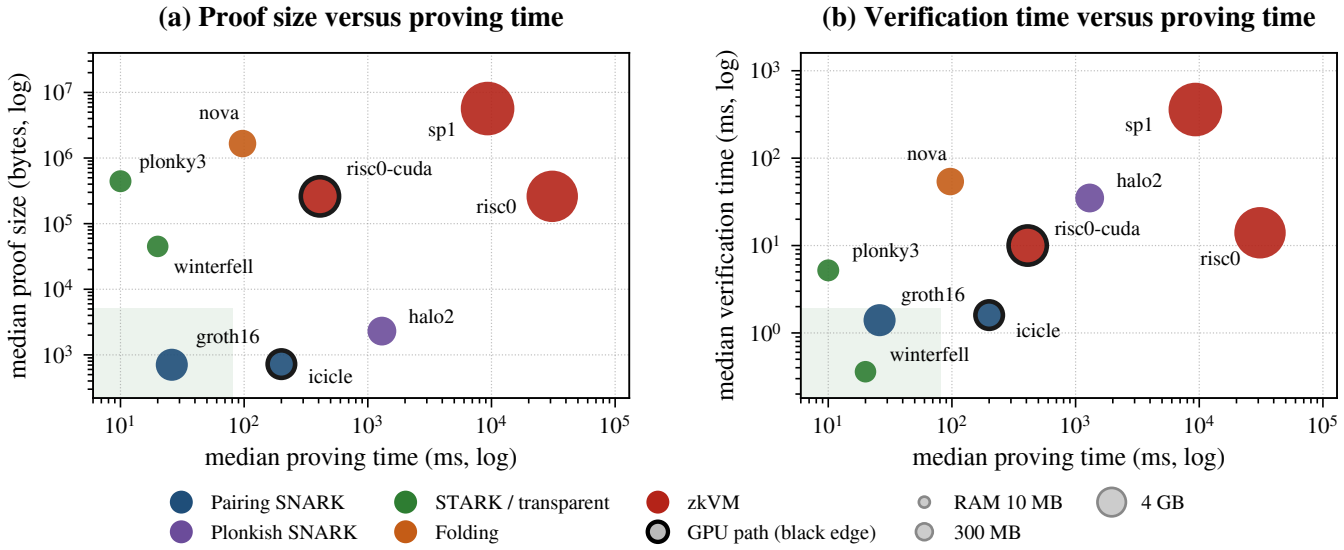

Figure 3. Deployment trade-off from the CPU=8 primary-workload medians (one observation per case; no within-case variance). (a) proof size versus proving time; (b) verification time versus proving time; both axes log-scale. Marker color encodes the proof-system family; marker area is proportional to the median peak RAM; and a black edge marks GPU-accelerated paths. The bottom-left corner (green) is the jointly preferred region.

nonlinear `mimc`, application-style `merkle`, permutation-and-lookup `grandproduct`). The secondary workloads (SHA-256, Poseidon) measure specialized engineering as much as the proof system. Setup dominates at small inputs, acceleration or custom gadgets at large ones, so we use them for crossover points, not the main ranking. On SHA-256 at CPU=8, ICICLE's GPU proving time is slower than CPU Groth16 at 256 bytes (640 ms versus 228 ms) and faster at 1024 bytes (321 ms versus 519 ms) and 4096 bytes (857 ms versus 2028 ms). Setup still dominates total wall time at every size (tens to hundreds of seconds). SP1 versus SP1-noAVX shows that AVX affects throughput in a workload-dependent way, so instruction-level optimization is not merely an implementation detail for zkVM proving.

Focusing on RISC Zero proving modes across the `merkle` and `fibonacci` workloads, GPU acceleration sharply improves proving time relative to CPU RISC Zero but shifts the bottleneck toward VRAM and GPU runtime. The `risc0-groth16` wrapping mode compresses the proof from about 261 KB to a Groth16 size (529 B at Merkle depths 10 to 50). Proving time rises to about 72 s to 83 s, of which about 62 s is the wrap. The outer Groth16 layer adds trusted setup and pairing assumptions, so the composite profile is no longer just the inner transparent STARK. This wrap was scheduled only to a depth of 50, so it is not a depth-matched row in Table IV.

Table IV
MERKLE-PATH MICRO-BENCHMARK AT DEPTH 200, CPU=8, ONE RUN PER CELL. HOST-NATIVE TIMES. `RISC0-GROTH16` RAN ONLY TO DEPTH 50 (529 B, 4.8 MS VERIFY, 83 S PROVE).

Scheme	Prove ms	Verify ms	Proof bytes	Peak MB
`groth16`	168	2.5	708	492
`icicle`	200	1.5	722	171
`halo2`	5 400	124	3 072	709
`plonky3`	80	7.1	576 334	54
`winterfell`	75	1.5	62 913	107
`nova`	81	90	5 215 064	206
`risc0-cuda`	747	11	280 174	990
`sp1`	13 000	362	5 674 577	5 100
`risc0`	82 000	15	280 174	8 600

Merkle as an inclusion-proof micro-benchmark. Of the primary workloads, `merkle` is the closest primitive to a light-client or rollup inclusion check. The path uses a canonical MiMC schedule, not an Ethereum Merkle-Patricia or Keccak tree. Table IV reports depth 200 at CPU=8, the largest depth completed by all nine main systems. Proof size is a receipt-size proxy, not calldata or blob cost. Verification is native host time, not EVM gas. A pairing-SNARK inclusion proof (708 B, 2.5 ms host verify) is the smallest artifact. A transparent proof (63 KB for Winterfell, 576 KB for Plonky3) shifts cost to bandwidth. A zkVM receipt (280 KB) is larger unless wrapped (`risc0-groth16` recovers 529 B at depth 50). These figures illustrate a posting-size trade-off on this host. They are not a measured L1, L2, or light-client cost model.

Takeaways. Three trade-offs emerge. Transparent circuit-oriented systems lead in terms of proving time and memory. Pairing-based systems lead to proof size. zkVMs lead on programmability. Hardware (CPU cores, GPU memory, SIMD) materially shifts the picture, so the binding constraint determines the recommended system (Table V). We phrase these as comparisons evidenced *in this campaign*. The single-machine, mostly single-run budget excludes universal-protocol claims, variance analysis, and privacy properties from its

Table V
DEPLOYMENT FIT ON THIS MACHINE (S **STRONG**, P **PARTIAL**, N **POOR**). AUTHOR JUDGMENT FROM TABLE III, NOT A RANKING. LAST ROW: RELATIVE PROVE-TIME × RATE, SETUP ASSUMED AMORTIZED.

Binding constraint	`g16`	`icl`	`h2`	`pk3`	`wf`	`nv`	`r0`	`r0c`	`sp1`
Compact native verify	S	S	P	n	P	n	n	n	n
Transparent (no setup)	n	n	S	S	S	P	S	S	S
General-purpose programs	n	n	n	n	n	P	S	S	S
GPU proving service	P	S	n	n	n	n	P	S	n
Memory-constrained / edge	P	P	P	S	S	P	n	n	n
Rel. $/M (illustr.)	2.6	65	130	1.0	2.0	9.7	3100	140	930

scope.

VII. DISCUSSION

The lesson is that ZKP benchmark results must be read as *deployment evidence*, not as a universal ranking. The same proof family can excel under one bottleneck and disappoint under another. This section applies the measured results to five binding constraints, adds an indicative cost lens, and states limitations.

A. Deployment Guidance

Table V maps the evaluated systems onto five binding deployment constraints, read from the CPU=8 medians (Table III) under the single-machine, single-run caveat of Section VII-C. It is a decision guide, not a ranking. Two notes cannot be encoded in the symbols. For *GPU services*, evaluate acceleration against workload size and pipeline cost, not the proof kernel alone, and report VRAM beside runtime. The *general-program* fit of RISC Zero and SP1 is appropriate when engineering complexity, rather than raw performance, is the primary cost. The benchmark confirms the price (memory, proof size, execution overhead) that buys that programmability.

Applied here: a compact-receipt constraint selects Groth16 or ICICLE (Table IV), with optional zkVM wrapping when the guest program is the product (and with the outer pairing assumptions that wrapping adds). A no-trusted-setup or low-RAM constraint selects Plonky3, Winterfell, or Halo2. A GPU-providing service selects ICICLE or `risc0-cuda` only after the workload has amortized initialization costs. General-purpose programs select RISC Zero or SP1, at the price of megabytes of proof bandwidth and gigabytes of RAM. A system that times out at the required size is not a candidate. Hash-based profiles here are conjectural FRI bits, not a proven post-quantum drop-in.

B. Indicative Cost-Efficiency Analysis

Multiplying the CPU=8 proving-time medians of Table III by $0.36 per 8-vCPU-hour and $1.20 per GPU-hour yields the last row of Table V. The timings come from a consumer workstation, so the dollars are a *relative* ordering on identical hardware, not purchasable rates. They also assume that setup amortizes. For one-shot proving, Groth16's total time (3,800 ms versus 26 ms proving) shows that setup can dominate. Three relative observations follow. **(i)** Transparent circuit-oriented systems and Groth16 sit at roughly $1 to $3 per million proofs on this illustration, not as a quoted service price. **(ii)** Against Groth16's prove median, the GPU-rate figure for `icicle` is tens of times higher on these small primary workloads. Against CPU RISC Zero, `risc0-cuda` is cheaper on the same row. Both GPU paths improve on the larger SHA-256 workloads. **(iii)** CPU zkVM figures are much larger than the circuit row. Energy was not measured.

C. Validity

Six threats bound the claims. *Synthetic scope*: the primary workloads isolate proving-system primitives, not end-to-end rollup, signature, or EVM contracts (Section VI-D). *Single-run measurements* support order-of-magnitude trends, not statistical rankings. *Single hardware* means other CPUs, GPUs, and drivers can reorder systems, so Section VII-B is an illustration. *Security and backend heterogeneity*: systems were measured as configured, not retuned to a common target, and contract-equivalent is not internally identical. *Artifact accounting* differs because setup, keys, guest builds, and wrapping can dominate small workloads. *Code and data*: the artifact is released with this publication so the campaign can be re-run.

D. Future Work

The next steps are a repeated multi-hardware campaign with confidence intervals, a normalized-security track, and application-level contracts (rollup state transitions, signature verification, private transfers, EVM fragments) under the same validation. Also, including more recursive wrapping and more enhanced zero-knowledge proof scheme techniques. Privacy leakage of journals, metadata, and wrapping is a separate measurement problem.

VIII. CONCLUSION

We presented a unified *contract-validated* host-agent framework for benchmarking zero-knowledge proving systems. Backends run in containers; runs are tied to hashed manifests; successful results are validated against a workload-specific semantic contract; and failed runs are recorded as data. Under the disclosed single-machine, single-run budget, the results identify trade-offs rather than a single winner. Transparent systems lead in low-memory proving, Groth16 and ICICLE in sub-kilobyte proofs, and zkVMs in general programs at higher memory and bandwidth costs. On the Merkle-path micro-benchmark, the same split is a receipt-size trade-off, not a measured L1 or L2 cost. A benchmark that informs blockchain infrastructure must validate the public statement being compared, disclose security profiles, and report on memory, artifact size, reliability, and cost, in addition to proving time.

DISCLOSURE AND ACKNOWLEDGEMENT

The host, adapters, Dockerfiles, results, and experiments are available at https://github.com/xhulek02/zkbench. The authors used generative AI (ChatGPT-5 and Claude 5) as a writing aid. All content and results are the authors' own, and the authors take full responsibility. This work was supported by the Science Grant Agency VEGA under project 1/0771/26, and by the BUT internal project FIT-S-23-8151.

References

[1] S. Chaliasos, I. Reif, A. Torralba-Agell, J. Ernstberger, A. Kattis, and B. Livshits, "Analyzing and benchmarking ZK-Rollups," in *6th Conference on Advances in Financial Technologies (AFT 2024)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), vol. 316. Schloss Dagstuhl — Leibniz-Zentrum für Informatik, 2024, pp. 6:1–6:24. [Online]. Available: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.AFT.2024.6

[2] J. Park and S. Lehman, "The zkVM wars: Making sense of cryptography's fiercest battleground," Symbolic Capital, Oct. 2024, industry analysis. Accessed: 2026-05-27. [Online]. Available: https://www.symbolic.capital/writing/the-zkvm-wars

[3] J. Groth, "On the size of pairing-based non-interactive arguments," in *Advances in Cryptology — EUROCRYPT 2016*, ser. Lecture Notes in Computer Science, vol. 9666. Springer, 2016, pp. 305–326. [Online]. Available: https://eprint.iacr.org/2016/260

[4] A. Gabizon, Z. J. Williamson, and O. Ciobotaru, "PLONK: Permutations over Lagrange-bases for oecumenical noninteractive arguments of knowledge," *Cryptology ePrint Archive*, vol. Paper 2019/953, 2019, accessed: 2026-05-27. [Online]. Available: https://eprint.iacr.org/2019/953

[5] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev, "Scalable, transparent, and post-quantum secure computational integrity," Cryptology ePrint Archive, Paper 2018/046, 2018, accessed: 2026-05-27. [Online]. Available: https://eprint.iacr.org/2018/046

[6] K. Velička, "Zero-knowledge virtual machines," Bachelor's Thesis, Charles University, Faculty of Mathematics and Physics, Prague, Czech Republic, 2025, supervisor: Pavel Hubáček. Accessed: 2026-05-27. [Online]. Available: https://dspace.cuni.cz/handle/20.500.11956/200877

[7] J. Bruestle, P. Gafni, and the RISC Zero Team, "RISC Zero zkVM: Scalable, transparent arguments of RISC-V integrity," Technical whitepaper, Aug. 2023, whitepaper dated 2023-08-11. Accessed: 2026-05-27. [Online]. Available: https://dev.risczero.com/proof-system-in-detail.pdf

[8] Succinct Labs, "SP1 introduction," Succinct Documentation, 2026, accessed: 2026-05-27. [Online]. Available: https://docs.succinct.xyz/docs/sp1/introduction

[9] T. Verma, Y. Yuan, N. Talati, and T. Austin, "ZKProphet: Understanding performance of zero-knowledge proofs on GPUs," in *2025 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 2025, pp. 142–154, open-access preprint: arXiv:2509.22684. [Online]. Available: https://arxiv.org/abs/2509.22684

[10] N. Samardzic, S. Langowski, S. Devadas, and D. Sanchez, "Accelerating zero-knowledge proofs through hardware-algorithm co-design," in *57th IEEE/ACM International Symposium on Microarchitecture (MICRO 2024)*. IEEE, 2024, pp. 366–379. [Online]. Available: https://doi.org/10.1109/MICRO61859.2024.00035

[11] Q. Hu, C. Peng, Y. Dai, M. Luo, and D. He, "A survey on the implementation of zk-SNARK hardware accelerators," *Chinese Journal of Electronics*, vol. 35, no. 3, pp. 1180–1194, 2026. [Online]. Available: https://cje.ejournal.org.cn/article/doi/10.23919/cje.2025.00.209

[12] J. Ernstberger, S. Chaliasos, G. Kadianakis, S. Steinhorst, P. Jovanovic, A. Gervais, B. Livshits, and M. Orrù, "zk-Bench: A toolset for comparative evaluation and performance benchmarking of SNARKs," in *Security and Cryptography for Networks — SCN 2024*, ser. Lecture Notes in Computer Science, vol. 14973. Springer, 2024, pp. 46–72, peer-reviewed version; open-access preprint: IACR ePrint 2023/1503. [Online]. Available: https://eprint.iacr.org/2023/1503

[13] Celer Network, "The pantheon of zero knowledge proof development frameworks (updated!)," Celer Network Blog, Aug. 2023, accessed: 2026-05-27. [Online]. Available: https://blog.celer.network/2023/08/04/the-pantheon-of-zero-knowledge-proof-development-frameworks/

[14] M. El-Hajj and B. Oude Roelink, "Evaluating the efficiency of zk-SNARK, zk-STARK, and Bulletproof in real-world scenarios: A benchmark study," *Information*, vol. 15, no. 8, p. 463, 2024. [Online]. Available: https://doi.org/10.3390/info15080463

[15] Y. Sakemi, S. Kanno, and R. S. Wahby, "Pairing-friendly curves," IRTF CFRG Internet-Draft draft-irtf-cfrg-pairing-friendly-curves-12, 2025, expired draft; work in progress. Accessed: 2026-05-27. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-pairing-friendly-curves-12

[16] RISC Zero, "Cryptographic security model," RISC Zero Developer Documentation, version 3.0, 2026, accessed: 2026-05-27. [Online]. Available: https://dev.risczero.com/api/security-model

[17] Succinct Labs, "SP1 security model," Succinct Documentation, 2026, accessed: 2026-05-27. [Online]. Available: https://docs.succinct.xyz/docs/sp1/security/security-model

[18] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof-systems," in *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing (STOC 1985)*. ACM, 1985, pp. 291–304. [Online]. Available: https://doi.org/10.1145/22145.22178

[19] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *Advances in Cryptology — CRYPTO '86*, ser. Lecture Notes in Computer Science, vol. 263. Springer, 1987, pp. 186–194. [Online]. Available: https://doi.org/10.1007/3-540-47721-7_12

[20] J. Thaler, *Proofs, Arguments, and Zero-Knowledge*, ser. Foundations and Trends in Privacy and Security, vol. 4, no. 2–4. now Publishers, 2022. [Online]. Available: http://dx.doi.org/10.1561/3300000030

[21] R. Gennaro, C. Gentry, B. Parno, and M. Raykova, "Quadratic span programs and succinct NIZKs without PCPs," in *Advances in Cryptology — EUROCRYPT 2013*, ser. Lecture Notes in Computer Science, vol. 7881. Springer, 2013, pp. 626–645. [Online]. Available: https://doi.org/10.1007/978-3-642-38348-9_37

[22] B. Parno, J. Howell, C. Gentry, and M. Raykova, "Pinocchio: Nearly practical verifiable computation," in *2013 IEEE Symposium on Security and Privacy (S&P 2013)*. IEEE, 2013, pp. 238–252. [Online]. Available: https://doi.org/10.1109/SP.2013.47

[23] A. Kate, G. M. Zaverucha, and I. Goldberg, "Constant-size commitments to polynomials and their applications," in *Advances in Cryptology — ASIACRYPT 2010*, ser. Lecture Notes in Computer Science, M. Abe, Ed., vol. 6477. Springer, 2010, pp. 177–194. [Online]. Available: https://doi.org/10.1007/978-3-642-17373-8_11

[24] zkCollective, "zk-Harness: A benchmarking framework for general-purpose zero-knowledge proof languages and libraries," GitHub repository, zkCollective/zk-Harness, 2025, accessed: 2026-06-09. [Online]. Available: https://github.com/zkCollective/zk-Harness

[25] D. Benarroch, A. Nicolas, J. Thaler, and E. Tromer, "Community proposal: A benchmarking framework for (zero-knowledge) proof systems," ZKProof Standards, Workshop 3, 2020, accessed: 2026-06-09. [Online]. Available: https://docs.zkproof.org/pages/standards/accepted-workshop3/proposal-benchmarking.pdf

[26] C. Hochrainer, A. Isychev, V. Wüstholz, and M. Christakis, "Fuzzing processing pipelines for zero-knowledge circuits," in *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security (CCS 2025)*. ACM, 2025, pp. 783–797. [Online]. Available: https://doi.org/10.1145/3719027.3744791

[27] D. Xiao, Z. Liu, Y. Peng, and S. Wang, "MTZK: Testing and exploring bugs in zero-knowledge (ZK) compilers," in *Network and Distributed System Security Symposium (NDSS 2025)*. Internet Society, 2025. [Online]. Available: https://dx.doi.org/10.14722/ndss.2025.230530

[28] J. Liang, D. Hu, P. Wu, Y. Yang, Q. Shen, and Z. Wu, "SoK: Understanding zk-SNARKs: The gap between research and practice," in *34th USENIX Security Symposium*. USENIX Association, 2025, pp. 2085–2104. [Online]. Available: https://www.usenix.org/conference/usenixsecurity25/presentation/liang-sok

[29] M. Kobelt, M. Sober, and S. Schulte, "A benchmark for different implementations of zero-knowledge proof systems," in *2023 IEEE International Conference on Blockchain (Blockchain)*. Danzhou, China: IEEE, 2023, pp. 33–40. [Online]. Available: https://doi.org/10.1109/Blockchain60715.2023.00015

[30] N. Sheybani, A. Ahmed, M. Kinsy, and F. Koushanfar, "Zero-knowledge proof frameworks: A systematic survey," arXiv preprint arXiv:2502.07063, 2025, accessed: 2026-06-09. [Online]. Available: https://arxiv.org/abs/2502.07063

[31] Z. Bao, T. Zang, Y. Liu, and W. Zhang, "Efficient number theoretic transform accelerator on the Versal platform powered by the AI Engine," *Future Generation Computer Systems*, vol. 166, p. 107728, 2025. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X25000238

[32] Y. Yang, Y. Cheng, H. Tang, G. Yang, B. Zhang, and K. Ren, "SoK: Understanding zkVM: From research to practice," in *Proceedings of the 21st ACM Asia Conference on Computer and Communications Security (ASIA CCS 2026)*. ACM, 2026, pp. 852–868. [Online]. Available: https://eprint.iacr.org/2026/525

[33] T. Gassmann, S. Chaliasos, T. Sotiropoulos, and Z. Su, "Evaluating compiler optimization impacts on zkVM performance," in *Proceedings of the 31st ACM International Conference on Architectural Support*

for Programming Languages and Operating Systems (ASPLOS 2026). ACM, 2026, pp. 697–714. [Online]. Available: https://arxiv.org/abs/2508.17518

[34] A. Arun, S. Setty, and J. Thaler, "Jolt: SNARKs for virtual machines via lookups," in *Advances in Cryptology — EUROCRYPT 2024*, ser. Lecture Notes in Computer Science, vol. 14656. Springer, 2024, pp. 3–33. [Online]. Available: https://eprint.iacr.org/2023/1217

[35] grandchildrice, "Benchmarking zkVMs: Current state and prospects," Fenbushi Capital, industry report, Aug. 2025, sponsored by Fenbushi Capital. Accessed: 2026-06-09. [Online]. Available: https://fenbushi.vc/2025/08/29/benchmarking-zkvms-current-state-and-prospects/

[36] Ethereum Foundation, "Ethproofs: A block proof explorer for Ethereum zkVMs," Public benchmark explorer, 2025, launched January 2025. Accessed: 2026-06-09. [Online]. Available: https://ethproofs.org

[37] I. Hagopian, "Benchmarking zkVMs for Ethereum," Ethereum Foundation zkEVM blog, Oct. 2025, accessed: 2026-08-18. [Online]. Available: https://zkevm.ethereum.foundation/blog/benchmarking-zkvms

[38] T. Xie, J. Zhang, Z. Cheng, F. Zhang, Y. Zhang, Y. Jia, D. Boneh, and D. Song, "zkBridge: Trustless cross-chain bridges made practical," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS 2022)*. ACM, 2022, pp. 3003–3017. [Online]. Available: https://doi.org/10.1145/3548606.3560652

[39] W. Wang, L. Zhou, A. Yaish, F. Zhang, B. Fisch, and B. Livshits, "Prooϕ: A ZKP market mechanism," in *Financial Cryptography and Data Security — FC 2025*, ser. Lecture Notes in Computer Science, vol. 15751. Springer, 2025, pp. 180–196, open-access preprint: arXiv:2404.06495. [Online]. Available: https://arxiv.org/abs/2404.06495

[40] Polybase Labs, "polybase/zk-benchmarks," GitHub repository, 2023, community scripts; not the artifact of Ernstberger et al. zk-Bench. Accessed: 2026-05-27. [Online]. Available: https://github.com/polybase/zk-benchmarks

[41] Celer Network, "brevis-network/zk-benchmark," GitHub repository, 2023, archived; formerly celer-network/zk-benchmark. Accessed: 2026-06-11. [Online]. Available: https://github.com/brevis-network/zk-benchmark

[42] M. Bellés-Muñoz, M. Isabel, J. L. Muñoz-Tapia, A. Rubio, and J. Baylina, "Circom: A circuit description language for building zero-knowledge applications," *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 6, pp. 4733–4751, 2023. [Online]. Available: https://doi.org/10.1109/TDSC.2022.3232813

[43] Ingonyama, "ingonyama-zk/icicle," GitHub repository, 2025, accessed: 2026-05-27. [Online]. Available: https://github.com/ingonyama-zk/icicle

[44] Ingonyama, "ICICLE documentation," Ingonyama developer documentation, v3.5.0, 2025, accessed: 2026-05-27. [Online]. Available: https://dev.ingonyama.com/3.5.0/icicle/overview

[45] Ingonyama, "ingonyama-zk/icicle-snark," GitHub repository, 2025, accessed: 2026-05-27. [Online]. Available: https://github.com/ingonyama-zk/icicle-snark

[46] Electric Coin Company, "zcash/halo2," GitHub repository, 2026, accessed: 2026-05-27. [Online]. Available: https://github.com/zcash/halo2

[47] Electric Coin Company, "Protocol description — the halo2 book," halo2 online documentation, 2026, accessed: 2026-05-27. [Online]. Available: https://zcash.github.io/halo2/design/protocol.html

[48] Plonky3 contributors, "Plonky3/Plonky3," GitHub repository, 2026, accessed: 2026-05-27. [Online]. Available: https://github.com/Plonky3/Plonky3

[49] Facebook (Meta), "facebook/winterfell," GitHub repository, 2026, accessed: 2026-05-27. [Online]. Available: https://github.com/facebook/winterfell

[50] Facebook (Meta), "ProofOptions in winterfell," Rust documentation on docs.rs, 2026, accessed: 2026-05-27. [Online]. Available: https://docs.rs/winterfell/latest/winterfell/struct.ProofOptions.html

[51] A. Kothapalli, S. Setty, and I. Tzialla, "Nova: Recursive zero-knowledge arguments from folding schemes," in *Advances in Cryptology — CRYPTO 2022*, ser. Lecture Notes in Computer Science, vol. 13510. Springer, 2022, pp. 359–388. [Online]. Available: https://eprint.iacr.org/2021/370

[52] RISC Zero, "risc0/risc0," GitHub repository, 2025, accessed: 2026-05-27. [Online]. Available: https://github.com/risc0/risc0

[53] Succinct Labs, "succinctlabs/sp1," GitHub repository, 2025, accessed: 2026-05-27. [Online]. Available: https://github.com/succinctlabs/sp1

[54] Succinct Labs, "SP1 hardware acceleration," Succinct Documentation, SP1 v6 (Hypercube), 2026, accessed: 2026-05-27. [Online]. Available: https://docs.succinct.xyz/docs/sp1/generating-proofs/hardware-acceleration