Nexus 1.0: Enabling Verifiable Computation

Daniel Marin [*]

Michel Abdalla, Paul Govereau, Jens Groth, Samuel Judson, Kristian Sosnin, Guru Vamsi Policharla, and Yinuo Zhang [†]

Nexus Labs
{daniel,michel,paul,jens,sam,kristian,vamsi,yinuo}@nexus.xyz

January, 2024

Abstract

We introduce the Nexus project, an endeavor to enable verifiable computation, at Internet scale. The world has come a long way since Turing introduced in 1936 a *universal computing machine*, a hypothetical machine capable of executing any computation. This concept is considered the origin of the general-purpose computer, and was used by von Neumann to introduce the von Neumann architecture, a physical instantiation of the Universal Turing Machine. This architecture now powers virtually all modern computers.

In this paper, we introduce the Nexus zkVM (zero-knowledge virtual machine), a machine capable of *proving* any computation. That is, the machine produces succinct zero-knowledge proofs of correct program execution for any stateful machine (e.g. RISC-V, EVM, Wasm) with any particular instruction set. The Nexus zkVM focuses on proving large computations (e.g. 1B+ CPU cycles), and is architected to enable massively parallelized incremental proof generation, well suited for parallel proving in a distributed prover network.

The zkVM is powered by modern high-speed recursive proof systems (i.e. folding / accumulation) that allow proofs to be combined and aggregated, realizing the notions of Incrementally Verifiable Computation (IVC) and its generalization, Proof-Carrying Data (PCD). Further, we introduce the Nexus Virtual Machine (NVM), a simple, minimal, and extensible Universal Turing Machine: a virtual CPU architecture specifically designed to maximize prover performance. The NVM is the core computational model of the Nexus zkVM, and can simulate with minimal overhead any other ISA like RISC-V, EVM, Wasm, etc, through compilation and emulation techniques, as well as extensions in the instruction set (e.g. SHA-256).

We briefly describe the Nexus Network, an upcoming large-scale distributed prover network that aggregates the collective CPU / GPU power of a pool of heterogeneous computers to instantiate an extremely parallelized large-scale proof generation system for the Nexus zkVM. This enables the zkVM to operate at a scale (measured in CPU cycles *proved* per second) proportional to the collective computing power of the network.

Fundamentally, the Nexus project seeks to bring the concrete cost of verifiable computation down by orders of magnitude through a variety of scientific, engineering, and economic techniques, so that it may finally become a practical form of computation. The project builds upon decades of scientific research in cryptography, complexity theory, compilers, and high-performance computing. We focus on delivering a simple developer experience and a system designed to power production-grade applications, with initial support for Rust programs.

We envision a future for the Internet where the integrity of computations and data are protected by proofs: a future where human cooperation is enhanced by mathematical truth. This constitutes our first construction towards that vision.

[*]Main author.

[†]Thanks to the Nexus team for helpful discussions, listed here in alphabetical order.

Contents

1 Summary

In this whitepaper, we describe our vision for the Nexus project and our first set of technologies. For a formal treatment and new techniques, please refer to our follow on technical paper.

TL;DR: Nexus is building a distributed zkVM,
designed to *prove* a **trillion CPU cycles per second**.

We believe that the world's computers can unite into a single, verifiable supercomputer,
and jointly prove humanity's computation.

How to read. This document progresses in an increasing level of technical detail.

- **For general readers**: we recommend reading the Summary (Sec. 1) to learn about:
    1. The Nexus zkVM (Sec. 1.1)
    2. The Nexus Virtual Machine (Sec. 1.2 and 1.3)
    3. The Nexus Network (Sec. 1.4)
- **For developers**: we recommend checking out the Example (Sec. 1.6) and open-source implementation (Sec. 1.5), as well as the official Nexus documentation `https://docs.nexus.xyz`.
- **For advanced readers and cryptographers**: we recommend reading the Introduction (Sec. 2), and the historical Background (Sec. 3). For an in depth technical understanding of the Nexus zkVM, we recommend Section 4 and onwards.

For questions, feedback, or to get involved, please reach out to us at `hello@nexus.xyz`.

1.1 The Nexus zkVM

We introduce the Nexus zkVM, a machine capable of proving any computation. Our implementation is fully open source[1]. The zkVM focuses on proving very large computations (say, 1B+ CPU cycles). It achieves this through extreme parallelization: the zkVM realizes lightweight and unbounded Incrementally Verifiable Computation (IVC) [Val08], a primitive that allows proofs of correct computation to be computed and updated incrementally, through the use of modern high-speed recursive zero-knowledge proof systems (i.e. folding / accumulation schemes) [KST22; KS22; KS23b; KS23a; BC23]. These techniques allow for high-speed proof accumulation (see Fig 1), without using SNARKs at all.

Proof accumulation (without SNARKs), see Fig 1, is a novel technique that allows proofs to be combined and aggregated, which in itself allows proof generation to be performed efficiently in a large-scale network of untrusted computers. Through this, the zkVM effectivelly realizes the notion of Proof-Carrying Data (PCD) [CT10], a generalization of IVC to the distributed setting, wherein proof computation is performed by a distributed prover network. However, realizing high-speed IVC through SNARK-less recursion comes at the expense of large proofs. To tackle this, we compress the final accumulated proof with a recursive sequence of SNARKs.

Execution Sequence. The Nexus zkVM execution sequence is separated into three stages, with an additional one-time setup phase. See Fig. 2 for a visual representation.

[1] `https://github.com/nexus-xyz/nexus-zkvm`

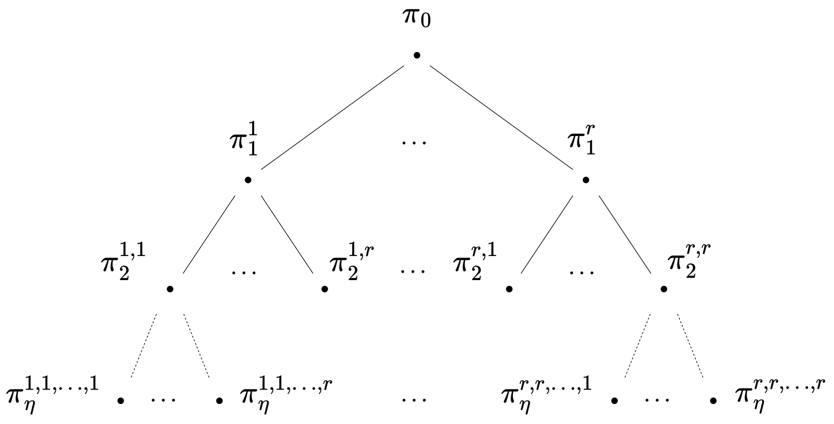


Figure 1: An r-ary proof accumulation tree of depth η, *without* SNARKs.

- **One-time setup.** Optionally specify a custom machine architecture $\Xi = [F_1, \dots, F_\ell]$ as a list of ℓ additional CPU instructions, and generate public parameters for the system. This only needs to be done once for a given machine architecture.

  The machine Ξ can be, for example, the empty list $\Xi = []$ (i.e. default execution on the NVM, see Sec. 1.2), a single circuit $\Xi = [F_1]$ (e.g. BLS signature aggregation [BLS01]), a full machine architecture (e.g. EVM [Woo+14], RISC-V [WLPA14], Wasm [Haa+17]), or a list of custom user-defined instructions (e.g. SHA-256, ECDSA sign) that extend the NVM.

  We call these extensions zkVM co-processors, see Sec. 1.3.

Then, for every program Ψ encoded for the machine Ξ, we execute the following steps:

- **Compilation.** Compile the program to the Nexus Virtual Machine (NVM) instruction set. The NVM will execute the the program Ψ on the machine Ξ.

- **Execution.** Execute the program on the NVM, generating the full execution trace.

- **Folding.** In a massively parallelized fashion, produce and accumulate IVC proofs π_i for blocks of execution. This step is the core of the proving process, and is the most computationally intensive. However, it is extremely parallelizable, and can be done by a distributed prover network.

  Every folding step requires computing a *single* multi-scalar multiplication (MSM). This is also the lowest prover overhead in the context of recursive proof composition [KS23b]. In addition, MSMs are themselves highly parallelizable, as has been well studied by industry efforts [Aas+22].

- **Compression.** Finally, compress the accumulated proof with a sequence of (zk)-SNARKs.

1.2 The Nexus Virtual Machine

The Nexus Virtual Machine (NVM) is a simple, minimal, and extensible Instruction Set Architecture (ISA) and Random-Access Machine (RAM) with a von Neumann architecture that enables universal computation. That is, it is a Universal Turing Machine. The NVM is inspired by vnTinyRAM [CGTV20] and the RISC-V [WLPA14] ISA, but unlike the latter, it designed to maximize prover performance.
The NVM features:

- **A simple CPU architecture**: simple 32-bit instruction set, 40 instructions, and simplified instruction decoding.

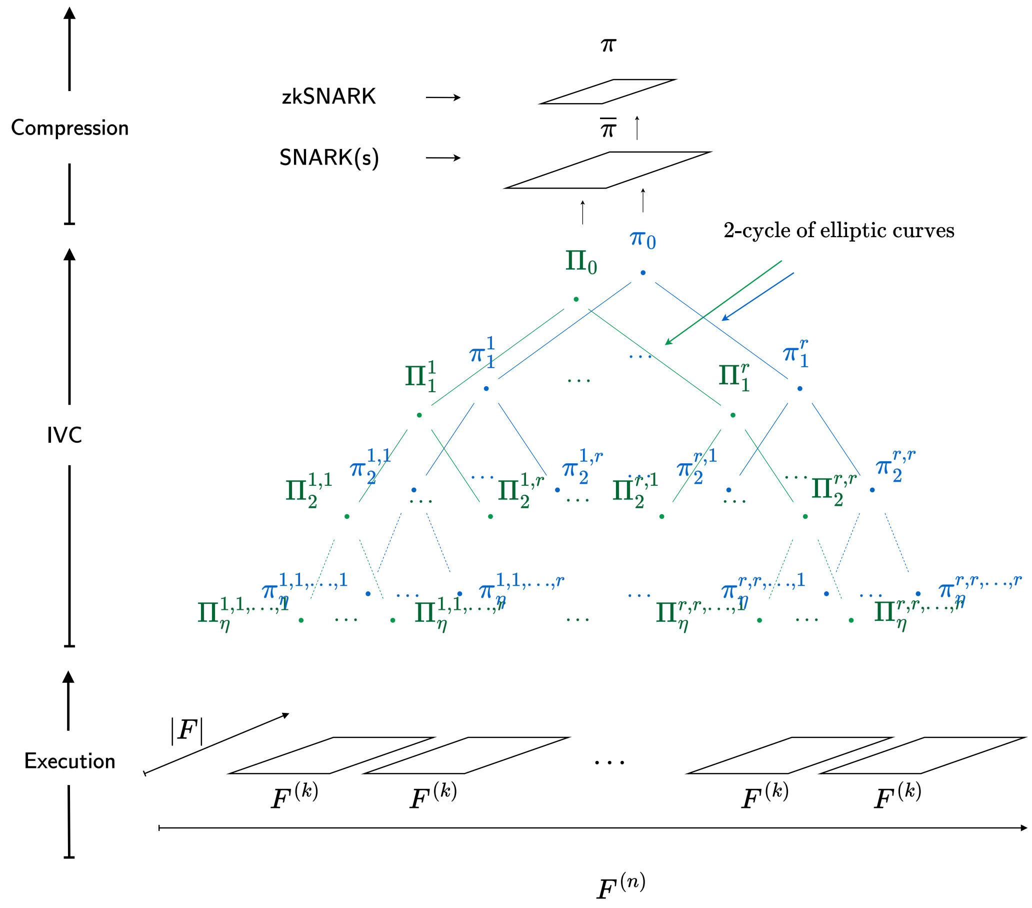


Figure 2: The Nexus zkVM execution sequence. Here, F represents a CPU cycle on the Nexus Virtual Machine. Step 2 (IVC / PCD) is the massively parallelized distributed SNARK-less incremental proof generation, which happens (for technical reasons) in a 2-cycle of elliptic curves [TG14; NBS23; KS23a].

- **A simple memory model**: a simple 32-bit address space, with a single stack and heap, proved incrementally with Merkle Trees [Mer87] and Poseidon hashes [Gra+21].
- **A simple I/O model**: a simple 32-bit input / output model, with single tapes for public input (x), private input (w), and public output (y). Given a program Ψ, the NVM executes its instructions according to the ISA on the public and private inputs and records the resulting output.
- **Extensibility**: the NVM can be extended with custom instructions, which we call zkVM co-processors. These are user-defined instructions written as CCS circuits [STW23a], a generalization of R1CS [GGPR13], Plonkish [GWC19; CBBZ23], and AIR [BBHR19; Sta21; BCKL22], that extend the NVM instruction set, and can be used to accelerate custom instructions, without affecting per-cycle prover performance. Nexus co-processors are thus conceptually similar to EVM precompiles [Woo+14].

The Nexus zkVM can be seen as a combination of the NVM (the computational model) and a proof system (the prover). The zkVM runs the NVM and produces exactly the same output. In addition, it also constructs a succinct proof of correct computation. That is, the zkVM proves in zero knowledge, knowledge of a private witness w such that $\Psi(x, w) = y$. Note that, due to the zk-SNARK at the end of the compression step (see Fig. 2), the Nexus zkVM is *actually* zero knowledge (i.e. not only succinct) and, assuming the prover does not leak it, it hides the (optional) private witness w.

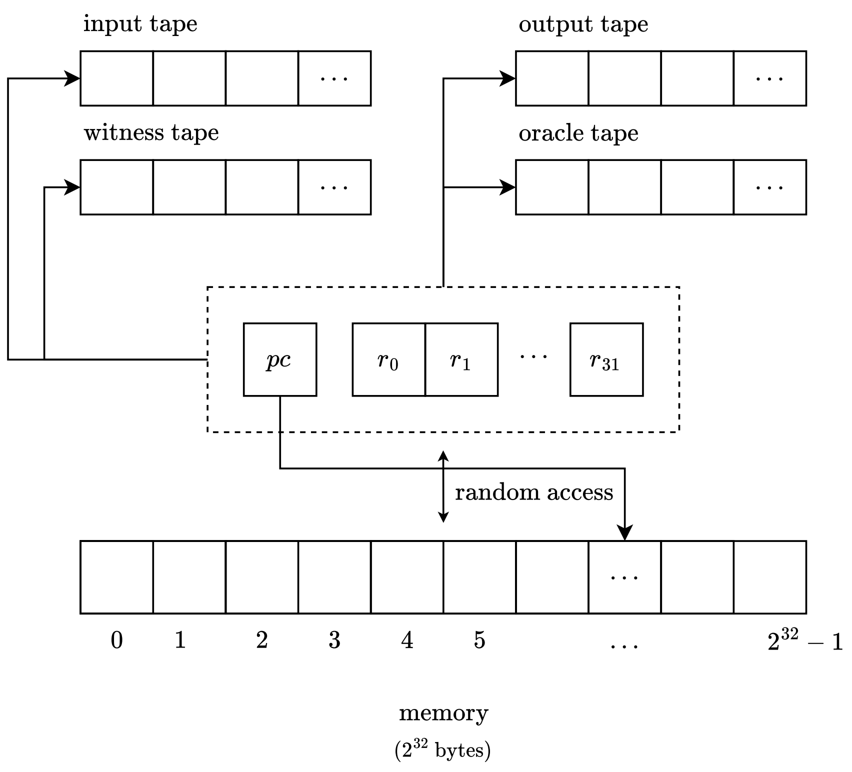


Figure 3: The Nexus Virtual Machine.

1.3 The Nexus zkVM co-processors.

The notion of zkVM co-processors is a powerful one inspired by [KS22]. While in theory any program can be proven as a sequence of CPU instructions, proving many kinds of simple programs in this model is practically infeasible. For example, proving even a single SHA-256 hash in the virtual machine model would involve proving about 64,000 CPU cycles that emulate the SHA-256 function in-software. In contrast, proving a manually-written SHA-256 hash circuit (like an ASIC) directly, *outside* of the abstraction of a VM, would involve proving only about 30k constraints, which requires roughly 1000x less computation. Such is the cost of abstraction.

In the traditional zkVM design (e.g. vnTinyRAM [CGTV20]), one uses *universal circuits* to simulate a whole CPU. Adding a new instruction to the CPU, involves increasing the total number of constraints proved per step, so the natural strategy is to minimize the size of the CPU being proven.

To escape this CPU vs. ASIC dilema, the Nexus zkVM introduces the concept of zkVM co-processors, where the cost of proving custom co-processor extensions of the NVM instruction set, is only paid for by the prover when that particular co-processor is executed by the guest program. This allows the Nexus zkVM to maintain the developer-friendly CPU abstraction with a small Turing-complete ISA, while allowing ASIC-like extensions on the instruction set (see Fig. 5), while only paying for the cost of those instructions when they are actually executed. These techniques are possible due to recent advancements in *non-uniform* IVC and related folding / accumulation techniques [KS22; BC23; ZGGX23; AST23; GHK23].

In particular, zkVM co-processors allow the zkVM to accelerate custom cryptographic primitives (e.g. SHA-256, ECDSA signatures), higher-level operations (e.g. matrix multiplications, computing square roots, etc.), and even recursively composing with other SNARKs by injecting their verifier circuits as custom instructions in the zkVM. The downside is that these custom-purpose circuits need to be

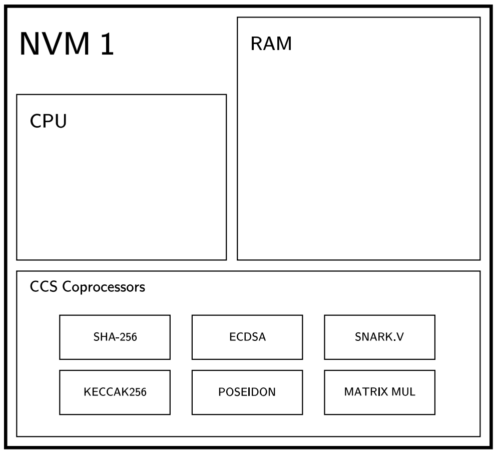


Figure 4: The Nexus Virtual Machine (NVM) Instruction Set Architecture (ISA).

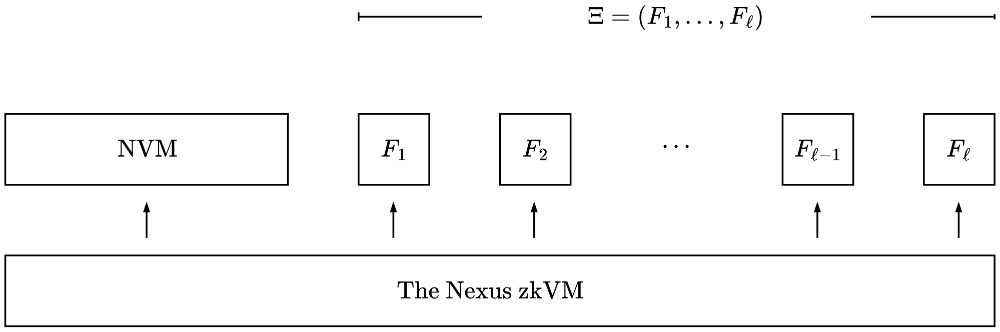


Figure 5: The Nexus Virtual Machine zkVM co-processors, designed to allow for ASIC-like performance for accelerated instructions as extensions on the NVM instruction set.

manually implemented, but they only need to be implemented once.

Simplicity and Universality. Beyond optimizing for prover performance, the NVM seeks to answer a secondary question, of a social nature: Can we design a simple enough universal model of computation (i.e. a concrete instantiation of the Universal Turing Machine), such that a proof of computation on the NVM is universally convincing?

For example, consider proving the Fibonacci function $\mathsf{Fib}(n)$ by producing a proof π_1 through emulation of an x86 machine (with 2^{120} potential instructions, and large architectural complexity), versus obtaining a proof π_2 through emulation on a say, 32-instruction machine, with only basic ADD, SUB, XOR, etc. instructions. Which proof, π_1 or π_2, might be more convincing as a faithful proof of the $\mathsf{Fib(n)}$ computation? Which proof, π_1 or π_2, might be more convincing to someone living 10 years in the future? Which verifier circuit might be simpler to implement and audit?

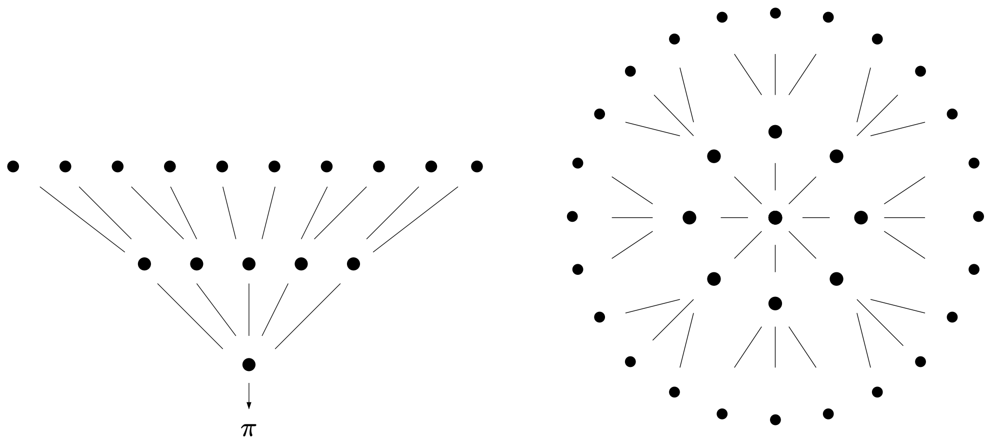
1.4 The Nexus Network

The Nexus project seeks to enable verifiable computation at Internet scale. To this end, we present a system by which the total throughput of the Nexus zkVM can be increased by orders of magnitude. We call this system the Nexus Network, a system that aggregates the collective CPU / GPU power of a heterogeneous computer network to instantiate an extremely parallelized large-scale proof generation system for the Nexus zkVM. This allows the zkVM to operate at a scale (measured in CPU cycles *proved* per second) proportional to the collective computing power of the network.

Distributed Supercomputers. Large-scale volunteer computing over the Internet has seen great success in some notable projects. These projects have harnessed the collective computing power of millions of volunteers to solve large-scale scientific problems, with the total computing power of these systems measured in FLOPS (floating point operations per second). Examples include:

- The Great Internet Mersenne Prime Search (GIMPS) [96] in 1996, reaching 14 teraFLOPS in the early 2004s.
- SETI@Home [And+02] in 1999 from NASA, which received the 2008 Guinness World Records for the largest computation in history, reaching 668 teraFLOPS.
- Folding@Home [Beb+09] in 2000 from Stanford University, reaching 2.43 exaFLOPS, making it the first exaFLOP computing system.

The Nexus Network seeks to do the same, but for the problem of verifiable computation. Thus, the Nexus Network can be seen as a *distributed verifiable supercomputer*, that allows the zkVM to scale its throughput in a way proportional to the collective computing power of the network.



(a) Tree view.

(b) Aerial view.

Figure 6: The Nexus Network. Proof aggregation happens in a tree-like fashion.

Verifiability in Distributed Computation. The projects mentioned above relied on varied heuristics for ensuring the integrity of the distributed computation. For example, SETI@Home relied on replicating computations between multiple nodes, and then comparing results amongst them. If there was a disagreement, the computation would be repeated with another randomly selected set of volunteers. This gave at best probabilistic guarantees on the integrity of the computation.

In contrast, the Nexus Network gives provable guarantees on the integrity of the distributed computation, as the output itself is a proof. Note that this is not only true of the final proof, but also of all the intermediate proofs in the incrementally verifiable computation. For the first time, such a distributed verifiable computation system is practically possible due to the instantiation of the Proof-Carrying Data system based on the high-speed recursive proof aggregation techniques of the Nexus Network.

A Verifiable Supercomputer. Thus, we pose a bold question:

Can we unite the world's computers into a single, verifiable supercomputer?

In particular, we think that, provided enough computing power, the Nexus zkVM may reach *trillions of CPU cycles proved per second*, and so be able to prove computations that are orders of magnitude larger than what is currently possible. The Nexus Network is the first step towards this vision.

Practical matters. The Nexus zkVM partitions the proving process into chunks of computation, which can be arbitrarily small, and then accumulates proofs incrementally. Thanks to this, the Nexus provers can operate with as little as 1GB of RAM. This allows even small devices like smartphones, and laptops to participate in the network and supply useful compute. In practice, we expect large powerful provers to dominate the supply of compute in the system, as clusters of GPUs, FPGAs or ASICs, optimized for computing MSMs, are many times more powerful.

1.5 Open-Source Implementation

So what has been implemented? The Nexus zkVM and the Nexus Network are fully open source, and both are implemented in Rust. These include:

1. **Proof accumulation: Nova, CycleFold, and HyperNova.** Following the original open-source implementation of Nova [KST22] from Microsoft Research in 2022, we provide the first production-grade implementation of the Nova and follow-up proof accumulation systems, all built from-the-ground-up in arkworks [ark22]:

    - The Nova folding scheme [KST22]
    - The CycleFold multi-folding scheme [KS23a]
    - The Nova + CycleFold IVC proof system (see Sec. 5.4)
    - The binary-tree parallelized Nova scheme (see Sec. 5.6)
    - In progress, the first implementations of CCS [STW23a] and HyperNova [KS23b]

    We hope these are useful to the community and other projects as well. We are actively improving these with new techniques [BC23; ZZD23; ZGGX23]. Our implementation presents concretely a 50k recursion overhead (measured in number of constraints) for Nova with CycleFold, and a 200k recursion overhead for parallel Nova with CycleFold. We shall report measurements for HyperNova in our follow-on technical paper.

2. **The Nexus zkVM.** An implementation of the Nexus zkVM, and all the steps in the execution sequence: generation, compilation (witness extraction), execution, folding, and proof compression. See Fig. 2.

    Using the Nexus zkVM on any Rust program such as Example 1 is as simple as doing:

    ```
    $ cargo nexus prove → $ cargo nexus verify
    ```

3. **The Nexus Virtual Machine.** An implementation of the NVM, and it's arithmetized version in R1CS, with Merkle proofs for memory checking, as well as a RISC-V-to-NVM compiler.

    The NVM arithmetization, which is rather naive still, is a circuit of size 30k per CPU cycle. This is a large circuit (mainly due to memory-checking), but we expect upcoming improved memory-checking techniques to reduce the size of the machine by 10x, with further order-of-magnitude improvements coming from the use of zkVM co-processors [KS22; ZGGX23; GHK23], and lookup arguments [STW23b; AST23].

Normal users do not have to worry about this intermediate optimization, but it leads to significant performance improvements from compiler optimizations, and better security due to the simplicity of the machine.

4. **The Nexus Network.** An open-source implementation of the Nexus Network, featuring three different types of nodes:

   - **The Nexus MSM Prover**, a prover node that can be run on any machine with at least 1 GB of RAM. These nodes supply compute to the network by computing MSMs on-demand. Supplying compute to the system is as simple as doing:

     ```
     $ cargo nexus compute
     ```

   - **The Nexus PCD Prover**, a prover node that can be run on any machine, and can supply compute to the network by computing IVC proofs. The Nexus team currently operates these.
   - **The Nexus Aggregator**, a node that aggregates proofs and compresses them with SNARKs, by applying the Nexus zk-SNARK compression sequence (see Fig. 2).

5. **The Nexus Proof Compression Sequence.**

   We present:

   - **A Nova-friendly zkSNARK**. The first implementation (besides Microsoft's [KST22]) of a zk-SNARK for an aggregation-friendly generalization of R1CS: A modified Spartan [Set20] zk-SNARK for the commited relaxed R1CS [KST22] relation, adapted from the original Spartan arkworks [ark22] implementation for R1CS.
   - An implementation of the Zeromorph [KT23] polynomial commitment scheme.

   We further apply a final level of proof compression with other SNARKs [Gro16; BGH19; BGH20] to achieve Ethereum verifiability for Nova proofs, which allows Nexus to conncect to Ethereum. These details will be described in the technical paper.

What has not yet been implemented:

- **Non-uniformity.** Our current implementations do fully-uniform IVC (see Fig. 5), but not non-uniform. Support for CCS extensions is actively being developed.

- **zkVM I/O**. The Nexus zkVM currently does not support I/O (e.g. loading a JSON file). All input must be encoded in the program itself. This is a small but important limitation that will be addressed in the comming weeks.

1.6 Example

```rust
#![no_std]
#![no_main]

#[nexus::main]
fn fib(n: u32) -> u32 {
    match n {
        0 => 1,
        1 => 1,
        _ => fib(n - 1) + fib(n - 2),
    }
}
```

Listing 1: Example of a Rust program provable on the Nexus zkVM.

2 Introduction

Turing Machines. In his seminal 1936 paper [Tur+36], Turing introduced the Turing machine, a formalization of the notion of computation. He was the first to observe that general-purpose computation is possible and showed the existence of a *universal* Turing machine, a machine which can simulate the execution of any other machine. Similarly, Kurt Gödel, Alonzo Church, and others had also arrived at formal definitions of computation in the early 1930s, all of which ended up being identical in expressive power.

The von Neumann Architecture. The notion of the Universal Turing Machine quickly led other scientists, such as von Neumann, to construct implementations. Today, essentially all digital computers follow the "von Neumann architecture" [Von93]. The Nexus VM also follows the von Neumann architecture.

Classical Proofs. In 1956, Gödel wrote a letter to von Neumann [Göd56; Sip92], where he identified how "it would have consequences of the greatest magnitude" if there existed a practical machine that could efficiently prove mathematical theorems. As we will see, this notion is close to that of proving *computations*. Gödel had essentially posed what we know today as the (still unsolved) $\mathbf{P} = \mathbf{NP}$ question, which deals with the relationship between deterministic (super-polynomial time) provers (i.e. the class **NP**), (polynomial time) verifiers (i.e. the class **P**), and the notion of proofs[2].

Zero-Knowledge Proofs. Fast forward to 1985, the story of (zero-knowledge) interactive proofs began at MIT, in a paper by Goldwasser, Micali, and Rackoff [GMR19] that introduced the class **IP**, a generalization of **NP** that allows *randomness* and *interaction* between the prover and verifier. This fundamentally changed the notion of *proof* itself. The definition of *zero-knowledge*, which they introduced as well, considers proofs that reveal nothing besides their own validity. Their paper would win the first ever Gödel Prize.

Since then, a cambrian explosion of research introduced other notions of proofs, such as Arthur-Merlin protocols [Bab85] (concurrent work), MIPs [BGKW19; FRS88; BFL91], PCPs [FRS88; BFLS91; AS92; Aro+92], IOPs [BCS16], NIZK [BFM88; BSMP91], proofs of knowlede [SP92], succinct arguments and CS proofs [Kil92; Mic94], and (zk)-SNARKs [Mic94; Gro10; BCCT12; GGPR13]. For a short recap of the history of zero-knowledge proofs and modern techniques until present, see Section 3.

The Nexus project builds upon these works and nearly four decades of research since the invention of zero-knowledge proofs. We believe that due to very recent scientific progress, truly scalable zero-knowledge for the Internet era is close within the horizon.

2.1 Why are zero-knowledge proofs still not practical?

The ASIC Approach. Current typical approaches to compute zero-knowledge proofs consist in writing arithmetic circuits, either manually (e.g. R1CS, Plonkish, AIR, etc.), through libraries like Bellman [Zkc15] or arkworks [ark22], or through a higher-level DSL (e.g. Circom [ide21], Zokrates [Zok17], Noir [Noi20], Leo [Chi+21], etc.) with a DSL-to-circuit compiler, and then proving the circuit with a zk-SNARK. For the zk-SNARK, Groth16 [Gro16] is the most widely used in practice (due to practical Ethereum compatibility and its small proof size), followed by Halo and Halo2 [BGH20]. The ASIC approach is thus coneptually similar to designing a custom ASIC for a given computation, and then proving the ASIC's execution with a zk-SNARK.

[2]For a more precise description, see the excellent books in computational complexity theory [AB09] and [Wig19].

Disadvantages. Unfortunately, this approach is very error prone and requires a high degree of expertise. Further, the DSL approach places static bounds on the programs (e.g. arrays and loops must have a fixed size known at compile-time) [Chi+21], does not support input-dependent control flow and memory accesses [BCTV14], and does not allow for self-modifying code. This approach also results in a different circuit being generated for every program, which requires redeploying new verifier code to be deployed each time, a highly impractical and often undesirable operation.

These are all large limitations in practice: even if the developer is willing to learn a new DSL or to manually write arithmetic circuits (which is a highly non-trivial task), they will learn that most programs are practically uncomputable in this model.

The CPU Approach. A modern approach, introduced in [BCTV14], is to prove *computer programs* as opposed to arithmetic circuits. This is done by emulating a CPU or Virtual Machine (VM), by implementing the full VM as a single *universal* circuit. In this model, the size N of the universal circuit is proportional to the sum of the sizes of the instructions in its instruction set. Then, a sequence of state transitions of the VM is proven by glue-ing many such universal circuits together, up to a static bound T on the runtime of the program, and then proving the $(N \cdot T)$-sized circuit with a SNARK. This approach was introduced by TinyRAM [CGTV20], and has resulted in industry applications like the Cairo zkVM [GPR21] and other zkEVM projects for the Ethereum Virtual Machine.

Advantages. The advantage of this approach is better developer experience. Writing programs for a pre-existent or well-defined ISA is significantly simpler, as developers can exploit tooling and compiler infrastructure. Moreover, since the circuit is universal, only a single verifier needs to be deployed (this is in practice a significant advantage). Further, the large effort of writing the universal circuit for a given VM like the EVM (a very large effort that has taken months or years to complete to serveral projects) only needs to be done once, and then any program for that ISA, up to the specified time bound T, can be proven.

Disadvantages. However, the CPU approach has several disadvantages, mainly related to performance and scalability.

1. **Bounded computation.** This approach requires deciding on a static bound T on the runtime of all possible program executions. Proving beyond T cycles requires either (1) generating new public parameters and deploying a new verifier for a new bound T' (increasing also prover time and memory) [Ben+13; BCTV14], or (2) doing SNARK recursion [BGH20; BCMS20; Bün+21; CGSY23].

2. **High cost of SNARK recursion.** The second option consists in proving executions in chunks of size T and "glue-ing" the proofs together by writing the *full SNARK verifier itself as a circuit* (which is in practice very hard to do), and proving that the SNARK verifier at step i has accepted the SNARK proof from step $i - 1$. This approach is extremely costly and is well-known to be impractical [BCTV17; CCDW20; KST22]. In practice, this has led projects like TinyRAM, the Cairo zkVM, and other zkEVM projects, to consume large computational resources while only being able to prove practically small programs (e.g. smart contracts).

3. **Low performance due to emulation.** However, this approach faces serious scalability issues. First, the VM abstraction leads to much larger circuits than circuits that do not. For example, proving a keccak256 hash, verifying an ECDSA signature, or aggregating BLS signatures through emulation consists on proving tens of thousands or millions of CPU cycles, as these operations have to be emulated within the CPU abstraction [GPR21]. Secondly, proving T copies of a universal circuit requires large prover machines with very large memory (e.g. about 100 GBs of RAM), necessary to materialize the $(T \cdot N)$-sized witness.

4. **Practical security.** Lastly, these techniques are hardly auditable. Even if one implements a whole VM as a circuit (after months or years of effort), and say, a full SNARK verifier as a circuit as well (for recursion), it is still hard to convince oneself that the circuits are actually implemented correctly, and have no vulnerabilities [OWBB23; CMS23; Cog+23]. Even a single missing constraint in the circuit implementation would lead to a full soundness exploit (e.g. as shown in [NBS23]), and the verifier would accept invalid proofs.

2.2 The Nexus System

The Nexus system addresses the above limitations in scalability:

- **The Nexus Network.** The network addresses problems (1) and (2), through unbounded SNARK-less recursion and high-speed proof aggregation. In other words, the Nexus zkVM can connect to the Nexus Network and perform massively parallelized proving of unbounded computations in a distributed prover network, which partitions and aggregates proofs incrementally.

- **The Nexus zkVM.** The Nexus zkVM addresses problems (3) and (4), through the NVM and the use of zkVM co-processors, which allow for ASIC-like performance for accelerated instructions in the NVM instruction set. This allows the Nexus zkVM to maintain the developer-friendly CPU abstraction with a small Turing-complete ISA, while allowing ASIC-like extensions on the instruction set. Further, the NVM is extremely simple, auditable and highly modular.

We hope that these two systems combined will allow developers to focus on *declaratively* writing, in any programming language, what they want to prove, and simply receive a proof *within seconds*.

We believe users should not have to think through the complex (and highly non-trivial) security and performance properties behind the 40+ years of zero-knowledge research and related high-performance engineering that power the Nexus system.

3 Background

We briefly review the history of scientific advancements that have led us up to this point in time. As we shall see, we seem to sit at the brink of a time where a new form of computation, *verifiable computation*, becomes available to humanity. We believe it is very few times in a civilization's history that a new form of computation is introduced, and we are excited to be part of this moment.

The Nexus project builds on the shoulders giants: great scientists and mathematicians, and their decades-long work. For an in-depth discussion, see Goldreich's excellent article [Gol93] *A Taxonomy of Proof Systems*, or the summary in Ben-Sasson et al. [BCS16], both upon which we base our discussion.

Interactive Proofs. Goldwasser, Micali, and Rackoff [GMR19] introduced interactive proofs in 1985. In a k-round interactive proof, a probabilistic polynomial time verifier exchanges k messages with an all powerful prover, and then accepts or rejects. $\mathsf{IP}[k]$ is the class of languages with a k-round interactive proof, which generalizes the class NP by allowing *randomness* and *interaction* between the prover and verifier. Independently, Babai introduced Arthur-Merlin games [Bab85]. $\mathsf{AM}[k]$ is the class of languages with a k-round Arthur–Merlin game. Goldwasser and Sipser [GS86] showed that the two models are equally powerful, that is $\mathsf{IP}[k] \subseteq \mathsf{AM}[k+2]$. In 1992, Lund, Fortnow, Karloff, and Nisan [LFKN92] introduced the *sum-check* (interactive proof) protocol [LFKN92] (which now powers most of our system, see Sec. 4.1, and our implementations for the [Set20; KS23b; STW23a] protocols), and Shamir [Sha92], building on this, showed that $\mathsf{IP} = \mathsf{PSPACE}$. The latter was a major result showing that interactive proofs are more powerful than previously thought.

Multi-prover interactive proofs (MIPs). Ben-Or, Goldwasser, and Widgerson [BGKW19] introduced multi-prover interactive proofs in 1988. In a k-round p-prover interactive proof, a probabilistic polynomial-time verifier interacts k times with p non-communicating all-powerful provers, and then accepts or rejects. $\mathsf{MIP}[p, k]$ is the class of languages that have a k-round p-prover interactive proof. In [BGKW19], the authors prove that 2 provers always suffice (i.e. $\mathsf{MIP}[p, k] = \mathsf{MIP}[2, k]$). Fortnow, Rompel, and Sipser [FRS88] show that $\mathsf{MIP}[\mathsf{poly}(n), \mathsf{poly}(n)] \subseteq \mathsf{NEXP}$. Two years later, Babai, Fortnow and Lund [BFL91] showed that $\mathsf{MIP}[2, 1] = \mathsf{NEXP}$.

Probabilistically checkable proofs (PCPs). Probabilistically checkable proofs were introduced by [FRS88; BFLS91; AS92; Aro+92]. In a probabilistically-checkable proof, a probabilistic polynomial-time verifier has oracle access to a proof string; $\mathsf{PCP}[r, q]$ is the class of languages for which the verifier uses at most r bits of randomness, and queries at most q locations of the proof. The above results on MIPs imply that $\mathsf{PCP}[\mathsf{poly}(n), \mathsf{poly}(n)] = \mathsf{NEXP}$. Later works "scaled down" this result to NP: Babai, Fortnow, Levin and Szegedy [BFLS91] show that $\mathsf{NP} = \mathsf{PCP}[O(\log n), \mathsf{poly}(\log n)]$; Arora and Safra [AS92] show that $\mathsf{NP} = \mathsf{PCP}[O(\log n), O(\sqrt{\log n})]$; and Arora, Lund, Motwani, Sudan, and Szegedy [Aro+92] showed that $\mathsf{NP} = \mathsf{PCP}[O(\log n), O(1)]$. This last is known as the PCP Theorem, a major result in complexity theory, and another a paper that would win the Gödel Prize.

Argument systems In cryptography, one often considers *argument systems*, which are IPs where soundness is relaxed from perfect to only computational. This allows circumventing various well-known limitations of IPs [BHZ87; PS05]. Kilian [Kil92] provided the first construction of a succinct interactive argument system by employing PCPs in conjunction with Merkle trees [Mer87]. Micali [Mic94] through his computationally sound proof construction made a similar protocol non-interactive in the random oracle model [BR93] by applying the Fiat-Shamir transform [FS86], thereby obtaining the first zk-SNARK.

The Random Oracle Model (ROM). An idealized model for studying computationally-bounded provers is the Random Oracle Model (ROM) [BR93; CGH04; KM15]. At a high level, the ROM is a model where the prover has access to a random oracle, which is a function that returns random outputs for each input. The ROM is used in conjunction with the Fiat-Shamir transform [FS86] to compile interactive proofs into non-interactive ones, yielding NARKs (non-interactive argument systems) [BCCT12]. See Sec. 4.2 for the definition of NARKs.

zk-SNARKs Many works have obtained SNARK and zk-SNARK constructions [Kil92; Mic94]. After Micali, Lipmaa [DL08] obtained a construction based on added assumptions. Groth [Gro10] obtained the first pairing-based zk-SNARK. In practice, the most widely used zk-SNARK in industry is Groth16 [Gro16] due to its small proof size. Other constructions that have seen success are Halo and Halo2 [BGH20], Plonk [GWC19], Spartan [Set20], Marlin [Chi+20], HyperPlonk [CBBZ23], Bulletproofs [Bün+18], and others [Wah+18; MBKM19; Xie+19; AHIV17; Ben+19; ZXZS20; Gol+21; XZS22].

Interactive Oracle Proofs (IOPs). IOPs were introduced by Ben-Sasson, Chiesa, and Spooner [BCS16]. An IOP is a generalization of IPs, MIPs and PCPs, that can compiled to SNARKs using a suitable commitment scheme. Using a univariate polynomial commitment scheme, one can compile Polynomial-IOPs to SNARKs, yielding protocols like Sonic [MBKM19], Marlin [Chi+20], and Plonk [GWC19]. Using multilinear polynomial commitment schemes, one can combine multilinear-IOPs to SNARKs, yielding protocols like Hyrax [Wah+18], Libra [Xie+19], and Spartan [Set20]. Using vector commitment schemes, one can compile vector-IOPs to SNARKs, yielding protocols like STARK [BBHR18b], Aurora [Ben+19], Virgo [ZXZS20], Brakedown [Gol+21], and Orion [XZS22], generally all known as STARKs. While STARKs have the benefit of requiring no trusted setup and being plausibly post-quantum secure, they have large proof sizes (e.g. tens of kilobytes) and long verification times, and are therefore impractical for recursion.

Incrementally Verifiable Computation (IVC). Incrementally Verifiable Computation (IVC) was introduced in 2008 at MIT by Valiant [Val08]. As mentioned, IVC is the setting in which the prover runs a computation and incrementally updates proofs of correctness for it. Valiant also showed how to construct IVC through the recursive composition of SNARKs.

Proof-Carrying Data (PCD). Proof-Carrying Data (PDC) was introduced by Chiesa and Tromer [CT10], and generalizes IVC to the distributed setting, allowing mutually distrustful parties to perform distributed verifiable computations. In particular, PCD enables the parallel distributed computation of proofs in a DAG, which is sequential in the case of IVC.

PCD from SNARKs. A well-known approach to construct PCD / IVC is to use SNARKs for NP. At each step of the computation, the prover produces a SNARK proving that it has applied a function F correctly and that the SNARK verifier itself, represented as a circuit, has accepted the SNARK proof from the previous step. However, it is well-known that this approach is impractical [BCTV17; CCDW20], as recursive composition of SNARKs incurs in very significant overhead. Alternatively, one can use SNARKs without trusted setup (STARKs), but their verifiers are even more expensive than SNARKs with trusted setup, both asymptotically and concretely. As mentioned above, this makes STARKs impractical for recursion.

SNARKs for Virtual Machines and Cycles of Elliptic Curves. Following the work on IVC by Valiant, and theoretical work in recursive SNARKs [BCCT13], Ben-Sasson, Chiesa, Tromer and Virza [BCTV17] introduced the notion of cycles of elliptic curves (which we use extensively, see [KST22; NBS23; KS23a], Sec. 5.4, and Fig. 2) that are useful (almost necessary, if using KZG [KZG10] or Pedersen [Ped91] commitments, as opposed to FRI [BBHR18a]) to make recursive composition of SNARKs practical. Following this, Ben-Sasson, Chiesa, Tromer, Genkin and Virza built the first implementation of IVC for a simple virtual machine with Harvard architecture, TinyRAM [Ben+13]. The authors later provided the first construction of IVC for a virtual machine with von Neumann architecture, vnTinyRAM [BCTV14]. The vnTinyRAM construction later inspired the Cairo zkVM [GPR21] project, which now powers the Starkware [Sta22] zk-rollup [LNS20]. The vnTinyRAM project is an inspiration for the Nexus zkVM.

Accumulation. A recent line of work proposes the idea of batch verification of **NP** statements to reduce the prover overhead in constructing IVC / PCD. Bowe, Grigg and Hopwood constuct Halo [BGH19] by proposing an approach to recusive proof composition that replaced the SNARK verifier with a simpler *accumulator* algorithm, deferring full verification at the end of the IVC chain. Bünz, Chiesa, Mishra, and Spooner [BCMS20] formally defined *accumulation schemes*, and show they suffice to construct PCD. Following this, [Bün+21] show how to construct PCD from any NARK that satisfies a weak type of accumulation.

Folding Schemes. Folding Schemes were introduced by Kothapalli, Setty and Tzialla [KST22] in 2022 as a way to realize IVC without SNARKs by aggregating proofs for the NP statements themselves directly. Follow up work improves or generalizes Nova, including SuperNova [KS22], HyperNova [KS23b], CycleFold [KS23a], Protostar [BC23], ProtoGalaxy [EG23], PCD from Multi-folding Schemes [ZZD23], and KiloNova [ZGGX23]. The fundamental concept behind folding schemes is to leverage the homomorphic property of commitments (e.g. Pedersen commitments [Ped91]) to accumulate verification of each sequential step of IVC/PCD into a single instance, and so defer full verification at the end of the IVC/PCD chain.

4 Preliminaries

We adapt the definitions from [KST22; KS22; KS23b; KS23a; STW23a; Set20; AST23; BCMS20; Bün+21; BC23; ZZD23].

4.1 The Sum-Check Protocol

Let $g : \mathbb{F}^\ell \to \mathbb{F}$ be an ℓ-variate polynomial defined over a finite field $\mathbb{F}$, where the degree of each variable in g is at most d. Consider the problem of summing up the evaluations of g over the ℓ-dimensional Boolean hypercube

$$T = \sum_{x_1 \in \{0,1\}} \sum_{x_2 \in \{0,1\}} \cdots \sum_{x_\ell \in \{0,1\}} g(x_1, x_2, \ldots, x_\ell),$$

which requires exponential time in ℓ, as it requires evaluating g at $\mathcal{O}(2^\ell)$ points. The *sum-check* protocol [LFKN92] is an interactive protocol allowing a verifier $\mathcal{V}$ to delegate the computation of T to a computationally unbounded prover $\mathcal{P}$, so that $\mathcal{P}$ may convince $\mathcal{V}$ that T is the correct sum, and $\mathcal{V}$ may check this claim in time polynomial in ℓ. In the protocol, $\mathcal{V}$ takes as input randomness $r \in \mathbb{F}^\ell$, and interacts with $\mathcal{P}$ over a sequence of ℓ rounds. At the end of the interaction, $\mathcal{V}$ outputs a claim about the evaluation $g(r)$. We denote the execution of the sum-check protocol as

$$c \leftarrow \langle \mathcal{P}, \mathcal{V}(r) \rangle (g, \ell, d, T),$$

and it satisfies the following properties.

- **Completeness.** If $T = \sum_{x \in \{0,1\}^\ell} g(x)$, then for all $r \in \mathbb{F}^\ell$,
$$\Pr\left[c \leftarrow \langle \mathcal{P}, \mathcal{V}(r) \rangle (g, \ell, d, T) \wedge g(r) = c\right] = 1.$$
- **Soundness.** If $T \neq \sum_{x \in \{0,1\}^\ell} g(x)$, then for all $\mathcal{P}^*$ and all $r \in \mathbb{F}^\ell$,
$$\Pr\left[c \leftarrow \langle \mathcal{P}^*, \mathcal{V}(r) \rangle (g, \ell, d, T) \wedge g(r) = c\right] \leq d \cdot \ell / |\mathbb{F}|$$
- **Succinctness.** The communication cost is $\mathcal{O}(\ell \cdot d)$ elements of $\mathbb{F}$.

4.2 Non-Interactive Arguments of Knowledge (NARKs)

Definition 4.1 (NARK). Let $\mathcal{R}$ be a relation over tuples $(\mathsf{pp}, \mathsf{s}, u, w)$ of public parameters, structures, instances and witnesses. A non-interactive argument of knowledge (NARK) for $\mathcal{R}$ is a four-tuple of PPT algorithms $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$, denoting the generator, encoder, prover, and verifier, respectively, with the following interface.

- $\mathcal{G}(1^\lambda) \to \mathsf{pp}$: Given a security parameter λ, samples public parameters pp.
- $\mathcal{K}(\mathsf{pp}, \mathsf{s}) \to (\mathsf{pk}, \mathsf{vk})$: Given a structure s, outputs a prover key pk and verifier key vk.
- $\mathcal{P}(\mathsf{pk}, u, w) \to \pi$: Given an instance u and witness u, outputs a proof π proving that $(\mathsf{pp}, \mathsf{s}, u, w) \in \mathcal{R}$.
- $\mathcal{V}(\mathsf{vk}, u, \pi) \to \{0,1\}$: Given an instance u and a proof π, outputs 1 (`accept`) or 0 (`reject`).

A NARK scheme $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ must satisfy the following requirements.

- **Completeness**. For any PPT adversary $\mathcal{A}$

$$\Pr\left[\mathcal{V}(\mathsf{vk}, u, \pi) = 1 \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda), \\ (\mathsf{s}, u, w) \leftarrow \mathcal{A}(\mathsf{pp}), \\ (\mathsf{pp}, \mathsf{s}, u, w) \in \mathcal{R}, \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \mathsf{s}), \\ \pi \leftarrow \mathcal{P}(\mathsf{pk}, u, w) \end{array}\right] = 1$$

- **Knowledge Soundness**. For all PPT adversaries $\mathcal{A}$ there exists a PPT extractor $\mathcal{E}$ such that, for all randomness ρ

$$\Pr\left[\begin{array}{l} \mathcal{V}(\mathsf{vk}, u, \pi) = 1, \\ (\mathsf{pp}, \mathsf{s}, u, w) \notin \mathcal{R} \end{array} \middle| \begin{array}{l} \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda), \\ (s, u, \pi) \leftarrow \mathcal{A}(\mathsf{pp}; \rho), \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \mathsf{s}) \\ w \leftarrow \mathcal{E}(\mathsf{pp}; \rho) \end{array}\right] \leq \mathsf{negl}(\lambda)$$

Definition 4.2 (Zero Knowledge)**.** A NARK $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ for a relation $\mathcal{R}$ is zero knowledge if there exists a PPT simulator $\mathcal{S}$ such that for all PPT adversaries $\mathcal{A}$

$$\left\{ (\mathsf{pp}, \mathsf{s}, u, \pi) \middle| \begin{array}{l} \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda), \\ (\mathsf{s}, u, w) \leftarrow \mathcal{A}(\mathsf{pp}), \\ (\mathsf{pp}, \mathsf{s}, u, w) \in \mathcal{R}, \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \mathsf{s}), \\ \pi \leftarrow \mathcal{P}(\mathsf{pk}, u, w) \end{array} \right\} \cong \left\{ (\mathsf{pp}, \mathsf{s}, u, \pi) \middle| \begin{array}{l} (\mathsf{pp}, \tau) \leftarrow \mathcal{S}(1^\lambda), \\ (\mathsf{s}, u, w) \leftarrow \mathcal{A}(\mathsf{pp}), \\ (\mathsf{pp}, \mathsf{s}, u, w) \in \mathcal{R}, \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \mathsf{s}), \\ \pi \leftarrow \mathcal{S}(\mathsf{pp}, u, \tau) \end{array} \right\}$$

Definition 4.3 (Succinctness)**.** A NARK is succinct, if the size of the proof π and the verifier's time and space complexity are at most polylogarithmic in the size of the witness w.

4.3 Incrementally Verifiable Computation (IVC)

Definition 4.4 (IVC)**.** An incrementally verifiable computation (IVC) scheme is a four-tuple of PPT algorithms $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$, denoting the generator, encoder, prover, and verifier, respectively, with the following interface.

- $\mathcal{G}(1^\lambda) \rightarrow \mathsf{pp}$: Given a security parameter λ, samples public parameters pp.
- $\mathcal{K}(\mathsf{pp}, F) \rightarrow (\mathsf{pk}, \mathsf{vk})$: Given a polynomial-time function F, outputs a prover key pk and a verifier key vk.
- $\mathcal{P}(\mathsf{pk}, (i, z_0, z_i), \omega_i, \Pi_i) \rightarrow \Pi_{i+1}$: Given a counter i, an initial input z_0, a claimed output after i iterations z_i, a non-deterministic advice ω_i, and an IVC proof Π_i, produces a new IVC proof Π_{i+1} attesting to $z_{i+1} = F(z_i, \omega_i)$.
- $\mathcal{V}(\mathsf{vk}, (i, z_0, z_i), \Pi_i) \rightarrow \{0, 1\}$: Given a counter i, an initial input z_0, a claimed output after i iterations z_i, and an IVC proof Π_i attesting to z_i, outputs 1 (`accept`) or 0 (`reject`).

An IVC scheme $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ must satisfy the following properties.

- **Perfect Completeness**. For any PPT adversary $\mathcal{A}$

$$\Pr\left[\mathcal{V}(\mathsf{vk}, (i+1, z_0, z_{i+1}), \Pi_{i+1}) = 1 \middle| \begin{array}{l} \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda), \\ (F, (i, z_0, z_i), \omega_i, \Pi_i) \leftarrow \mathcal{A}(\mathsf{pp}), \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, F), \\ z_{i+1} \leftarrow F(z_i, \omega_i), \\ \mathcal{V}(\mathsf{vk}, (i, z_0, z_i), \Pi_i) = 1, \\ \Pi_{i+1} \leftarrow \mathcal{P}(\mathsf{pk}, (i, z_0, z_i), \omega_i, \Pi_i) \end{array}\right] = 1$$

- **Knowledge Soundness**. For any constant $n \in \mathbb{N}$, and for any PPT adversaries $\mathcal{P}^*$ there exists a PPT extractor $\mathcal{E}$ such that, for all randomness ρ

$$\Pr\left[\begin{array}{l} z_n \neq z, \\ \mathcal{V}(\mathsf{vk}, (n, z_0, z), \Pi) = 1 \end{array} \middle| \begin{array}{l} \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda), \\ (F, (z_0, z, \Pi)) \leftarrow \mathcal{P}^*(\mathsf{pp}; \rho), \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, F), \\ (\omega_0, \ldots, \omega_{n-1}) \leftarrow \mathcal{E}(\mathsf{pp}; \rho), \\ z_{i+1} \leftarrow F(z_i, \omega_i) \quad \forall i \in \{0, \ldots, n-1\} \end{array}\right] \leq \mathsf{negl}(\lambda)$$

- **Succinctness**. The size of an IVC proof Π is independent of the number of iterations n.

4.4 Non-uniform Incrementally Verifiable Computation (NIVC)

Non-uniform IVC (NIVC) generalizes IVC to handle an arbitrary number of polynomial-time functions $(F_1, \dots, F_\ell)$, where the choice of which F_j for $j \in [\ell]$ is executed at each step i is determined by an additional polynomial-time control function φ, invoked as $j \leftarrow \varphi(z_{i-1}, \omega_{i-1})$.

More succinctly, for a given $(\varphi, (F_1, \dots, F_\ell))$ and (n, z_0, z_n) an NIVC scheme allows the prover to prove knowledge of intermediate outputs $(z_1, \dots, z_{n-1})$ and non-deterministic advice $(\omega_0, \dots, \omega_{n-1})$ such that

$$z_{i+1} = F_{\varphi(z_i,\omega_i)}(z_i, \omega_i)$$

for all $i \in \{0, \dots, n-1\}$.

Definition 4.5 (NIVC)**.** A non-uniform incrementally verifiable computation (NIVC) scheme is a four-tuple of PPT algorithms $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$, denoting the generator, encoder, prover, and verifier, respectively, with the following interface.

- $\mathcal{G}(1^\lambda) \to \mathsf{pp}$: Given a security parameter λ, samples public parameters pp.
- $\mathcal{K}(\mathsf{pp}, (\varphi, (F_1, \dots, F_\ell))) \to (\mathsf{pk}, \mathsf{vk})$: Given a control function φ, and functions $(F_1, \dots, F_\ell)$, outputs a prover key pk and verifier key vk, where $\ell \geq 1$, φ produces an element in $\{1, \dots, \ell\}$, and φ and F_j for $j \in \{1, \dots, \ell\}$ are polynomial-time computable functions.
- $\mathcal{P}(\mathsf{pk}, (i, z_0, z_i), \omega_i, \Pi_i) \to \Pi_{i+1}$: Given a counter i, an initial input z_0, a claimed output after i applications z_i, a non-deterministic advice ω_i, and an NIVC proof Π_i attesting to z_i, outputs a new proof Π_{i+1} attesting to $z_{i+1} = F_{\varphi(z_i,\omega_i)}(z_i, \omega_i)$.
- $\mathcal{V}(\mathsf{vk}, (i, z_0, z_i), \Pi_i) \to \{0, 1\}$: Given a counter i, an initial input z_0, a claimed output after i applications z_i, and an NIVC proof Π_i, outputs 1 (`accept`) or 0 (`reject`).

An NIVC scheme $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ must satisfy the following properties.

- **Perfect Completeness.** For any PPT adversary $\mathcal{A}$

$$\Pr\left[\mathcal{V}(\mathsf{vk}, (i+1, z_0, z_{i+1}), \Pi_{i+1}) = 1 \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda), \\ (\varphi, (F_1, \dots, F_\ell), (i, z_0, z_i), \omega_i, \Pi_i) \leftarrow \mathcal{A}(\mathsf{pp}), \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, (\varphi, (F_1, \dots, F_\ell))), \\ \mathcal{V}(\mathsf{vk}, (i, z_0, z_i), \Pi_i) = 1, \\ z_{i+1} \leftarrow F_{\varphi(z_i,\omega_i)}(z_i, \omega_i), \\ \Pi_{i+1} \leftarrow \mathcal{P}(\mathsf{pk}, (i, z_0, z_i), \omega_i, \Pi_i) \end{array}\right] = 1$$

- **Knowledge Soundness.** For any constant $n \in \mathbb{N}$, and for any PPT adversaries $\mathcal{P}^*$, there exists a PPT extractor $\mathcal{E}$ such that, for all randomness ρ

$$\Pr\left[\begin{array}{l} z \neq z_n \\ \mathcal{V}(\mathsf{vk}, (n, z_0, z), \Pi) = 1 \end{array} \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda), \\ (\varphi, (F_1, \dots, F_\ell), (z_0, z, \Pi)) \leftarrow \mathcal{P}^*(\mathsf{pp}; \rho), \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, (\varphi, (F_1, \dots, F_\ell))), \\ (\omega_0, \dots, \omega_{n-1}) \leftarrow \mathcal{E}(\mathsf{pp}; \rho), \\ z_{i+1} \leftarrow F_{\varphi(z_i,\omega_i)}(z_i, \omega_i), \quad \forall i \in \{0, \dots, n-1\} \end{array}\right] \leq \mathsf{negl}(\lambda)$$

- **Succinctness.** The size of an IVC proof Π is independent of the number of applications n.
- **Efficiency.** The prover's time and space complexity are linear in the size of the function applied at step i, i.e. $\mathcal{O}(|F_{\varphi(z_i,\omega_i)}|)$.

4.5 Proof-Carrying Data

Definition 4.6 (PCD Transcript)**.** We define a PCD transcript T to be a directed acyclic graph where each vertex $v \in V(\mathsf{T})$ is labeled by local data $\omega^{(v)}$ and each edge $e \in E(\mathsf{T})$ is labeled by a message $z^{(e)}$. We define the output of a transcript $\mathsf{o}(\mathsf{T})$ as the message $z^{(e)}$ of the lexicographically-first edge $e = (u, v)$ such that v is a sink.

Definition 4.7 (PCD Compliance)**.** Let φ be a predicate. We define a vertex $u \in V(\mathsf{T})$ to be φ-compliant if for all outgoing edges $e = (u, v) \in E(\mathsf{T})$:

- If u has no incoming edges, then $\varphi(z^{(e)}, \omega^{(u)}, [\perp, \dots, \perp]) = 0$.
- If u has incoming edges $[e_i]_{i=1}^{r}$, then $\varphi(z^{(e)}, \omega^{(u)}, [z^{(e_i)}]_{i=1}^{r}) = 1$.

We say T is φ-compliant, if all its vertices are φ-compliant. We use the notation $\varphi(\mathsf{T}) \to \{0, 1\}$ to denote the φ-compliance of T.

Definition 4.8 (PCD [Bün+21])**.** A proof-carrying data (PCD) scheme is a four-tuple of PPT algorithms $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ denoting the generator, encoder, prover, and verifier, respectively, with the following interface.

- $\mathcal{G}(1^\lambda) \to \mathsf{pp}$: Given a security parameter λ, samples public parameters pp.
- $\mathcal{K}(\mathsf{pp}, \varphi) \to (\mathsf{pk}, \mathsf{vk})$: Given a compliance predicate φ, outputs a prover key pk and a verifier key vk.
- $\mathcal{P}(\mathsf{pk}, z, \omega, [z_i, \Pi_i]_{i=1}^{r}) \to \Pi$: Given a message z, local data ω, messages $[z_i]_{i=1}^{r}$ and corresponding PCD proofs $[\Pi_i]_{i=1}^{r}$, outputs a new proof Π attesting to $\varphi(z, \omega, [z_i]_{i=1}^{r}) = 1$.
- $\mathcal{V}(\mathsf{vk}, z, \Pi) \to \{0, 1\}$: Given a message z and a PCD proof Π, outputs 1 (`accept`) or 0 (`reject`).

A PCD scheme $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ must satisfy the following properties.

- **Perfect Completeness.** For any PPT adversary $\mathcal{A}$

$$\Pr\left[\mathcal{V}(\mathsf{vk}, z, \Pi) = 1 \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda), \\ (\varphi, z, \omega, [z_i, \Pi_i]_{i=1}^{r}]) \leftarrow \mathcal{A}(\mathsf{pp}), \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \varphi), \\ \varphi(z, \omega, [z_i]_{i=1}^{r}) = 1, \\ \forall i \in [r], \quad \mathcal{V}(\mathsf{vk}, z_i, \Pi_i) = 1, \\ \Pi \leftarrow \mathcal{P}(\mathsf{pk}, z, \omega, [z_i, \Pi_i]_{i=1}^{r}) \end{array}\right] = 1$$

- **Knowledge Soundness.** For any PPT adversaries $\mathcal{P}^*$, there exists a PPT extractor $\mathcal{E}$ such that, for all randomness ρ

$$\Pr\left[\begin{array}{l} \mathcal{V}(\mathsf{vk}, z, \Pi) = 1, \\ \mathsf{o}(\mathsf{T}) = z \wedge \varphi(\mathsf{T}) = 0 \end{array} \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda), \\ (\varphi, z, \Pi) \leftarrow \mathcal{P}^*(\mathsf{pp}; \rho), \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \varphi), \\ \mathsf{T} \leftarrow \mathcal{E}(\mathsf{pp}; \rho) \end{array}\right] \leq \mathsf{negl}(\lambda)$$

- **Succinctness.** The size of a PCD proof Π is independent of the size of T.

4.6 Folding Schemes

Definition 4.9 (Folding Scheme [KST22]). Let $\mathcal{R}$ be a relation over tuples $(\mathsf{pp}, \mathsf{s}, u, w)$ of public parameters, structures, instances and witnesses. A folding scheme for $\mathcal{R}$ is a four-tuple of PPT algorithms $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$, denoting the generator, encoder, prover, and verifier, respectively, with the following interface.

- $\mathcal{G}(1^\lambda) \to \mathsf{pp}$: Given a security parameter λ, samples public parameters pp.
- $\mathcal{K}(\mathsf{pp}, \mathsf{s}) \to (\mathsf{pk}, \mathsf{vk})$: Given a structure s, outputs a prover key pk and a verifier key vk.
- $\mathcal{P}(\mathsf{pk}, (u_1, w_1), (u_2, w_2)) \to (u, w)$: Given instance-witness tuples (u_1, w_1) and (u_2, w_2), outputs a new instance withness tuple (u, w) of the same size.
- $\mathcal{V}(\mathsf{vk}, u_1, u_2) \to u$: Given instances u_1 and u_2, outputs a new instance u.

Let $\langle \mathcal{P}, \mathcal{V} \rangle$ denote a function computing the interaction between $\mathcal{P}$ and $\mathcal{V}$, invoked as

$$(u, w) \leftarrow \langle \mathcal{P}, \mathcal{V} \rangle((\mathsf{pk}, \mathsf{vk}), (u_1, u_2), (w_1, w_2))$$

where $\langle \mathcal{P}, \mathcal{V} \rangle$ runs the interaction on prover input $(\mathsf{pk}, (u_1, w_1), (u_2, w_2))$ and verifier input (vk, u_1, u_2), and where u is the verifier's output folded instance and w is the prover's output folded witness.

A folding scheme $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ is required to satisfy the following requirements:

- **Perfect Completeness**. For any PPT adversary $\mathcal{A}$

$$\Pr\left[(\mathsf{pp}, \mathsf{s}, u, w) \in \mathcal{R} \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda), \\ (\mathsf{s}, (u_1, w_1), (u_2, w_2)) \leftarrow \mathcal{A}(\mathsf{pp}), \\ (\mathsf{pp}, \mathsf{s}, u_1, w_1), (\mathsf{pp}, \mathsf{s}, u_2, w_2) \in \mathcal{R}, \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \mathsf{s}), \\ (u, w) \leftarrow \langle \mathcal{P}, \mathcal{V} \rangle((\mathsf{pk}, \mathsf{vk}), (u_1, u_2), (w_1, w_2)) \end{array} \right] = 1$$

- **Knowledge Soundness**. For any PPT adversaries $\mathcal{A}$ and $\mathcal{P}^*$ there exists a PPT extractor $\mathcal{E}$ such that, for all randomness ρ

$$\Pr\left[(\mathsf{pp}, \mathsf{s}, u, w) \in \mathcal{R} \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda) \\ (\mathsf{s}, (u_1, u_2), \tau) \leftarrow \mathcal{A}(\mathsf{pp}; \rho) \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \mathsf{s}) \\ (u, w) \leftarrow \langle \mathcal{P}^*, \mathcal{V} \rangle((\mathsf{pk}, \mathsf{vk}), (u_1, u_2), \tau) \end{array} \right] -$$
$$\Pr\left[\begin{array}{l} (\mathsf{pp}, \mathsf{s}, u_1, w_1) \in \mathcal{R} \\ (\mathsf{pp}, \mathsf{s}, u_2, w_2) \in \mathcal{R} \end{array} \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda) \\ (\mathsf{s}, (u_1, u_2), \tau) \leftarrow \mathcal{A}(\mathsf{pp}; \rho) \\ (w_1, w_2) \leftarrow \mathcal{E}(\mathsf{pp}; \rho) \end{array} \right] \le \mathsf{negl}(\lambda)$$

- **Efficiency**. The communication costs and $\mathcal{V}$'s computation are lower when $\mathcal{P}$ and $\mathcal{V}$ participate in the folding scheme, than the case in which $\mathcal{V}$ checks the witnesses individually for the original instances.

Let

$$\mathsf{tr} \leftarrow \mathsf{trace}(\langle \mathcal{P}, \mathcal{V} \rangle, \mathsf{input})$$

denote the interaction transcript of running $\langle \mathcal{P}, \mathcal{V} \rangle$ on input input.

Definition 4.10 (Zero Knowledge)**.** A folding scheme $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ for $\mathcal{R}$ satisfies zero knowledge if there exists a PPT simulator $\mathcal{S}$ such that for all PPT adversaries $\mathcal{A}$ and $\mathcal{V}^*$, and all randomness ρ

$$\left\{ \mathsf{tr} \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda), \\ (\mathsf{s}, (u_1, w_1), (u_2, w_2)) \leftarrow \mathcal{A}(\mathsf{pp}), \\ (\mathsf{pp}, \mathsf{s}, u_1, w_1), (\mathsf{pp}, \mathsf{s}, u_2, w_2) \in \mathcal{R}, \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \mathsf{s}), \\ \mathsf{tr} \leftarrow \mathsf{trace}(\langle \mathcal{P}, \mathcal{V}^*(\rho) \rangle, ((\mathsf{pk}, \mathsf{vk}), (u_1, u_2), (w_1, w_2))) \end{array} \right\} \cong$$

$$\left\{ \mathsf{tr} \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda), \\ (\mathsf{s}, (u_1, w_1), (u_2, w_2)) \leftarrow \mathcal{A}(\mathsf{pp}), \\ (\mathsf{pp}, \mathsf{s}, u_1, w_1), (\mathsf{pp}, \mathsf{s}, u_2, w_2) \in \mathcal{R}, \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \mathsf{s}), \\ \mathsf{tr} \leftarrow \mathcal{S}((\mathsf{pk}, \mathsf{vk}), (u_1, u_2); \rho) \end{array} \right\}$$

Definition 4.11 (Non-Interactive)**.** A folding scheme $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ is non-interactive if the interaction between $\mathcal{P}$ and $\mathcal{V}$ consists of a single message from $\mathcal{P}$ to $\mathcal{V}$. In this case, we update the function signatures of $\mathcal{P}$ and $\mathcal{V}$ to include the message as an output of $\mathcal{P}$ and as an input to $\mathcal{V}$, as follows:

- $\mathcal{P}(\mathsf{pk}, (u_1, w_1), (u_2, w_2)) \rightarrow (u, w, \pi)$
- $\mathcal{V}(\mathsf{vk}, (u_1, u_2), \pi) \rightarrow u$

Definition 4.12 (Public-coin)**.** A folding scheme $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ is called public-coin if all messages sent from $\mathcal{V}$ to $\mathcal{P}$ are sampled from a uniform distribution.

4.7 Multi-Folding Schemes

Definition 4.13 (Multi-Folding schemes [KS23b])**.** Let $\mathcal{R}_1$ and $\mathcal{R}_2$ be relations over tuples $(\mathsf{pp}, \mathsf{s}, u, w)$ of public parameters, structures, instances and witnesses, let compat be a predicate that $\mathcal{R}_1$ and $\mathcal{R}_2$ structures should satisfy, and let $\mu, \nu \in \mathbb{N}$ be size parameters. A multi-folding scheme Π_{MFS} for $(\mathcal{R}_1, \mathcal{R}_2, \mathsf{compat}, \mu, \nu)$ is a four-tuple of PPT algorithms $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ denoting the generator, encoder, prover, and verifier, with the following interface:

- $\mathcal{G}(1^\lambda) \rightarrow \mathsf{pp}$: Given a security parameter λ, samples public parameters pp.
- $\mathcal{K}(\mathsf{pp}, (\mathsf{s}_1, \mathsf{s}_2)) \rightarrow (\mathsf{pk}, \mathsf{vk})$: Given public parameters pp and structures $(\mathsf{s}_1, \mathsf{s}_2)$, outputs a prover key pk and a verifier key vk.
- $\mathcal{P}(\mathsf{pk}, (\vec{u}_1, \vec{w}_1), (\vec{u}_2, \vec{w}_2)) \rightarrow (u, w)$: Given an instance-witness vector pair $(\vec{u}_1, \vec{w}_1)$ of length μ with elements of structure s_1 in $\mathcal{R}_1$, and an instance-witness vector pair $(\vec{u}_1, \vec{w}_1)$ of length ν with elements of structure s_2 in $\mathcal{R}_2$, outputs an instance-witness pair (u, w) in $\mathcal{R}_1$.
- $\mathcal{V}(\mathsf{vk}, (\vec{u}_1, \vec{u}_2)) \rightarrow u$: Given a vector of instances $\vec{u}_1$ of length μ with elements in $\mathcal{R}_1$ and a vector of instances $\vec{u}_2$ of length ν with elements in $\mathcal{R}_2$, outputs a new folded instance u in $\mathcal{R}_1$.

Let $\langle \mathcal{P}, \mathcal{V} \rangle$ denote the the interaction between $\mathcal{P}$ and $\mathcal{V}$ treated as a function that takes input $((\mathsf{pk}, \mathsf{vk}), (\vec{u}_1, \vec{u}_2), (\vec{w}_1, \vec{w}_2)$ and runs the prover-verifier interaction on prover input $(\mathsf{pk}, (\vec{u}_1, \vec{w}_1), (\vec{u}_2, \vec{w}_2))$ and verifier input $(\mathsf{vk}, (\vec{u}_1, \vec{u}_2))$. At the end of the interaction, $\langle \mathcal{P}, \mathcal{V} \rangle$ outputs the folded instance-witness tuple (u, w).

A multi-folding scheme $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ for $(\mathcal{R}_1, \mathcal{R}_2, \mathsf{compat}, \mu, \nu)$ must satisfy the following requirements.

- **Perfect Completeness**. For all PPT adversaries $\mathcal{A}$:

$$\Pr\left[(\mathsf{pp}, \mathsf{s}_1, u, w) \in \mathcal{R}_1 \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda), \\ ((\mathsf{s}_1, \mathsf{s}_2), (\vec{u}_1, \vec{u}_2), (\vec{w}_1, \vec{w}_2)) \leftarrow \mathcal{A}(\mathsf{pp}), \\ \mathsf{compat}(\mathsf{s}_1, \mathsf{s}_2) = 1, \\ (\mathsf{pp}, \mathsf{s}_1, \vec{u}_1, \vec{w}_1), \in \mathcal{R}_1^{(\mu)}, \\ (\mathsf{pp}, \mathsf{s}_2, \vec{u}_2, \vec{w}_2), \in \mathcal{R}_2^{(\nu)}, \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, (\mathsf{s}_1, \mathsf{s}_2)), \\ (u, w) \leftarrow \langle \mathcal{P}, \mathcal{V} \rangle((\mathsf{pk}, \mathsf{vk}), (\vec{u}_1, \vec{u}_2), (\vec{w}_1, \vec{w}_2)) \end{array} \right] = 1$$

- **Knowledge Soundness**. For all PPT adversaries $\mathcal{A}$ and $\mathcal{P}^*$ there exists a PPT extractor $\mathcal{E}$ such that, for all randomness ρ

$$\Pr\left[(\mathsf{pp}, \mathsf{s}_1, u, w) \in \mathcal{R}_1 \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda), \\ ((\mathsf{s}_1, \mathsf{s}_2), (\vec{u}_1, \vec{u}_2), \tau) \leftarrow \mathcal{A}(\mathsf{pp}; \rho) \\ \mathsf{compat}(\mathsf{s}_1, \mathsf{s}_2) = 1, \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, (\mathsf{s}_1, \mathsf{s}_2)), \\ (u, w) \leftarrow \langle \mathcal{P}^*, \mathcal{V} \rangle((\mathsf{pk}, \mathsf{vk}), (\vec{u}_1, \vec{u}_2), \tau) \end{array} \right] -$$

$$\Pr\left[\begin{array}{l} (\mathsf{pp}, \mathsf{s}_1, \vec{u}_1, \vec{w}_1) \in \mathcal{R}_1^{(\mu)} \\ (\mathsf{pp}, \mathsf{s}_2, \vec{u}_2, \vec{w}_2) \in \mathcal{R}_2^{(\nu)} \end{array} \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda), \\ ((\mathsf{s}_1, \mathsf{s}_2), (\vec{u}_1, \vec{u}_2)) \leftarrow \mathcal{A}(\mathsf{pp}; \rho), \\ \mathsf{compat}(\mathsf{s}_1, \mathsf{s}_2) = 1, \\ (\vec{w}_1, \vec{w}_2) \leftarrow \mathcal{E}(\mathsf{pp}; \rho) \end{array} \right] \le \mathsf{negl}(\lambda)$$

- **Efficiency**. The communication and computation costs are lower in the multi-folding scheme than checking the instance-witness pairs individually.

Definition 4.14 (Zero Knowledge)**.** A multi-folding scheme $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ for $(\mathcal{R}_1, \mathcal{R}_2, \mathsf{compat}, \mu, \nu)$ satisfies zero-knowledge if there exists a PPT simulator $\mathcal{S}$ such that for all PPT adversaries $\mathcal{A}$ and $\mathcal{V}^*$, and all randomness ρ

$$\left\{ \mathsf{tr} \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda), \\ ((\mathsf{s}_1, \mathsf{s}_2), (\vec{u}_1, \vec{w}_1), (\vec{u}_2, \vec{w}_2)) \leftarrow \mathcal{A}(\mathsf{pp}), \\ \mathsf{compat}(\mathsf{s}_1, \mathsf{s}_2) = 1, \\ (\mathsf{pp}, \mathsf{s}_1, \vec{u}_1, \vec{w}_1) \in \mathcal{R}_1^{(\mu)}, \\ (\mathsf{pp}, \mathsf{s}_2, \vec{u}_2, \vec{w}_2) \in \mathcal{R}_2^{(\nu)}, \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, (\mathsf{s}_1, \mathsf{s}_2)), \\ \mathsf{tr} \leftarrow \mathsf{trace}(\langle \mathcal{P}, \mathcal{V}^*(\rho) \rangle, ((\mathsf{pk}, \mathsf{vk}), (\vec{u}_1, \vec{u}_2), (\vec{w}_1, \vec{w}_2))) \end{array} \right\} \cong$$

$$\left\{ \mathsf{tr} \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda), \\ (\mathsf{s}, (\vec{u}_1, \vec{w}_1), (\vec{u}_2, \vec{w}_2)) \leftarrow \mathcal{A}(\mathsf{pp}), \\ \mathsf{compat}(\mathsf{s}_1, \mathsf{s}_2) = 1, \\ (\mathsf{pp}, \mathsf{s}_1, \vec{u}_1, \vec{w}_1) \in \mathcal{R}_1^{(\mu)}, \\ (\mathsf{pp}, \mathsf{s}_2, \vec{u}_2, \vec{w}_2) \in \mathcal{R}_2^{(\nu)}, \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, (\mathsf{s}_1, \mathsf{s}_2)), \\ \mathsf{tr} \leftarrow \mathcal{S}((\mathsf{pk}, \mathsf{vk}), (\vec{u}_1, \vec{u}_2); \rho) \end{array} \right\}$$

Definition 4.15 (Non-Interactive)**.** A multi-folding scheme $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ is non-interactive if the interaction between $\mathcal{P}$ and $\mathcal{V}$ consists of a single message from $\mathcal{P}$ to $\mathcal{V}$. In this case, we update the function signatures of $\mathcal{P}$ and $\mathcal{V}$ to include the message as an output of $\mathcal{P}$ and as an input to $\mathcal{V}$, as follows:

- $\mathcal{P}(\mathsf{pk}, (\vec{u}_1, \vec{w}_1), (\vec{u}_2, \vec{w}_2)) \to (u, w, \pi)$
- $\mathcal{V}(\mathsf{vk}, (\vec{u}_1, \vec{u}_2), \pi) \to u$

Definition 4.16 (Public-coin)**.** A multi-folding scheme $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ is called public-coin if all messages sent from $\mathcal{V}$ to $\mathcal{P}$ are sampled from a uniform distribution.

4.8 Customizable Constraint Systems

4.8.1 The CCS Relation

Definition 4.17 (CCS [STW23a])**.** Let $\mathcal{R}_{\mathsf{CCS}}$ be the customizable constraint system relation (CCS), defined as follows. An $\mathcal{R}_{\mathsf{CCS}}$ structure

$$\mathsf{s}_{\mathsf{CCS}} = (m, n, N, \ell, t, q, d, [M_i]_{i=1}^{t}, [S_i]_{i=1}^{q}, [c_i]_{i=1}^{q}])$$

consists of:

- Size bounds $m, n, N, \ell, t, q, d \in \mathbb{N}$, where $n > \ell$.
- A sequence of t matrices $[M_1, \ldots, M_t]$ each in $\mathbb{F}^{m \times n}$, with at most $N = \Omega(\max(m, n))$ non-zero entries.
- A sequence of q multisets $[S_1, \ldots, S_q]$ each of cardinality at most d over the domain $\{1, \ldots, t\}$.
- A sequence of q constants $[c_1, \ldots, c_q]$ each in $\mathbb{F}$.

An $\mathcal{R}_{\mathsf{CCS}}$ instance x consists of public input $x \in \mathbb{F}^{\ell}$. An $\mathcal{R}_{\mathsf{CCS}}$ witness consists of a vector $w \in \mathbb{F}^{n-\ell-1}$. A CCS structure-instance tuple (s, x) is satisfied by a CCS witness w if

$$\sum_{i=1}^{q} c_i \cdot \bigcirc_{j \in S_i} M_j \cdot z = \mathbf{0}$$

where $z = (w, 1, x) \in \mathbb{F}^n$, $M_j \cdot z$ denotes matrix-vector multiplication, $\bigcirc$ denotes the Hadamard (entry-wise) product, and $\mathbf{0}$ is the zero vector in $\mathbb{F}^m$.

4.8.2 The Committed CCS Relation

Consider a CCS structure

$$\mathsf{s}_{\mathsf{CCS}} = (m, n, N, \ell, t, q, d, [M_i]_{i=1}^{t}, [S_i]_{i=1}^{q}, [c_i]_{i=1}^{q}])$$

Let $s = \lceil \log m \rceil$, $s' = \lceil \log n \rceil$ and $s'' = \lceil \log(n - \ell - 1) \rceil$. By interpreting each M_i as functions of type $\{0,1\}^s \times \{0,1\}^{s'} \to \mathbb{F}$, let $\widetilde{M}_i$ denote the MLE of M_i. Similarly, for a purported witness $w \in \mathbb{F}^{n-\ell-1}$ let $\widetilde{w}$ denote the MLE of w viewed as a function of type $\{0,1\}^{s''} \to \mathbb{F}$.

Definition 4.18 (Committed CCS)**.** Let $\mathcal{R}_{\mathsf{CCCS}}$ be the committed customizable constraint system (CCCS) relation, defined as follows. Let PC be an additively-homomorphic polynomial commitment scheme for multilinear polynomials over a finite field $\mathbb{F}$.

An $\mathcal{R}_{\mathsf{CCCS}}$ structure

$$\mathsf{s}_{\mathsf{CCCS}} = (m, n, N, \ell, t, q, d, \mathsf{pp}, [\widetilde{M}_i]_{i=1}^{t}, [S_i]_{i=1}^{q}, [c_i]_{i=1}^{q})$$

consists of:

- Size bounds $m, n, N, \ell, t, q, d \in \mathbb{N}$ and $\mathsf{pp} \leftarrow \mathsf{PC.Setup}(1^\lambda, s'')$, where $n > \ell$ and let $s = \lceil \log m \rceil$, $s' = \lceil \log n \rceil$, $s'' = \lceil \log(n - \ell - 1) \rceil$.

- A sequence of t multilinear polynomials $[\widetilde{M}_1, \ldots, \widetilde{M}_t]$ each in $\mathbb{F}[X_1, \ldots, X_{s+s'}]$ over $s + s'$ variables, with at most $N = \Omega(m)$ locations over the Boolean hypercube $\{0,1\}^{s+s'}$ evaluating to a non-zero value.

- A sequence of q multisets $[S_1, \ldots, S_q]$ each of cardinality at most d over the domain $\{1, \ldots, t\}$.

- A sequence of q constants $[c_1, \ldots, c_q]$ each in $\mathbb{F}$.

An $\mathcal{R}_{\mathsf{CCCS}}$ instance (C, x) consists of a commitment C to an s''-variate multilinear polynomial and public input $\mathsf{x} \in \mathbb{F}^{\ell}$. An $\mathcal{R}_{\mathsf{CCCS}}$ witness is an s''-variate multilinear polynomial $\widetilde{w}$.

An $\mathcal{R}_{\mathsf{CCCS}}$ structure-instance tuple $(\mathsf{s}, (C, \mathsf{x}))$ is satisfied by an $\mathcal{R}_{\mathsf{CCCS}}$ witness $\widetilde{w}$ if $\mathsf{PC.Com}(\mathsf{pp}, \widetilde{w}) = C$ and for all $x \in \{0,1\}^s$,

$$\sum_{i=1}^{q} c_i \cdot \left(\prod_{j \in S_i} \left(\sum_{y \in \{0,1\}^{s'}} \widetilde{M}_j(x, y) \cdot \widetilde{z}(y) \right) \right) = 0$$

where $\widetilde{z}$ is the s'-variate MLE of $z = (w, 1, \mathsf{x})$ viewed as a function of type $\{0,1\}^{s'} \to \mathbb{F}$.

4.8.3 The Linearized Committed CCS Relation

Definition 4.19 (Linearized committed CCS)**.** Let $\mathcal{R}_{\mathsf{LCCCS}}$ be the linearized committed customizable constraint system (LCCCS) relation, defined as follows. Let PC be an additively-homomorphic polynomial commitment scheme for multilinear polynomials over a finite field $\mathbb{F}$.

An $\mathcal{R}_{\mathsf{LCCCS}}$ structure

$$\mathsf{s}_{\mathsf{LCCCS}} = (m, n, N, \ell, t, q, d, \mathsf{pp}, [\widetilde{M}_i]_{i=1}^{t}, [S_i]_{i=1}^{q}, [c_i]_{i=1}^{q})$$

consists of:

- Size bounds $m, n, N, \ell, t, q, d \in \mathbb{N}$ and $\mathsf{pp} \leftarrow \mathsf{PC.Setup}(1^{\lambda}, s'')$, where $n > \ell$ and let $s = \lceil \log m \rceil$, $s' = \lceil \log n \rceil$, $s'' = \lceil \log(n - \ell - 1) \rceil$.

- A sequence of t multilinear polynomials $[\widetilde{M}_1, \ldots, \widetilde{M}_t]$ each in $\mathbb{F}[X_1, \ldots, X_{s+s'}]$ over $s + s'$ variables, with at most $N = \Omega(m)$ locations over the Boolean hypercube $\{0,1\}^{s+s'}$ evaluating to a non-zero value.

- A sequence of q multisets $[S_1, \ldots, S_q]$ each of cardinality at most d over the domain $\{1, \ldots, t\}$.

- A sequence of q constants $[c_1, \ldots, c_q]$ each in $\mathbb{F}$.

An $\mathcal{R}_{\mathsf{LCCCS}}$ instance $(C, u, \mathsf{x}, r, [v_i]_{i=1}^{t}])$ consists of a commitment C to an s''-variate multilinear polynomial, public input $\mathsf{x} \in \mathbb{F}^{\ell}$, and $u \in \mathbb{F}$, $r \in \mathbb{F}^s$, $v_1, \ldots, v_t \in \mathbb{F}$. An $\mathcal{R}_{\mathsf{LCCCS}}$ witness is an s''-variate multilinear polynomial $\widetilde{w}$.

An $\mathcal{R}_{\mathsf{LCCCS}}$ structure-instance pair $(\mathsf{s}, (C, u, \mathsf{x}, r, [v_i]_{i=1}^{t}))$ is satisfied by an $\mathcal{R}_{\mathsf{LCCCS}}$ witness $\widetilde{w}$ if $\mathsf{PC.Com}(\mathsf{pp}, \widetilde{w}) = C$ and $\forall i \in [t]$:

$$v_i = \sum_{y \in \{0,1\}^{s'}} \widetilde{M}_i(r, y) \cdot \widetilde{z}(y),$$

where $\widetilde{z}$ is the s'-variate MLE of $z = (w, u, \mathsf{x})$ viewed as a function of type $\{0,1\}^{s'} \to \mathbb{F}$.

5 Components

5.1 A Multi-Folding Scheme for CCS

Construction 5.1 (A Multi-Folding scheme for CCS [KS23b]). Let PC be an additively-homomorphic polynomial commitment scheme for multilinear polynomials over a finite field $\mathbb{F}$. We construct a multi-folding scheme for $(\mathcal{R}_{\mathsf{LCCCS}}, \mathcal{R}_{\mathsf{CCCS}}, \mathsf{compat}, \mu = 1, \nu = 1)$ as follows.

- $\mathsf{compat}(\mathsf{s}_1, \mathsf{s}_2) \to \{0, 1\}$
  - If $\mathsf{s}_1 = \mathsf{s}_2$ return 1, else return 0.
- $\mathcal{G}(1^\lambda) \to \mathsf{pp}$:
  1. Sample size bounds $m, n, N, \ell, t, q, d \in \mathbb{N}$ with $n > \ell$.
  2. Let $s = \lceil \log m \rceil$, $s' = \lceil \log n \rceil$, $s'' = \lceil \log(n - \ell - 1) \rceil$.
  3. $\mathsf{pp}_{\mathsf{PC}} \leftarrow \mathsf{Setup}(1^\lambda, s'')$
  4. Output $\mathsf{pp} \leftarrow (m, n, N, \ell, t, q, d, \mathsf{pp}_{\mathsf{PC}})$
- $\mathcal{K}(\mathsf{pp}, (\mathsf{s}_1, \mathsf{s}_2)) \to (\mathsf{pk}, \mathsf{vk})$:
  1. Let $\mathsf{pk} \leftarrow (\mathsf{pp}, \mathsf{s}_1)$ and $\mathsf{vk} \leftarrow \mathsf{pp}$
  2. Output $(\mathsf{pk}, \mathsf{vk})$
- $\langle \mathcal{P}, \mathcal{V} \rangle((\mathsf{pk}, \mathsf{vk}), (u_1, u_2), (w_1, w_2)) \to (\mathsf{u}, \mathsf{w})$:
  1. Parse u_1 as an $\mathcal{R}_{\mathsf{LCCCS}}$ instance $(C_1, u, \mathsf{x}_1, r_x, [v_i]_{i=1}^t)$. Parse u_2 as $\utilde{\text{an}}$ $\mathcal{R}_{\mathsf{CCCS}}$ instance $\utilde{(C_2, \mathsf{x}_2)}$. Parse w_1 and w_2 as multilinear polynomials $\widetilde{w}_1$ and $\widetilde{w}_2$. Let $\widetilde{z}_1 = (w_1, u, \mathsf{x}_1)$ and $\widetilde{z}_2 = (w_2, 1, \mathsf{x}_2)$.
  2. $\mathcal{V} \to \mathcal{P}$: Sample $\gamma \xleftarrow{R} \mathbb{F}$, $\beta \xleftarrow{R} \mathbb{F}^s$, $r'_x \xleftarrow{R} \mathbb{F}^s$. Send (γ, β) to $\mathcal{P}$.
  3. $\mathcal{V} \leftrightarrow \mathcal{P}$: Run the sum-check protocol
  $$c \leftarrow \langle \mathcal{P}, \mathcal{V}(r'_x) \rangle \left(g, s, d+1, \sum_{j \in [t]} \gamma^j \cdot v_j \right)$$
  where:
  $$g(x) := \left(\sum_{j \in [t]} \gamma^j \cdot L_j(x) \right) + \gamma^{t+1} \cdot Q(x)$$
  $$L_j(x) := \widetilde{\mathsf{eq}}(r_x, x) \cdot \left(\sum_{y \in \{0,1\}^{s'}} \widetilde{M}_j(x, y) \cdot \widetilde{z}_1(y) \right)$$
  $$Q(x) := \widetilde{\mathsf{eq}}(\beta, x) \cdot \left(\sum_{i=1}^{q} c_i \cdot \left(\prod_{j \in S_i} \left(\sum_{y \in \{0,1\}^{s'}} \widetilde{M}_j(x, y) \cdot \widetilde{z}_2(y) \right) \right) \right)$$
  4. $\mathcal{P} \to \mathcal{V}$: Send $[\sigma_i]_{i=1}^t$ and $[\theta_i]_{i=1}^t$ where for all $i \in [t]$:
  $$\sigma_i := \sum_{y \in \{0,1\}^{s'}} \widetilde{M}_i(r'_x, y) \cdot \widetilde{z}_1(y)$$
  $$\theta_i := \sum_{y \in \{0,1\}^{s'}} \widetilde{M}_i(r'_x, y) \cdot \widetilde{z}_2(y)$$

5. $\mathcal{V}$: Compute $e_1 \leftarrow \widetilde{\mathsf{eq}}(r_x, r'_x)$ and $e_2 \leftarrow \widetilde{\mathsf{eq}}(\beta, r'_x)$. Abort if:

$$c \neq \left(\sum_{j \in [t]} \gamma^j \cdot e_i \cdot \sigma_j\right) + \left(\gamma^{t+1} \cdot e_2 \cdot \left(\sum_{i=1}^{q} c_i \cdot \prod_{j \in S_i} \theta_j\right)\right)$$

6. $\mathcal{V} \to \mathcal{P}$: Sample $\rho \xleftarrow{R} \mathbb{F}$ and send to $\mathcal{P}$.
7. $\mathcal{P}, \mathcal{V}$: Output the folded $\mathcal{R}_{\mathsf{LCCCS}}$ instance $\mathsf{u} := (C', u', \mathsf{x}', r'_x, [v'_i]_{i=1}^{t})$ where $\forall i \in [t]$:

$$\begin{aligned} C' &:= C_1 + \rho \cdot C_2, \\ u' &:= u + \rho \cdot 1, \\ \mathsf{x}' &:= \mathsf{x}_1 + \rho \cdot \mathsf{x}_2, \\ v'_i &:= \sigma_i + \rho \cdot \theta_i. \end{aligned}$$

8. $\mathcal{P}$: Output the folded witness $\mathsf{w} := \widetilde{w}'$ where

$$\widetilde{w}' := \widetilde{w}_1 + \rho \cdot \widetilde{w}_2.$$

Theorem 5.1 (A multi-folding scheme for CCS)**.** *Construction 5.1 is a public-coin multi-folding scheme for* $(\mathcal{R}_{\mathsf{LCCCS}}, \mathcal{R}_{\mathsf{CCCS}}, \mathsf{compat}, \mu = 1, \nu = 1)$ *with perfect completeness and knowledge soundness.*

5.2 A Generalized Multi-Folding Scheme for CCS

Construction 5.2 (A Generalized Multi-Folding Scheme for CCS [ZZD23])**.** Let PC be an additively-homomorphic polynomial commitment scheme for multilinear polynomials over a finite field $\mathbb{F}$. We construct a multi-folding scheme for $(\mathcal{R}_{\mathsf{LCCCS}}, \mathcal{R}_{\mathsf{CCCS}}, \mathsf{compat}, \mu, \nu)$ as follows.

- $\mathsf{compat}(\mathsf{s}_1, \mathsf{s}_2) \to \{0, 1\}$
  - If $\mathsf{s}_1 = \mathsf{s}_2$ return 1, else return 0.
- $\mathcal{G}(1^\lambda) \to \mathsf{pp}$:
  1. Sample size bounds $m, n, N, \ell, t, q, d \in \mathbb{N}$ with $n > \ell$.
  2. Sample size bounds $\mu, \nu \in \mathbb{N}$.
  3. Let $s = \lceil \log m \rceil$, $s' = \lceil \log n \rceil$, $s'' = \lceil \log(n - \ell - 1) \rceil$.
  4. $\mathsf{pp}_{\mathsf{PC}} \leftarrow \mathsf{Setup}(1^\lambda, s'')$
  5. Output $\mathsf{pp} \leftarrow (m, n, N, \ell, t, q, d, \mu, \nu, \mathsf{pp}_{\mathsf{PC}})$
- $\mathcal{K}(\mathsf{pp}, (\mathsf{s}_1, \mathsf{s}_2)) \to (\mathsf{pk}, \mathsf{vk})$:
  1. Let $\mathsf{pk} \leftarrow (\mathsf{pp}, \mathsf{s}_1)$ and $\mathsf{vk} \leftarrow \mathsf{pp}$
  2. Output $(\mathsf{pk}, \mathsf{vk})$
- $\langle \mathcal{P}, \mathcal{V} \rangle((\mathsf{pk}, \mathsf{vk}), (\vec{u}_1, \vec{u}_2), (\vec{w}_1, \vec{w}_2)) \to (\mathsf{u}, \mathsf{w})$:
  1. Parse $(\vec{u}_1, \vec{w}_1)$ as $[\phi_k]_{k \in [\mu]}$ and $(\vec{u}_2, \vec{w}_2)$ as $[\psi_k]_{k \in [\nu]}$.
  2. $\mathcal{V} \to \mathcal{P}$: Sample $\gamma \xleftarrow{R} \mathbb{F}$, $\beta \xleftarrow{R} \mathbb{F}^s$, $r'_x \xleftarrow{R} \mathbb{F}^s$, and send (γ, β) to $\mathcal{P}$.

3. $\mathcal{V} \leftrightarrow \mathcal{P}$: Run the sum-check protocol

$$c \leftarrow \langle \mathcal{P}, \mathcal{V}(r'_x) \rangle \left(g, s, d+1, \sum_{k=1}^{\mu} \sum_{i=1}^{t} \gamma^{(k-1)t+i} \phi_k.v_i \right),$$

where

$$g(x) := \sum_{k=1}^{\mu} \sum_{i=1}^{t} \left(\gamma^{(k-1)t+i} \cdot L_{k,i}(x) \right) + \sum_{k'=1}^{\nu} \gamma^{\mu t + k'} \cdot Q_{k'}(x)$$
$$L_{k,i}(x) := \widetilde{\mathsf{eq}}(\phi_k.r_x, x) \cdot \left(\sum_{y \in \{0,1\}^{s'}} \widetilde{M}_i(x,y) \cdot \phi_k.\widetilde{z}_1(y) \right)$$
$$Q_{k'}(x) := \widetilde{\mathsf{eq}}(\beta, x) \cdot \left(\sum_{i=1}^{q} c_i \cdot \left(\prod_{j \in S_i} \left(\sum_{y \in \{0,1\}^{s'}} \widetilde{M}_j(x,y) \cdot \psi_{k'}.\widetilde{z}_2(y) \right) \right) \right)$$

4. $\mathcal{P} \rightarrow \mathcal{V}$: Send $[\sigma_{k,i}, \theta_{k',i}]_{k \in [\mu], k' \in [\nu], i \in [t]}$ where for all $i \in [t]$:

$$\sigma_{k,i} := \sum_{y \in \{0,1\}^{s'}} \widetilde{M}_i(r'_x, y) \cdot \phi_k.\widetilde{z}_1(y)$$
$$\theta_{k',i} := \sum_{y \in \{0,1\}^{s'}} \widetilde{M}_i(r'_x, y) \cdot \psi_{k'}.\widetilde{z}_2(y)$$

5. $\mathcal{V} \rightarrow \mathcal{P}$: Compute $[e_{k,1} := \widetilde{\mathsf{eq}}(\phi_k.r_x, r'_x)]_{k \in [\mu]}$, and $e_2 := \widetilde{\mathsf{eq}}(\beta, r'_x)$, and abort if

$$c \neq \sum_{k=1}^{\mu} \sum_{i=1}^{t} \left(\gamma^{(k-1)t+i} \cdot e_{k,1} \cdot \sigma_{k,i} \right) + \sum_{k'=1}^{\nu} \gamma^{\mu t + k'} \cdot e_2 \cdot \left(\sum_{i=1}^{q} c_i \cdot \prod_{j \in S_i} \theta_{k',j} \right)$$

6. $\mathcal{V} \rightarrow \mathcal{P}$: Sample $\rho \xleftarrow{R} \mathbb{F}$ and send it to $\mathcal{P}$.
7. $\mathcal{P}, \mathcal{V}$: Output the folded $\mathcal{R}_{\mathsf{LCCCS}}$ instance $\mathsf{u} := (C', u', \mathsf{x}', r'_x, [v_i]_{i=1}^{t}])$, where $\forall i \in [t]$:

$$C' := \sum_{k=1}^{\mu} \rho^{k-1} \cdot \phi_k.C_1 + \sum_{k'=1}^{\nu} \rho^{\mu-1+k'} \cdot \psi_{k'}.C_2,$$
$$u' := \sum_{k=1}^{\mu} \rho^{k-1} \cdot \phi_k.u + \sum_{k'=1}^{\nu} \rho^{\mu-1+k'} \cdot 1,$$
$$\mathsf{x}' := \sum_{k=1}^{\mu} \rho^{k-1} \cdot \phi_k.\mathsf{x}_1 + \sum_{k'=1}^{\nu} \rho^{\mu-1+k'} \cdot \psi_{k'}.\mathsf{x}_2,$$
$$v'_i := \sum_{k=1}^{\mu} \rho^{k-1} \cdot \sigma_{k,i} + \sum_{k'=1}^{\nu} \rho^{\mu-1+k'} \cdot \theta_{k',i}.$$

8. $\mathcal{P}$: Output the folded $\mathcal{R}_{\mathsf{LCCCS}}$ witness $\mathsf{w} := \widetilde{w}'$, where

$$\widetilde{w}' := \sum_{k=1}^{\mu} \rho^{k-1} \cdot \phi_k.\widetilde{w}_1 + \sum_{k'=1}^{\nu} \rho^{\mu-1+k'} \cdot \psi_{k'}.\widetilde{w}_2.$$

Theorem 5.2. *Construction 5.2 is a public-coin multi-folding scheme for $(\mathcal{R}_{\mathsf{LCCCS}}, \mathcal{R}_{\mathsf{CCCS}}, \mu, \nu)$ with perfect completeness and knowledge soundness.*

5.3 The Fiat-Shamir Transform For Multi-Folding Schemes

We construct a generic compiler FS that endows non-interactivity to multi-folding schemes via the Fiat-Shamir transform. We adapt the system from [FS86; KST22; KS23b].

Construction 5.3 (Fiat-Shamir transform for multi-folding schemes)**.** Let $\Pi_{\mathsf{MFS}} = (\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ be a public-coin multi-folding scheme for $(\mathcal{R}_1, \mathcal{R}_2, \mathsf{compat}, \mu = 1, \nu = 1)$ with ℓ rounds. Let ρ denote a random oracle. We construct a compiler FS which transforms Π_{MFS} into a non-interactive multi-folding scheme $\Pi_{\mathsf{NIMFS}} = (\mathcal{G}', \mathcal{K}', \mathcal{P}', \mathcal{V}')$ for $(\mathcal{R}_1, \mathcal{R}_2, \mathsf{compat}, \mu = 1, \nu = 1)$ in the random oracle model. We denote this transformation as

$$\Pi_{\mathsf{NIMFS}} = \mathsf{FS}[\Pi_{\mathsf{MFS}}]$$

The compiler operates as follows.

- $\mathcal{G}'(1^\lambda) \to \mathsf{pp}$: Output $\mathsf{pp} \leftarrow \mathcal{G}(1^\lambda)$.
- $\mathcal{K}'(\mathsf{pp}, (\mathsf{s}_1, \mathsf{s}_2)) \to (\mathsf{pk}, \mathsf{vk})$:
  - $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, (\mathsf{s}_1, \mathsf{s}_2))$.
  - $r_1 \leftarrow \rho(\mathsf{pp}, (\mathsf{s}_1, \mathsf{s}_2))$.
  - Output $(\mathsf{pk}', \mathsf{vk}') \leftarrow ((\mathsf{pk}, r_1), (\mathsf{vk}, r_1))$.
- $\mathcal{P}'(\mathsf{pk}', (u_1, w_2), (u_1, w_2)) \to (u, w, \pi)$:
  - Parse pk' as (pk, r_1).
  - Compute $(u, w) \leftarrow \mathcal{P}(\mathsf{pk}, (u_1, w_1), (u_2, w_2))$. On the ith message m_i, respond with verifier randomness $r_{i+1} \leftarrow \rho(m_i, r_i)$ for $i \in [\ell]$. Let $\pi = (m_1, \ldots, m_\ell)$ be the messages sent by $\mathcal{P}$.
  - Output (u, w, π).
- $\mathcal{V}'(\mathsf{vk}', (u_1, u_2), \pi) \to u$:
  - Parse vk' as (vk, r_1) and π as $(m_1, \ldots, m_\ell)$.
  - Compute $u \leftarrow \mathcal{V}(\mathsf{vk}, (u_1, u_2))$ with randomness $(r_1, \ldots, r_{\ell+1})$ computed as $r_{i+1} \leftarrow \rho(m_i, r_i)$. In round i, send the prover message m_i.
  - Output u.

Lemma 5.3 (Fiat-Shamir transform for multi-folding schemes)**.** *Construction 5.3 transforms a public-coin multi-folding scheme Π_{MFS} for $(\mathcal{R}_1, \mathcal{R}_2, \mathsf{compat}, \mu = 1, \nu = 1)$ into a non-interactive multi-folding scheme Π_{NIMFS} for $(\mathcal{R}_1, \mathcal{R}_2, \mathsf{compat}, \mu = 1, \nu = 1)$ in the random oracle model.*

5.4 The CycleFold Compiler

Let (E_1, E_2) denote a 2-cycle of elliptic curves. Let $\mathbb{F}_1$ and $\mathbb{F}_2$ denote the scalar fields of E_1 and E_2, respectively. Naturally, let $\mathbb{F}_2$ be the base field of E_1 and $\mathbb{F}_1$ be the base field of E_2. We adapt the system from [KS23a].

Construction 5.4 (CycleFold Transform)**.** Let $\Pi^{(E)}_{\mathsf{MFS}} = (\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ be a multi-folding scheme for $(\mathcal{R}_1, \mathcal{R}_2, \mathsf{compat})$ defined over a single elliptic curve. We construct a compiler CF, invoked as

$$\Pi^{(E_1,E_2)}_{\mathsf{MFS}} = \mathsf{CF}[\Pi^{(E)}_{\mathsf{MFS}}]$$

which transforms a multi-folding scheme $\Pi^{(E)}_{\mathsf{MFS}}$ described over a single curve into a multi-folding scheme $\Pi^{(E_1,E_2)}_{\mathsf{MFS}} = (\mathcal{G}', \mathcal{K}', \mathcal{P}', \mathcal{V}')$ for $(\mathcal{R}_1 \times \mathcal{R}_{\mathsf{CRR1CS}}, \mathcal{R}_2, \mathsf{compat}')$ that uses a 2-cycle of curves to delegate foreign arithmetic operations to a minimal $\mathbb{F}_2$-circuit. The compiler operates as follows.

First, we define a function F_{EC} that performs elliptic curve arithmethic on $\mathbb{F}_2$, as follows.

- $F_{\mathsf{EC}}(\rho, C_1, C_2, C')$:
    - Check that $C' = C_1 + \rho \cdot C_2$.

This function encodes a single scalar multiplication and addition over the curve E_2, and is thus efficiently representable as an R1CS circuit over $\mathbb{F}_2$.

- $\mathsf{compat}'(\mathsf{s}_1, \mathsf{s}_2) \to \{0, 1\}$:
    1. Parse s_1 as $(\mathsf{s}_{\mathcal{R}_1}, \mathsf{s}_{\mathsf{CRR1CS}})$.
    2. Parse s_2 as $\mathsf{s}_{\mathcal{R}_2}$.
    3. Check that $\mathsf{s}_{\mathsf{CRR1CS}} = \mathsf{s}_{\mathsf{EC}}$.
    4. Output $\mathsf{compat}(\mathsf{s}_{\mathcal{R}_1}, \mathsf{s}_{\mathcal{R}_2})$.
- $\mathcal{G}'(1^\lambda) \to \mathsf{pp}'$:
    - $\mathsf{pp} \leftarrow \mathsf{G}(1^\lambda)$.
    - Compute R1CS parameters for s_{EC}:
        * Compute size bounds $m, n, N, \ell \in \mathbb{N}$ where $n > \ell$
        * $\mathsf{pp}_E \leftarrow \mathsf{VC.Gen}(1^\lambda, m)$
        * $\mathsf{pp}_W \leftarrow \mathsf{VC.Gen}(1^\lambda, n - \ell - 1)$
        * Set $\mathsf{pp}_{\mathsf{EC}} \leftarrow (m, n, N, \ell, \mathsf{pp}_E, \mathsf{pp}_W)$
    - Output $\mathsf{pp}' \leftarrow (\mathsf{pp}, \mathsf{pp}_{\mathsf{EC}})$
- $\mathcal{K}'(\mathsf{pp}', (\mathsf{s}_1, \mathsf{s}_2)) \to (\mathsf{pk}', \mathsf{vk}')$:
    - Parse pp' as $(\mathsf{pp}, \mathsf{pp}_{\mathsf{EC}})$.
    - Parse s_1 as $(\mathsf{s}_{\mathcal{R}_1}, \mathsf{s}_{\mathsf{EC}})$.
    - Parse s_2 as $\mathsf{s}_{\mathcal{R}_2}$.
    - $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, (\mathsf{s}_{\mathcal{R}_1}, \mathsf{s}_{\mathcal{R}_2}))$
    - Set $\mathsf{pk}' \leftarrow (\mathsf{pk}, \mathsf{s}_{\mathsf{EC}})$, $\mathsf{vk}' \leftarrow \mathsf{vk}$.
    - Output $(\mathsf{pk}, \mathsf{vk})$.
- $\langle \mathcal{P}, \mathcal{V} \rangle'((\mathsf{pk}', \mathsf{vk}'), (u_1', w_1'), (u_2', w_2')) \to (u', w')$:
    - Parse pk' as $(\mathsf{pk}, \mathsf{s}_{\mathsf{EC}})$ and vk' as vk.
    - Parse u_1' as $(u_1, u_{\mathsf{EC},1})$ and u_2' as u_2.
    - Parse w_1' as $(w_1, w_{\mathsf{EC},1})$ and w_2' as w_2.
    - Run $(u, w) \leftarrow \langle \mathcal{P}, \mathcal{V} \rangle((\mathsf{pk}, \mathsf{vk}), (u_1, w_1), (u_2, w_2))$ except that:
        * Parse $u_{\mathsf{EC},1}$ as $(\overline{E}_1, u_1, \overline{W}_1, x_1)$.
        * Initialize the folded CRR1CS instance $u^*_{\mathsf{EC}} = (\overline{E}^*, u^*, \overline{W}^*, x^*)$ as $\overline{E}^* \leftarrow \overline{E}_1$, $u^* \leftarrow u_1$, $\overline{W}^* \leftarrow \overline{W}_1$, $x^* \leftarrow x_1$.
        * For each $i \in [k]$ of the k foreign-field computations that $\mathcal{V}$ performs over $\mathbb{F}_2$ of the form

$$C'^{(i)} \leftarrow C_1^{(i)} + \rho \cdot C_2^{(i)}$$

do instead:

· $\mathcal{P} \to \mathcal{V}$: Compute

$$(u^{(i)}_{\mathsf{EC},2}, w^{(i)}_{\mathsf{EC},2}) \leftarrow \mathsf{trace}(F_{\mathsf{EC}}, (\rho, C^{(i)}_1, C^{(i)}_2, C'^{(i)}))$$

where $u^{(i)}_{\mathsf{EC},2} = (\overline{E}^{(i)}_2, u^{(i)}_2, \overline{W}^{(i)}_2, x^{(i)}_2)$ and $w^{(i)}_{\mathsf{EC},2} = (E^{(i)}_2, W^{(i)}_2)$ are an instance-witness tuple with structure s_{EC}, such that $u^{(i)}_2 = 1$, $\overline{E}^{(i)}_2 = \overline{0}$ and $x^{(i)}_2 = (\rho, C^{(i)}_1, C^{(i)}_2, C'^{(i)})$ for some $C'^{(i)} \in \mathbb{G}_2$.

· $\mathcal{V}$: Abort if $u^{(i)}_2 \neq 1$ or $\overline{E}_2 \neq \overline{0}$ or $x^{(i)}_2 \neq (\rho, C^{(i)}_1, C^{(i)}_2, C'^{(i)})$ for some $C'^{(i)} \in \mathbb{G}_2$.

· $\mathcal{P} \to \mathcal{V}$: Send $\overline{T}^{(i)} = \mathsf{VC.Com}(\mathsf{pp}_E, T^{(i)})$ where

$$T^{(i)} = AZ_1 \circ BZ_2 + AZ_2 \circ BZ_1 - u^* \cdot CZ_2 - u^{(i)}_2 \cdot CZ_1$$

where $Z_1 = (W^*, x^*, u^*)$ and $Z_2 = (W^{(i)}_2, x^{(i)}_2, u^{(i)}_2)$.

· $\mathcal{V} \to \mathcal{P}$: $\mathcal{V}$ samples $\rho^{*,(i)} \xleftarrow{R} \mathbb{F}_1$ and sends it to $\mathcal{P}$.

· $\mathcal{P}, \mathcal{V}$: Update the CRR1CS instance $u^*_{\mathsf{EC}} = (\overline{E}^*, u^*, \overline{W}^*, x^*)$:

$$\begin{aligned}
\overline{E}^* &\leftarrow \overline{E}_1 + \rho^{*,(i)} \cdot \overline{T}^{(i)}\\
u^* &\leftarrow u_1 \ + \rho^{*,(i)} \cdot 1\\
\overline{W}^* &\leftarrow \overline{W}_1 + \rho^{*,(i)} \cdot \overline{W}^{(i)}_2\\
x^* &\leftarrow x_1 \ + \rho^{*,(i)} \cdot x^{(i)}_2
\end{aligned}$$

* Compute the folded $\mathcal{R}_1$ instance u via the $[C'^{(i)}]^k_{i=1}$ computed by $\mathcal{P}$, and set the folded CRR1CS $u_{\mathsf{EC}} \leftarrow u^*_{\mathsf{EC}}$.
* Output the folded final instance $u' \leftarrow (u, u_{\mathsf{EC}})$.

– $\mathcal{P}$: Invoke P to compute the folded witness w, and compute the folded CRR1CS witness $w_{\mathsf{EC}} \leftarrow (E^*, W^*)$, where:

$$\begin{aligned}
E^* &\leftarrow E_1 + \rho^{*,(i)} \cdot T^{(i)}\\
W^* &\leftarrow W_1 + \rho^{*,(i)} \cdot W^{(i)}_2
\end{aligned}$$

are computed for $i \in [k]$.

– Output the folded witness $w' \leftarrow (w, w_{\mathsf{EC}})$.

Theorem 5.4 (CycleFold transform for folding schemes)**.** *Construction 5.4 transforms a public-coin multi-folding scheme $\Pi^{(E)}_{\mathsf{MFS}}$ for $(\mathcal{R}_1, \mathcal{R}_2, \mathsf{compat}, \mu = 1, \nu = 1)$ defined over a single elliptic curve into a public-coin multi-folding scheme $\Pi^{(E_1,E_2)}_{\mathsf{MFS}}$ over $(\mathcal{R}_1 \times \mathcal{R}_{\mathsf{CRR1CS}}, \mathcal{R}_2, \mathsf{compat}', \mu = 1, \nu = 1)$ defined over a 2-cycle of elliptic curves.*

5.5 HyperNova

Construction 5.5 (HyperNova [KS23b])**.** Let $\mathsf{NIMFS} = \mathsf{FS}[\Pi_{\mathsf{MFS}}]$ be the non-interactive multi-folding scheme for $(\mathcal{R}_{\mathsf{LCCCS}}, \mathcal{R}_{\mathsf{CCCS}}, \mathsf{compat}, \mu = 1, \nu = 1)$ obtained from applying the Fiat-Shamir transform to Construction 5.1. We construct an IVC scheme for CCCS as follows.

Let F be a polynomial-time function, and let hash be a cryptographic hash function. We define an augmented function F' as follows, where all arguments are taken as non-deterministic advice.

• $F'(\mathsf{vk}, \mathsf{U}_i, \mathsf{u}_i, (i, z_0, z_i), \omega_i, \pi) \to \mathsf{x}$:

1. If $i = 0$, output $\mathsf{hash}(\mathsf{vk}, 1, z_0, F(z_0, \omega_0), \mathsf{u}_\perp)$
2. Else:
   (a) Check $\mathsf{u}_i.\mathsf{x} = \mathsf{hash}(\mathsf{vk}, i, z_0, z_i, \mathsf{U}_i)$
   (b) Fold $\mathsf{U}_{i+1} \leftarrow \mathsf{NIFMS.V}(\mathsf{vk}, \mathsf{U}_i, \mathsf{u}_i, \pi)$
   (c) Output $\mathsf{hash}(\mathsf{vk}, i+1, z_0, F(z_i, \omega_i), \mathsf{U}_{i+1})$

Because F' can be computed in polynomial time, it can be represented as an $\mathcal{R}_{\mathsf{CCCS}}$ structure $\mathsf{s}_{F'}$. Let

$$(\mathsf{u}_{i+1}, \mathsf{w}_{i+1}) \leftarrow \mathsf{trace}(F', (\mathsf{vk}, \mathsf{U}_i, \mathsf{u}_i, (i, z_0, z_i), \omega_i, \pi_i))$$

denote the satisfying $\mathcal{R}_{\mathsf{CCCS}}$ instance-witness pair $(\mathsf{u}_{i+1}, \mathsf{w}_{i+1})$ for the execution of F' on non-deterministic advice $(\mathsf{vk}, \mathsf{U}_i, \mathsf{u}_i, (i, z_0, z_i), \omega_i, \pi)$.

We define the IVC scheme $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ as follows.

- $\mathcal{G}(1^\lambda) \to \mathsf{pp}$: Output $\mathsf{pp} \leftarrow \mathsf{NIMFS.G}(1^\lambda)$.
- $\mathcal{K}(\mathsf{pp}, F) \to (\mathsf{pk}, \mathsf{vk})$:
  1. $(\mathsf{pk}_{\mathsf{fs}}, \mathsf{vk}_{\mathsf{fs}}) \leftarrow \mathsf{NIMFS.K}(\mathsf{pp}, (s_{F'}, s_{F'}))$
  2. Output $(\mathsf{pk}, \mathsf{vk}) \leftarrow ((F, \mathsf{pk}_{\mathsf{fs}}), (F, \mathsf{vk}_{\mathsf{fs}}))$
- $\mathcal{P}(\mathsf{pk}, (i, z_0, z_i), \omega_i, \Pi_i) \to \Pi_{i+1}$:
  1. Parse Π_i as $((\mathsf{U}_i, \mathsf{W}_i), (\mathsf{u}_i, \mathsf{w}_i))$
  2. Fold $(\mathsf{U}_{i+1}, \mathsf{W}_{i+1}, \pi) \leftarrow$ If $i = 0$ $\{(\mathsf{u}_\perp, \mathsf{w}_\perp, \perp\}$ else $\{\mathsf{NIMFS.P}(\mathsf{pk}, (\mathsf{U}_i, \mathsf{W}_i), (\mathsf{u}_i, \mathsf{w}_i))\}$
  3. Compute $(\mathsf{u}_{i+1}, \mathsf{w}_{i+1}) \leftarrow \mathsf{trace}(F', (\mathsf{vk}, \mathsf{U}_i, \mathsf{u}_i, (i, z_0, z_i), \omega_i, \pi))$
  4. Output $\Pi_{i+1} \leftarrow ((\mathsf{U}_{i+1}, \mathsf{W}_{i+1}), (\mathsf{u}_{i+1}, \mathsf{w}_{i+1}))$
- $\mathcal{V}(\mathsf{vk}, (i, z_0, z_i), \Pi_i) \to \{0, 1\}$:
  1. If $i = 0$, check $z_i = z_0$
  2. Else:
     (a) Parse Π_i as $((\mathsf{U}_i, \mathsf{W}_i), (\mathsf{u}_i, \mathsf{w}_i))$
     (b) Check $\mathsf{u}_i.\mathsf{x} = \mathsf{hash}(\mathsf{vk}, i, z_0, z_i, \mathsf{U}_i)$
     (c) Check that $(\mathsf{U}_i, \mathsf{W}_i)$ and $(\mathsf{u}_i, \mathsf{w}_i)$ are satisfying $\mathcal{R}_{\mathsf{LCCCS}}$ and $\mathcal{R}_{\mathsf{CCCS}}$ instance-witness pairs, respectively.

Theorem 5.5 (HyperNova IVC)**.** *Construction 5.5 is an IVC scheme with perfect completeness, knowledge soundness and succinctness.*

5.6 Parallel Nova

We construct a parallel Nova scheme for binary-tree PCD transcripts, which we later combine with [KS23a] two instantiate the scheme on a 2-cycle of elliptic curves, see Sec. 7.

Construction 5.6 (Parallel Nova)**.** Let $\mathsf{NIFS} = (\mathsf{G}, \mathsf{K}, \mathsf{P}, \mathsf{V})$ be the non-interactive folding scheme for $\mathcal{R}_{\mathsf{CRR1CS}}$. We construct a PCD scheme from Nova for binary-tree transcripts.

A message z is a tuple (n, z_ℓ, z_r) attesting to $F^{(n)}(z_\ell) = z_r$. We define the trivial message as $\perp = (0, z_\ell, z_r)$ for any $z_\ell = z_r$. We define T to be a binary tree, where each node $v = (n, z_\ell, z_r) \in V(\mathsf{T})$ is labeled by its outgoing message, and for every pair of nodes (u, v), there exists an edge $e = (u, v) \in E(\mathsf{T})$ between them iff $n^{(v)} = 2n^{(u)} + 1$ and either $z_\ell^{(v)} = z_\ell^{(u)}$ or $z_r^{(v)} = z_r^{(u)}$.

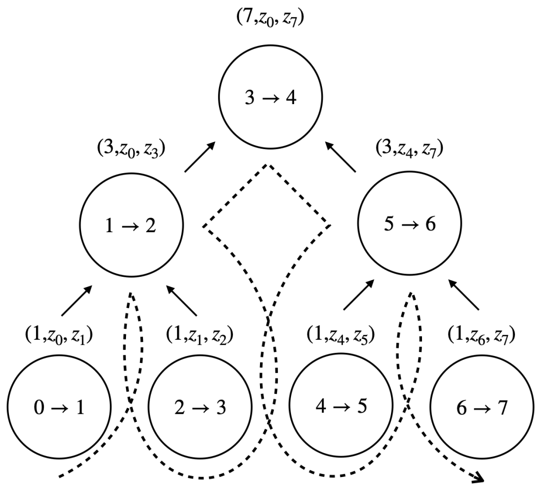


Figure 7: Post-order depth-first traversal of the binary-tree parallel Nova construction.

The nodes jointly compute F trough a post-order depth-first traversal of the tree, starting from the left-most leaf, passing through the root, and ending at the right-most leaf. The root accumulates the proofs of all its downstream descendants, therefore attesting to the whole computation.

Let F be a polynomial time function. First, we define a compliance predicate φ for F and an augmented compliance predicate φ' as follows:

- $\varphi(z, \omega, [z_\ell, z_r]) \to \{0, 1\}$:
    1. Parse z as (n, z_ℓ, z_r), z_ℓ as $(n_\ell, z_\ell^{(\ell)}, z_r^{(\ell)})$ and z_r as $(n_r, z_\ell^{(r)}, z_r^{(r)})$
    2. If $z_\ell = z_r = \perp$, check $z_\ell^{(\ell)} = z_r^{(\ell)}$ and $z_\ell^{(r)} = z_r^{(r)}$
    3. Check $z_r^{(\ell)} = F(z_\ell^{(r)}, \omega)$
    4. Check $n = n_\ell + 1 + n_r$
    5. Check $z_\ell = z_\ell^{(\ell)}$ and $z_r = z_r^{(r)}$
- $\varphi'(h, (z, \omega, [(z_\ell, \mathsf{U}_\ell, \mathsf{u}_\ell, \pi_\ell), (z_r, \mathsf{U}_r, \mathsf{u}_r, \pi_r)], \mathsf{vk}, \mathsf{U}, \pi)) \to \{0, 1\}$:
    1. Check $\varphi(z, \omega, [z_\ell, z_r]) = 1$.
    2. If $z_\ell = z_r = \perp$, then check $h = \mathsf{hash}(\mathsf{vk}, z, \perp)$.
       Else,
        (a) Check $\mathsf{u}_\ell.\mathsf{x} = \mathsf{hash}(\mathsf{vk}, z_\ell, \mathsf{U}_\ell)$
        (b) Check $\mathsf{u}_r.\mathsf{x} = \mathsf{hash}(\mathsf{vk}, z_r, \mathsf{U}_r)$
        (c) Fold $\mathsf{U}'_\ell \leftarrow \mathsf{NIFS.V}(\mathsf{vk}, \mathsf{U}_\ell, \mathsf{u}_\ell, \pi_\ell)$
        (d) Fold $\mathsf{U}'_r \leftarrow \mathsf{NIFS.V}(\mathsf{vk}, \mathsf{U}_r, \mathsf{u}_r, \pi_r)$
        (e) Fold $\mathsf{U} \leftarrow \mathsf{NIFS.V}(\mathsf{vk}, \mathsf{U}'_\ell, \mathsf{U}'_r, \pi)$
        (f) Check $h = \mathsf{hash}(\mathsf{vk}, z, \mathsf{U})$

Since φ' can be computed in polynomial time, it can be represented as an $\mathcal{R}_{\mathsf{CRR1CS}}$ structure. Let,

$$(\mathsf{u}, \mathsf{w}) \leftarrow \mathsf{trace}(R_\varphi, (h, (z, \omega, [(z_\ell, \mathsf{U}_\ell, \mathsf{u}_\ell, \pi_\ell), (z_r, \mathsf{U}_r, \mathsf{u}_r, \pi_r)], \mathsf{vk}, \mathsf{U}, \pi))$$

denote the satisfying $\mathcal{R}_{\mathsf{CRR1CS}}$ instance-witness pair for the execution of R_φ on the above input.

We define the PCD scheme $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ as follows.

- $\mathcal{G}(1^\lambda) \to \mathsf{pp}$: Output $\mathsf{pp} \leftarrow \mathsf{NIFS.G}(1^\lambda)$.
- $\mathcal{K}(\mathsf{pp}, \varphi) \to (\mathsf{pk}, \mathsf{vk})$:
  1. Compute $(\mathsf{pk_{fs}}, \mathsf{vk_{fs}}) \leftarrow \mathsf{NIFS.K}(\mathsf{pp}, \varphi')$
  2. Output $(\mathsf{pk}, \mathsf{vk}) \leftarrow ((\mathsf{pk_{fs}}, \mathsf{vk_{fs}}), \mathsf{vk_{fs}})$
- $\mathcal{P}(\mathsf{pk}, z, \omega, [(z_\ell, \Pi_\ell), (z_r, \Pi_r)]) \to \Pi$:
  1. Parse Π_ℓ as $((\mathsf{U}_\ell, \mathsf{W}_\ell), (\mathsf{u}_\ell, \mathsf{w}_\ell))$
  2. Parse Π_r as $((\mathsf{U}_r, \mathsf{W}_r), (\mathsf{u}_r, \mathsf{w}_r))$
  3. If $z_\ell = z_r = \perp$, then set $(\mathsf{U}, \mathsf{W}, \pi) \leftarrow (\mathsf{u}_\perp, \mathsf{w}_\perp, \perp)$ and $(\pi_\ell, \pi_r) \leftarrow (\perp, \perp)$.
     Else,
     (a) Fold $(\mathsf{U}'_\ell, \mathsf{W}'_\ell, \pi_\ell) \leftarrow \mathsf{NIFS.P}(\mathsf{pk}, (\mathsf{U}_\ell, \mathsf{W}_\ell), (\mathsf{u}_\ell, \mathsf{w}_\ell))$
     (b) Fold $(\mathsf{U}'_r, \mathsf{W}'_r, \pi_r) \leftarrow \mathsf{NIFS.P}(\mathsf{pk}, (\mathsf{U}_r, \mathsf{W}_r), (\mathsf{u}_r, \mathsf{w}_r))$
     (c) Fold $(\mathsf{U}, \mathsf{W}, \pi) \leftarrow \mathsf{NIFS.P}(\mathsf{pk}, (\mathsf{U}'_\ell, \mathsf{W}'_\ell), (\mathsf{U}'_r, \mathsf{W}'_r))$
  4. Compute $h \leftarrow \mathsf{hash}(\mathsf{vk}, z, \mathsf{U})$.
  5. $(\mathsf{u}, \mathsf{w}) \leftarrow \mathsf{trace}(\varphi', (h, (z, \omega, [(z_\ell, \mathsf{U}_\ell, \mathsf{u}_\ell, \pi_\ell), (z_r, \mathsf{U}_r, \mathsf{u}_r, \pi_r)], \mathsf{vk}, \mathsf{U}, \pi))$
  6. Output $\Pi \leftarrow ((\mathsf{U}, \mathsf{w}), (\mathsf{u}, \mathsf{w}))$
- $\mathcal{V}(\mathsf{vk}, z, \Pi) \to \{0, 1\}$:
  1. Parse Π as $((\mathsf{U}, \mathsf{w}), (\mathsf{u}, \mathsf{w}))$.
  2. Check that $\mathsf{u.x} = \mathsf{hash}(\mathsf{vk}, z, \mathsf{U})$.
  3. Check that (U, W) and (u, w) are satisfying instance-witness pairs with respect to the structure corresponding to φ'.

Theorem 5.6. *Construction 5.6 is a PCD scheme with perfect completeness, knowledge soundness and succinctness.*

5.7 Parallel HyperNova

Construction 5.7 (Parallel HyperNova)**.** Let $\mathsf{NIMFS} = (\mathsf{G}, \mathsf{K}, \mathsf{P}, \mathsf{V})$ be the non-interactive multi-folding scheme for $(\mathcal{R}_{\mathsf{LCCCS}}, \mathcal{R}_{\mathsf{CCCS}}, \mathsf{compat}, \mu, \nu)$ from [ZZD23]. We construct a PCD scheme from HyperNova for r-ary tree transcripts.

A message z is a tuple $(n, z^{(\ell)}, z^{(r)})$ attesting to $F^{(n)}(z^{(\ell)}) = z^{(r)}$. We define the trivial message as $\bot = (0, \bot, \bot)$. We define T to be an r-ary tree, where each node $v = (n, z^{(\ell)}, z^{(r)}) \in V(\mathsf{T})$ is labeled by its outgoing message.

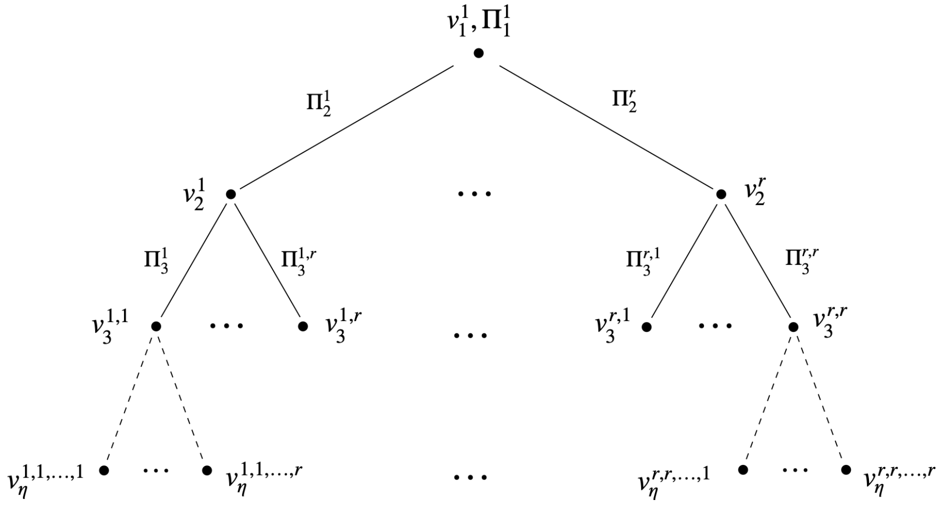


Figure 8: An r-ary tree transcript of a PCD scheme for a parallel HyperNova construction.

Let F be a polynomial time function. First, we define a compliance predicate φ for F and an augmented compliance predicate R_φ as follows:

- $\varphi(z, [\omega_i]_{i=1}^r, [z_i]_{i=1}^r) \to \{0, 1\}$:
  1. Parse z as $(n, z^{(\ell)}, z^{(r)})$.
  2. For $i \in [r]$, parse z_i as $(n_i, z_i^{(\ell)}, z_i^{(r)})$.
  3. For $i \in \{1, \ldots, r-1\}$, check $z_{i+1}^{(\ell)} = F(z_i^{(r)}, \omega_i)$.
  4. Check $n = r - 1 + \sum_{i \in [r]} n_i$.
  5. Check $z^{(\ell)} = z_1^{(\ell)}$ and $z^{(r)} = z_r^{(r)}$.
- $R_\varphi(h, (z, [\omega_i]_{i=1}^r, [(z_i, \mathsf{U}_i, \mathsf{u}_i)]_{i=1}^r, \mathsf{vk}, \mathsf{U}, \pi)) \to \{0, 1\}$:
  1. Check $\varphi(z, [\omega_i]_{i=1}^r, [z_i]_{i=1}^r) = 1$.
  2. If $z_i = \bot$ for all $i \in [r]$, then check $h = \mathsf{hash}(\mathsf{vk}, z, \bot)$.
     Else,
     (a) For $i \in [r]$, check $\mathsf{u}_i.\mathsf{x} = \mathsf{hash}(\mathsf{vk}, z_i, \mathsf{U}_i)$
     (b) Fold $\mathsf{U} \leftarrow \mathsf{NIMFS.V}(\mathsf{vk}, [\mathsf{U}_i]_{i=1}^r, [\mathsf{u}_i]_{i=1}^r, \pi)$
     (c) Check $h = \mathsf{hash}(\mathsf{vk}, z, \mathsf{U})$

Since R_φ can be computed in polynomial time, it can be represented as an $\mathcal{R}_{\mathsf{CCCS}}$ structure. Let,

$$(\mathsf{u}, \mathsf{w}) \leftarrow \mathsf{trace}(R_\varphi, (h, (z, [\omega_i]_{i=1}^r, [(z_i, \mathsf{U}_i, \mathsf{u}_i)]_{i=1}^r, \mathsf{vk}, \mathsf{U}, \pi))$$

denote the satisfying $\mathcal{R}_{\mathsf{CCCS}}$ instance-witness pair for the execution of R_φ on the above input.

We define the PCD scheme $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ as follows.

- $\mathcal{G}(1^\lambda) \to \mathsf{pp}$: Output $\mathsf{pp} \leftarrow \mathsf{NIMFS.G}(1^\lambda)$.
- $\mathcal{K}(\mathsf{pp}, \varphi) \to (\mathsf{pk}, \mathsf{vk})$:
    1. Compute $(\mathsf{pk_{fs}}, \mathsf{vk_{fs}}) \leftarrow \mathsf{NIMFS.K}(\mathsf{pp}, R_\varphi)$
    2. Output $(\mathsf{pk}, \mathsf{vk}) \leftarrow ((\mathsf{pk_{fs}}, \mathsf{vk_{fs}}), \mathsf{vk_{fs}})$
- $\mathcal{P}(\mathsf{pk}, z, [\omega_i]_{i=1}^r, [(z_i, \Pi_i)]_{i=1}^r) \to \Pi$:
    1. For $i \in [r]$, parse Π_i as $((\mathsf{U}_i, \mathsf{W}_i), (\mathsf{u}_i, \mathsf{w}_i))$
    2. If $z_i = \perp$ for all $i \in [r]$, then set $(\mathsf{U}, \mathsf{W}, \pi) \leftarrow (\mathsf{u}_\perp, \mathsf{w}_\perp, \perp)$.
       Else, fold $(\mathsf{U}, \mathsf{W}, \pi) \leftarrow \mathsf{NIMFS.P}(\mathsf{pk}, [(\mathsf{U}_i, \mathsf{W}_i)]_{i=1}^r, [(\mathsf{u}_i, \mathsf{w}_i)]_{i=1}^r)$
    3. Compute $h \leftarrow \mathsf{hash}(\mathsf{vk}, z, \mathsf{U})$.
    4. $(\mathsf{u}, \mathsf{w}) \leftarrow \mathsf{trace}(R_\varphi, (h, (z, [\omega_i]_{i=1}^r, [(z_i, \mathsf{U}_i, \mathsf{u}_i)]_{i=1}^r, \mathsf{vk}, \mathsf{U}, \pi))$
    5. Output $\Pi \leftarrow ((\mathsf{U}, \mathsf{w}), (\mathsf{u}, \mathsf{w}))$
- $\mathcal{V}(\mathsf{vk}, z, \Pi) \to \{0, 1\}$:
    1. Parse Π as $((\mathsf{U}, \mathsf{w}), (\mathsf{u}, \mathsf{w}))$.
    2. Check that $\mathsf{u.x} = \mathsf{hash}(\mathsf{vk}, z, \mathsf{U})$.
    3. Check that (U, W) and (u, w) are satisfying instance-witness pairs with respect to the structure corresponding to R_φ.

Theorem 5.7. *Construction 5.7 is a PCD scheme with perfect completeness, knowledge soundness and succinctness.*

6 Defining Zero-Knowledge Virtual Machines

Definition 6.1 (zkVM Scheme)**.** Let VC be a binding vector commitment scheme with succinct commitments. We define a *zero-knowledge verifiable computing machine* (zkVM) scheme as a four-tuple of PPT algorithms $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$, denoting the generator, encoder, prover and verifier, with the following interface.

- $\mathcal{G}(1^\lambda) \to \mathsf{pp}$: Given a security parameter λ, samples public parameters pp.
- $\mathcal{K}(\mathsf{pp}, \Xi) \to (\mathsf{pk}, \mathsf{vk})$: Given public parameters pp, and the encoding of a machine architecture $\Xi = (\varphi, (F_1, \dots, F_\ell))$, outputs a prover key pk and verifier key vk.
- $\mathcal{P}(\mathsf{pk}, F^\Xi, x, \omega) \to (y, \pi)$: Given a prover key pk, a program F^Ξ encoded for Ξ, a public input x, and a private input ω, outputs a claimed output y and a proof π, attesting to $y = F^\Xi(x, \omega)$.
- $\mathcal{V}(\mathsf{vk}, \overline{F^\Xi}, x, y, \pi) \to \{0, 1\}$: Given a verifier key vk, a commitment to a program $\overline{F^\Xi}$ encoded for Ξ, a public input x, a claimed output y and a proof π, outputs 1 (`accept`) or 0 (`reject`).

A zkVM scheme $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ must satisfy the following requirements. In what follows, we implicitly consider $\overline{F^\Xi}$ to be computed via the commit operation of the vector commitment scheme on F^Ξ.

- **Perfect Completeness**. For all PPT adversaries $\mathcal{A}$:

$$\Pr\left[\mathcal{V}(\mathsf{vk}, \overline{F^\Xi}, x, y, \pi) = 1 \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda) \\ (\Xi, F^\Xi, x, \omega) \leftarrow \mathcal{A}(\mathsf{pp}) \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \Xi) \\ (y, \pi) \leftarrow \mathcal{P}(\mathsf{pk}, F^\Xi, x, \omega) \end{array}\right] = 1$$

- **Knowledge Soundness**. For all PPT adversaries $\mathcal{P}^*$, there exists a PPT extractor $\mathcal{E}$ such that for all randomness ρ

$$\Pr\left[\begin{array}{l} y' \neq y, \\ \mathcal{V}(\mathsf{vk}, \overline{F^\Xi}, x, y, \pi) = 1 \end{array} \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda) \\ (\Xi, F^\Xi, x, y, \pi) \leftarrow \mathcal{A}(\mathsf{pp}; \rho) \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \Xi) \\ \omega \leftarrow \mathcal{E}(\mathsf{pp}; \rho) \\ y' \leftarrow F^\Xi(x, \omega) \end{array}\right] \leq \mathsf{negl}(\lambda)$$

- **Zero-Knowledge**. For all PPT adversaries $\mathcal{A}$ there exists a PPT simulator $\mathcal{S}$ such that

$$\left\{ (\mathsf{pp}, \overline{F^\Xi}, x, y, \pi) \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda) \\ (\Xi, F^\Xi, x, \omega) \leftarrow \mathcal{A}(\mathsf{pp}) \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \Xi) \\ (y, \pi) \leftarrow \mathcal{P}(\mathsf{pk}, F^\Xi, x, \omega) \end{array}\right\} \cong \left\{ (\mathsf{pp}, \overline{F^\Xi}, x, y, \pi) \;\middle|\; \begin{array}{l} (\mathsf{pp}, \tau) \leftarrow \mathcal{S}(1^\lambda) \\ (\Xi, F^\Xi, x, \omega) \leftarrow \mathcal{A}(\mathsf{pp}) \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \Xi) \\ (y, \pi) \leftarrow \mathcal{S}(\mathsf{pp}, F^\Xi, x, \tau) \end{array}\right\}$$

- **Succinctness**. The proof size, and verifier time and space complexities are $\mathcal{O}(1)$.

Remark. Definition 6.1 is equivalent to a zk-SNARK for the language $\mathcal{L}_\Xi$ of computations encoded for the machine architecture Ξ. Formally, we define the language $\mathcal{L}_\Xi$ as follows

$$\mathcal{L}_\Xi = \left\{ ((F^\Xi, x, y), \omega) \mid F^\Xi(x, \omega) = y \right\}$$

In particular, a zkVM scheme Π_{zkVM} for a machine architecture Ξ is equivalent to a zk-SNARK for the language $\mathcal{L}_\Xi$ with $\mathcal{O}(1)$ proof size, and $\mathcal{O}(1)$ verifier space-time complexity.

7 The Nexus zkVM: A General-Purpose IVC Machine

Lastly, we construct the Nexus PCD prover as a series of compiler transformations, and then we use it as a black-box for constructing the Nexus zkVM. This section is not totally formal, and it is intended to provide a high-level overview of the construction. We refer the reader to the technical paper for a more detailed treatment of the construction.

Construction 7.1 (The Nexus zkVM)**.** The Nexus zkVM is a parallelized SuperNova-HyperNova-CycleFold machine with succinct proof compression. That is, let $\mathcal{R}_{\mathsf{CCS}} \in \mathsf{NPC}$ be the NP-complete CCS language for arithmetic circuit satisfiability. Consider the protocol obtained by the following sequence of transformations:

$$\Pi = \mathsf{PCD}[\mathsf{NIVC}[\mathsf{FS}[\mathsf{CF}[\mathsf{MFS}[\mathcal{R}_{\mathsf{CCS}}]]]]]$$

where

- $\mathsf{MFS}[\mathcal{R}_{\mathsf{CCS}}]$: denotes the multi-folding scheme for $(\mathcal{R}_{\mathsf{LCCCS}}, \mathcal{R}_{\mathsf{CCCS}}, \mathsf{compat}, \mu, \nu)$ from Construction 5.1.
- CF: denotes the CycleFold transformation from Construction 5.4.
- FS: denotes the Fiat-Shamir transformation for multi-folding schemes from Construction 5.3.
- NIVC: denotes the SuperNova non-uniform IVC transformation for non-interactive multi-folding schemes.
- PCD: denotes the parallelization (proof-carrying-data) transformation for NIVC schemes.[3]

Further, let $\mathsf{zkSNARK} = (\mathsf{G}, \mathsf{K}, \mathsf{P}, \mathsf{V})$ be a zk-SNARK for $\mathcal{R}_{\mathsf{CCS}}$. Let init and end be functions that set the initial state and extract the output of the machine, respectively. In particular, the init function initializes two machine tapes: T_z the machine's main memory tape, and T_ω, a read-only private memory tape, readable only through non-deterministic IVC advice.

Next, we define an augmented IVC verifier function V'_{IVC} as follows, where all arguments are taken as non-deterministic advice. This function is essentially comprises the Nexus zkVM "bootloading" process.

- $V'_{\mathsf{IVC}}(\mathsf{vk}, (\overline{F^\Xi}, x, y), (F^\Xi, (n, z_0, z_n), \omega, \pi_{\mathsf{IVC}})) \to \mathsf{x}$
  1. Check $\mathsf{VC.Open}(\mathsf{vk}, \overline{F^\Xi}, F^\Xi) = 1$
  2. Check $z_0 = \mathsf{init}(\Xi, (F^\Xi, x, \omega))$
  3. Check $\Pi.\mathsf{V}(\mathsf{vk}, (n, z_0, z_n), \pi_{\mathsf{IVC}}) = 1$
  4. Check $y = \mathsf{end}(\Xi, z_n)$
  5. Output $\mathsf{hash}(\mathsf{vk}, (\overline{F^\Xi}, x, y))$

Because V'_{IVC} can be computed in polynomial time, it can be represented as an $\mathcal{R}_{\mathsf{CCS}}$ instance $\mathsf{s}_{V'_{\mathsf{IVC}}}$. Let

$$(\mathsf{u}, \mathsf{w}) \leftarrow \mathsf{trace}(V'_{\mathsf{IVC}}, \mathsf{input})$$

denote the satisfying $\mathcal{R}_{\mathsf{CCS}}$ instance-witness pair (u, w) for the execution of V'_{IVC} on non-deterministic advice input.

Finally, we construct a parallelized zkVM scheme

$$\Pi_{\mathsf{zkVM}} = (\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$$

as follows.

[3]The problem of endowing PCD with SuperNova-like non-uniformity will be described in later work.

- $\mathcal{G}(1^\lambda) \to \mathsf{pp}$: Output $\Pi.\mathsf{G}(1^\lambda)$.
- $\mathcal{K}(\mathsf{pp}, \Xi) \to (\mathsf{pk}, \mathsf{vk})$: Output $\Pi.\mathsf{K}(\mathsf{pp}, \Xi)$.
- $\mathcal{P}(\mathsf{pk}, F^\Xi, x, \omega) \to (y, \pi)$:

  1. **Generate the IVC proof**:
     (a) Initialize the state of the machine:
        – $z_0 \leftarrow \mathsf{init}(\Xi, (F^\Xi, x, \omega))$
     (b) Generate the execution trace:
        – $(\vec{z}, \vec{w}) \leftarrow \mathsf{trace}(\Xi, z_0)$
     (c) Compute the IVC proof through the r-ary PCD parallelization tree:
        – $\pi_{\mathsf{IVC}} \leftarrow \Pi.\mathsf{P}(\mathsf{pk}, (\vec{z}, \vec{w}))$
  2. **Compress**:
     (a) Extract the output of the machine:
        – $y \leftarrow \mathsf{end}(\Xi, z_n)$
     (b) Compute the instance-witness pair for the execution of the augmented IVC verifier:
        – $(\mathsf{u}, \mathsf{w}) \leftarrow \mathsf{trace}(V'_{\mathsf{IVC}}, ((\overline{F^\Xi}, x, y), (F^\Xi, (n, z_0, z_n), \omega, \pi_{\mathsf{IVC}})))$
     (c) Generate the compressed zk-SNARK proof:
        – $\pi_{\mathsf{zkSNARK}} \leftarrow \mathsf{zkSNARK}(\mathsf{pk}, \mathsf{u}, \mathsf{w})$
  3. **Output**:
     (a) Set the final proof:
        – $\pi \leftarrow (\mathsf{u}, \pi_{\mathsf{zkSNARK}})$
     (b) Output (y, π)

- $\mathcal{V}(\mathsf{vk}, \overline{F^\Xi}, x, y, \pi) \to \{0, 1\}$:

  1. Parse π as $(\mathsf{u}, \pi_{\mathsf{zkSNARK}})$
  2. Check $\mathsf{u.x} = \mathsf{hash}(\mathsf{vk}, (\overline{F^\Xi}, x, y))$
  3. Check $\mathsf{zkSNARK.V}(\mathsf{vk}, \mathsf{u}, \pi_{\mathsf{zkSNARK}}) = 1$

8 The Nexus Network: A Verifiable Supercomputer

The Nexus Network is a physical instantiation of the IVC / PCD prover of the Nexus zkVM (Fig. 2). It is combined with a variety of algorithms that reduce communication overhead and allocate chunks of computation in a massively-parallelized work-stealing fashion (like worker threads in a gigantic CPU). Due to length (and time) constraints, the details will be released in a separate document.

References

[Tur+36] Alan Mathison Turing et al. “On computable numbers, with an application to the Entscheidungsproblem”. In: *J. of Math* 58.345-363 (1936), p. 5 (cit. on p. 12).

[Göd56] Kurt Gödel. Mar. 1956. URL: https://www.anilada.com/notes/godel-letter.pdf (cit. on p. 12).

[Bab85] László Babai. “Trading group theory for randomness”. In: *Proceedings of the seventeenth annual ACM symposium on Theory of computing*. 1985, pp. 421–429 (cit. on pp. 12, 15).

[FS86] Amos Fiat and Adi Shamir. “How to prove yourself: Practical solutions to identification and signature problems”. In: *Conference on the theory and application of cryptographic techniques*. Springer. 1986, pp. 186–194 (cit. on pp. 15, 16, 30).

[GS86] Shafi Goldwasser and Michael Sipser. “Private coins versus public coins in interactive proof systems”. In: *Proceedings of the eighteenth annual ACM symposium on Theory of computing*. 1986, pp. 59–68 (cit. on p. 15).

[BHZ87] Ravi B Boppana, Johan Hastad, and Stathis Zachos. “Does co-NP have short interactive proofs?” In: *Information Processing Letters* 25.2 (1987), pp. 127–132 (cit. on p. 15).

[Mer87] Ralph C Merkle. “A digital signature based on a conventional encryption function”. In: *Conference on the theory and application of cryptographic techniques*. Springer. 1987, pp. 369–378 (cit. on pp. 5, 15).

[BFM88] Manuel Blum, Paul Feldman, and Silvio Micali. “Non-Interactive Zero-Knowledge and Its Applications (Extended Abstract)”. In: *STOC*. 1988 (cit. on p. 12).

[FRS88] L Fortnow, J Rompel, and M Sipser. “On the power of multi-power interactive protocols”. In: *1988 Structure in Complexity Theory Third Annual Conference*. IEEE Computer Society. 1988, pp. 156–157 (cit. on pp. 12, 15).

[BFLS91] László Babai, Lance Fortnow, Leonid A Levin, and Mario Szegedy. “Checking computations in polylogarithmic time”. In: *Proceedings of the twenty-third annual ACM symposium on Theory of computing*. 1991, pp. 21–32 (cit. on pp. 12, 15).

[BFL91] László Babai, Lance Fortnow, and Carsten Lund. “Non-deterministic exponential time has two-prover interactive protocols”. In: *Computational complexity* 1 (1991), pp. 3–40 (cit. on pp. 12, 15).

[BSMP91] Manuel Blum, Alfredo De Santis, Silvio Micali, and Giuseppe Persiano. “Noninteractive Zero-Knowledge”. In: *SIAM J. Comput.* 20.6 (1991), pp. 1084–1118. DOI: 10.1137/0220068. URL: https://doi.org/10.1137/0220068 (cit. on p. 12).

[Ped91] Torben Pryds Pedersen. “Non-interactive and information-theoretic secure verifiable secret sharing”. In: *Annual international cryptology conference*. Springer. 1991, pp. 129–140 (cit. on pp. 16, 17).

[AS92] S Arora and S Safra. “Probabilistic checking of proofs; a new characterization of NP”. In: *Proceedings., 33rd Annual Symposium on Foundations of Computer Science*. IEEE. 1992, pp. 2–13 (cit. on pp. 12, 15).

[Aro+92] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. “Proof verification and hardness of approximation problems”. In: *33rd Annual Symposium on Foundations of Computer Science, FOCS 1992*. IEEE Computer Society. 1992, pp. 14–23 (cit. on pp. 12, 15).

[Kil92] Joe Kilian. “A note on efficient zero-knowledge proofs and arguments”. In: *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*. 1992, pp. 723–732 (cit. on pp. 12, 15, 16).

[LFKN92] Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. "Algebraic methods for interactive proof systems". In: *Journal of the ACM (JACM)* 39.4 (1992), pp. 859–868 (cit. on pp. 15, 18).

[SP92] Alfredo De Santis and Giuseppe Persiano. "Zero-Knowledge Proofs of Knowledge Without Interaction (Extended Abstract)". In: *33rd Annual Symposium on Foundations of Computer Science, Pittsburgh, Pennsylvania, USA, 24-27 October 1992*. IEEE Computer Society, 1992, pp. 427–436. DOI: `10.1109/SFCS.1992.267809`. URL: `https://doi.org/10.1109/SFCS.1992.267809` (cit. on p. 12).

[Sha92] Adi Shamir. "Ip= pspace". In: *Journal of the ACM (JACM)* 39.4 (1992), pp. 869–877 (cit. on p. 15).

[Sip92] Michael Sipser. "The history and status of the P versus NP question". In: *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*. 1992, pp. 603–618 (cit. on p. 12).

[BR93] Mihir Bellare and Phillip Rogaway. "Random oracles are practical: A paradigm for designing efficient protocols". In: *Proceedings of the 1st ACM Conference on Computer and Communications Security*. 1993, pp. 62–73 (cit. on p. 15).

[Gol93] Oded Goldreich. "A taxonomy of proof systems (part 1)". In: *ACM SIGACT News* 24.4 (1993), pp. 2–13 (cit. on p. 15).

[Von93] John Von Neumann. "First Draft of a Report on the EDVAC". In: *IEEE Annals of the History of Computing* 15.4 (1993), pp. 27–75 (cit. on p. 12).

[Mic94] Silvio Micali. "CS proofs". In: *Proceedings 35th Annual Symposium on Foundations of Computer Science*. IEEE. 1994, pp. 436–453 (cit. on pp. 12, 15, 16).

[96] 1996. URL: `https://www.mersenne.org/` (cit. on p. 8).

[BLS01] Dan Boneh, Ben Lynn, and Hovav Shacham. "Short signatures from the Weil pairing". In: *International conference on the theory and application of cryptology and information security*. Springer. 2001, pp. 514–532 (cit. on p. 4).

[And+02] David P Anderson, Jeff Cobb, Eric Korpela, Matt Lebofsky, and Dan Werthimer. "SETI@home: an experiment in public-resource computing". In: *Communications of the ACM* 45.11 (2002), pp. 56–61 (cit. on p. 8).

[CGH04] Ran Canetti, Oded Goldreich, and Shai Halevi. "The random oracle methodology, revisited". In: *Journal of the ACM (JACM)* 51.4 (2004), pp. 557–594 (cit. on p. 15).

[PS05] Rafael Pass and Abhi Shelat. "Unconditional characterizations of non-interactive zero-knowledge". In: *Advances in Cryptology–CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005. Proceedings 25*. Springer. 2005, pp. 118–134 (cit. on p. 15).

[DL08] Giovanni Di Crescenzo and Helger Lipmaa. "Succinct NP proofs from an extractability assumption". In: *Logic and Theory of Algorithms: 4th Conference on Computability in Europe, CiE 2008, Athens, Greece, June 15-20, 2008 Proceedings 4*. Springer. 2008, pp. 175–185 (cit. on p. 16).

[Val08] Paul Valiant. "Incrementally verifiable computation or proofs of knowledge imply time/space efficiency". In: *Theory of Cryptography: Fifth Theory of Cryptography Conference, TCC 2008, New York, USA, March 19-21, 2008. Proceedings 5*. Springer. 2008, pp. 1–18 (cit. on pp. 3, 16).

[AB09] Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009 (cit. on p. 12).

[Beb+09] Adam L Beberg, Daniel L Ensign, Guha Jayachandran, Siraj Khaliq, and Vijay S Pande. "Folding@ home: Lessons from eight years of volunteer distributed computing". In: *2009 IEEE International Symposium on Parallel & Distributed Processing*. IEEE. 2009, pp. 1–8 (cit. on p. 8).

[CT10] Alessandro Chiesa and Eran Tromer. "Proof-Carrying Data and Hearsay Arguments from Signature Cards." In: *ICS*. Vol. 10. 2010, pp. 310–331 (cit. on pp. 3, 16).

[Gro10] Jens Groth. "Short pairing-based non-interactive zero-knowledge arguments". In: *Advances in Cryptology-ASIACRYPT 2010: 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings 16*. Springer. 2010, pp. 321–340 (cit. on pp. 12, 16).

[KZG10] Aniket Kate, Gregory M Zaverucha, and Ian Goldberg. "Constant-size commitments to polynomials and their applications". In: *Advances in Cryptology-ASIACRYPT 2010: 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings 16*. Springer. 2010, pp. 177–194 (cit. on p. 16).

[BCCT12] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. "From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again". In: *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*. 2012, pp. 326–349 (cit. on pp. 12, 16).

[Ben+13] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. "SNARKs for C: Verifying program executions succinctly and in zero knowledge". In: *Advances in Cryptology–CRYPTO 2013: 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II*. Springer. 2013, pp. 90–108 (cit. on pp. 13, 17).

[BCCT13] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. "Recursive composition and bootstrapping for SNARKs and proof-carrying data". In: *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*. 2013, pp. 111–120 (cit. on p. 16).

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. "Quadratic span programs and succinct NIZKs without PCPs". In: *Advances in Cryptology–EUROCRYPT 2013: 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings 32*. Springer. 2013, pp. 626–645 (cit. on pp. 5, 12).

[BCTV14] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. "Succinct {Non-Interactive} zero knowledge for a von neumann architecture". In: *23rd USENIX Security Symposium (USENIX Security 14)*. 2014, pp. 781–796 (cit. on pp. 13, 17).

[TG14] Ben-Sasson E Chiesa A Tromer and E Virza M Garay JA Gennaro. "R Scalable zero knowledge via cycles of elliptic curves". In: *Advances in Cryptology–CRYPTO*. Vol. 2014. 2014 (cit. on p. 5).

[WLPA14] Andrew Waterman, Yunsup Lee, David A Patterson, and Krste Asanovic. "The RISC-V instruction set manual, volume I: User-level ISA, version 2.0". In: *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2014-54* (2014) (cit. on p. 4).

[Woo+14] Gavin Wood et al. "Ethereum: A secure decentralised generalised transaction ledger". In: *Ethereum project yellow paper* 151.2014 (2014), pp. 1–32 (cit. on pp. 4, 5).

[KM15] Neal Koblitz and Alfred J Menezes. "The random oracle model: a twenty-year retrospective". In: *Designs, Codes and Cryptography* 77 (2015), pp. 587–610 (cit. on p. 15).

[Zkc15] Zkcrypto. *Zkcrypto/bellman: ZK-snark library*. 2015. URL: `https://github.com/zkcrypto/bellman` (cit. on p. 12).

[BCS16] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. “Interactive oracle proofs”. In: *Theory of Cryptography: 14th International Conference, TCC 2016-B, Beijing, China, October 31-November 3, 2016, Proceedings, Part II 14*. Springer. 2016, pp. 31–60 (cit. on pp. 12, 15, 16).

[Gro16] Jens Groth. “On the size of pairing-based non-interactive arguments”. In: *Advances in Cryptology–EUROCRYPT 2016: 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II 35*. Springer. 2016, pp. 305–326 (cit. on pp. 10, 12, 16).

[AHIV17] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. “Ligero: Lightweight sublinear arguments without a trusted setup”. In: *Proceedings of the 2017 acm sigsac conference on computer and communications security*. 2017, pp. 2087–2104 (cit. on p. 16).

[BCTV17] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. “Scalable zero knowledge via cycles of elliptic curves”. In: *Algorithmica* 79 (2017), pp. 1102–1160 (cit. on pp. 13, 16).

[Haa+17] Andreas Haas, Andreas Rossberg, Derek L Schuff, Ben L Titzer, Michael Holman, Dan Gohman, Luke Wagner, Alon Zakai, and JF Bastien. “Bringing the web up to speed with WebAssembly”. In: *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*. 2017, pp. 185–200 (cit. on p. 4).

[Zok17] Zokrates. *Zokrates/Zokrates: A toolbox for zkSNARKs on ethereum*. 2017. URL: https://github.com/Zokrates/ZoKrates (cit. on p. 12).

[BBHR18a] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. “Fast reed-solomon interactive oracle proofs of proximity”. In: *45th international colloquium on automata, languages, and programming (icalp 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. 2018 (cit. on p. 16).

[BBHR18b] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. “Scalable, transparent, and post-quantum secure computational integrity”. In: *Cryptology ePrint Archive* (2018) (cit. on p. 16).

[Bün+18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. “Bulletproofs: Short proofs for confidential transactions and more”. In: *2018 IEEE symposium on security and privacy (SP)*. IEEE. 2018, pp. 315–334 (cit. on p. 16).

[Wah+18] Riad S Wahby, Ioanna Tzialla, Abhi Shelat, Justin Thaler, and Michael Walfish. “Doubly-efficient zkSNARKs without trusted setup”. In: *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2018, pp. 926–943 (cit. on p. 16).

[BGKW19] Michael Ben-Or, Shafi Goldwasser, Joe Kilian, and Avi Wigderson. “Multi-prover interactive proofs: How to remove intractability assumptions”. In: *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*. 2019, pp. 373–410 (cit. on pp. 12, 15).

[BBHR19] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. “Scalable zero knowledge with no trusted setup”. In: *Advances in Cryptology–CRYPTO 2019: 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2019, Proceedings, Part III 39*. Springer. 2019, pp. 701–732 (cit. on p. 5).

[Ben+19] Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P Ward. “Aurora: Transparent succinct arguments for R1CS”. In: *Advances in Cryptology–EUROCRYPT 2019: 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19–23, 2019, Proceedings, Part I 38*. Springer. 2019, pp. 103–128 (cit. on p. 16).

[BGH19] Sean Bowe, Jack Grigg, and Daira Hopwood. "Recursive proof composition without a trusted setup". In: *Cryptology ePrint Archive* (2019) (cit. on pp. 10, 17).

[GWC19] Ariel Gabizon, Zachary J Williamson, and Oana Ciobotaru. "Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge". In: *Cryptology ePrint Archive* (2019) (cit. on pp. 5, 16).

[GMR19] Shafi Goldwasser, Silvio Micali, and Chales Rackoff. "The knowledge complexity of interactive proof-systems". In: *Providing sound foundations for cryptography: On the work of shafi goldwasser and silvio micali*. 2019, pp. 203–225 (cit. on pp. 12, 15).

[MBKM19] Mary Maller, Sean Bowe, Markulf Kohlweiss, and Sarah Meiklejohn. "Sonic: Zero-knowledge SNARKs from linear-size universal and updatable structured reference strings". In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 2019, pp. 2111–2128 (cit. on p. 16).

[Wig19] Avi Wigderson. *Mathematics and computation: A theory revolutionizing technology and science*. Princeton University Press, 2019 (cit. on p. 12).

[Xie+19] Tiacheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song. "Libra: Succinct zero-knowledge proofs with optimal prover computation". In: *Advances in Cryptology–CRYPTO 2019: 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2019, Proceedings, Part III 39*. Springer. 2019, pp. 733–764 (cit. on p. 16).

[BGH20] Sean Bowe, Jack Grigg, and Daira Hopwood. *Zcash/Halo2: The halo2 zero-knowledge proving system*. 2020. URL: https://github.com/zcash/halo2 (cit. on pp. 10, 12, 13, 16).

[BCMS20] Benedikt Bünz, Alessandro Chiesa, Pratyush Mishra, and Nicholas Spooner. "Proof-carrying data from accumulation schemes". In: *Cryptology ePrint Archive* (2020) (cit. on pp. 13, 17, 18).

[CCDW20] Weikeng Chen, Alessandro Chiesa, Emma Dauterman, and Nicholas P Ward. "Reducing participation costs via incremental verification for ledger systems". In: *Cryptology ePrint Archive* (2020) (cit. on pp. 13, 16).

[Chi+20] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, and Nicholas Ward. "Marlin: Preprocessing zkSNARKs with universal and updatable SRS". In: *Advances in Cryptology–EUROCRYPT 2020: 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10–14, 2020, Proceedings, Part I 39*. Springer. 2020, pp. 738–768 (cit. on p. 16).

[CGTV20] Eli Ben-Sasson Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. "TinyRAM Architecture Specification v2. 000". In: (2020) (cit. on pp. 4, 6, 13).

[LNS20] Jonathan Lee, Kirill Nikitin, and Srinath Setty. "Replicated state machines without replicated execution". In: *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2020, pp. 119–134 (cit. on p. 17).

[Noi20] Noir-Lang. *Noir-lang/noir: Noir is a domain specific language for zero knowledge proofs*. 2020. URL: https://github.com/noir-lang/noir (cit. on p. 12).

[Set20] Srinath Setty. "Spartan: Efficient and general-purpose zkSNARKs without trusted setup". In: *Annual International Cryptology Conference*. Springer. 2020, pp. 704–737 (cit. on pp. 10, 15, 16, 18).

[ZXZS20] Jiaheng Zhang, Tiancheng Xie, Yupeng Zhang, and Dawn Song. "Transparent polynomial delegation and its applications to zero knowledge proof". In: *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2020, pp. 859–876 (cit. on p. 16).

[Bün+21] Benedikt Bünz, Alessandro Chiesa, William Lin, Pratyush Mishra, and Nicholas Spooner. "Proof-carrying data without succinct arguments". In: *Advances in Cryptology–CRYPTO 2021: 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16–20, 2021, Proceedings, Part I 41*. Springer. 2021, pp. 681–710 (cit. on pp. 13, 17, 18, 21).

[Chi+21] Collin Chin, Howard Wu, Raymond Chu, Alessandro Coglio, Eric McCarthy, and Eric Smith. "Leo: A programming language for formally verified, zero-knowledge applications". In: *Cryptology ePrint Archive* (2021) (cit. on pp. 12, 13).

[GPR21] Lior Goldberg, Shahar Papini, and Michael Riabzev. "Cairo–a Turing-complete STARK-friendly CPU architecture". In: *Cryptology ePrint Archive* (2021) (cit. on pp. 13, 17).

[Gol+21] Alexander Golovnev, Jonathan Lee, Srinath Setty, Justin Thaler, and Riad S Wahby. "Brakedown: Linear-time and post-quantum SNARKs for R1CS". In: *Cryptology ePrint Archive* (2021) (cit. on p. 16).

[Gra+21] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. "Poseidon: A new hash function for {Zero-Knowledge} proof systems". In: *30th USENIX Security Symposium (USENIX Security 21)*. 2021, pp. 519–535 (cit. on p. 5).

[ide21] iden3. *Iden3/Circom: Zksnark Circuit compiler*. 2021. URL: https://github.com/iden3/circom (cit. on p. 12).

[Sta21] StarkWare. *ethSTARK Documentation*. Cryptology ePrint Archive, Paper 2021/582. https://eprint.iacr.org/2021/582. 2021. URL: https://eprint.iacr.org/2021/582 (cit. on p. 5).

[Aas+22] Kaveh Aasaraai, Don Beaver, Emanuele Cesena, Rahul Maganti, Nicolas Stalder, and Javier Varela. "Fpga acceleration of multi-scalar multiplication: Cyclonemsm". In: *Cryptology ePrint Archive* (2022) (cit. on p. 4).

[ark22] arkworks. *arkworks zkSNARK ecosystem*. 2022. URL: https://arkworks.rs (cit. on pp. 9, 10, 12).

[BCKL22] Eli Ben-Sasson, Dan Carmon, Swastik Kopparty, and David Levit. "Scalable and transparent proofs over all large fields, via elliptic curves". In: *Electronic Colloquium on Computational Complexity, Report*. Vol. 110. 2022, p. 2022 (cit. on p. 5).

[KS22] Abhiram Kothapalli and Srinath Setty. "SuperNova: Proving universal machine executions without universal circuits". In: *Cryptology ePrint Archive* (2022) (cit. on pp. 3, 6, 10, 17, 18).

[KST22] Abhiram Kothapalli, Srinath Setty, and Ioanna Tzialla. "Nova: Recursive zero-knowledge arguments from folding schemes". In: *Annual International Cryptology Conference*. Springer. 2022, pp. 359–388 (cit. on pp. 3, 9, 10, 13, 16–18, 22, 30).

[Sta22] StarkWare. *Recursive Starks*. Aug. 2022. URL: https://medium.com/starkware/recursive-starks-78f8dd401025 (cit. on p. 17).

[XZS22] Tiancheng Xie, Yupeng Zhang, and Dawn Song. "Orion: Zero knowledge proof with linear prover time". In: *Annual International Cryptology Conference*. Springer. 2022, pp. 299–328 (cit. on p. 16).

[AST23] Arasu Arun, Srinath Setty, and Justin Thaler. "Jolt: Snarks for virtual machines via lookups". In: *Cryptology ePrint Archive* (2023) (cit. on pp. 6, 10, 18).

[BC23] Benedikt Bünz and Binyi Chen. "Protostar: Generic efficient accumulation/folding for special sound protocols". In: *Cryptology ePrint Archive* (2023) (cit. on pp. 3, 6, 9, 17, 18).

[CBBZ23] Binyi Chen, Benedikt Bünz, Dan Boneh, and Zhenfei Zhang. “Hyperplonk: Plonk with linear-time prover and high-degree custom gates”. In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2023, pp. 499–530 (cit. on pp. 5, 16).

[CGSY23] Alessandro Chiesa, Ziyi Guan, Shahar Samocha, and Eylon Yogev. “Security Bounds for Proof-Carrying Data from Straightline Extractors”. In: *Cryptology ePrint Archive* (2023) (cit. on p. 13).

[Cog+23] Alessandro Coglio, Eric McCarthy, Eric Smith, Collin Chin, Pranav Gaddamadugu, and Michel Dellepere. “Compositional Formal Verification of Zero-Knowledge Circuits”. In: *Cryptology ePrint Archive* (2023) (cit. on p. 14).

[CMS23] Alessandro Coglio, Eric McCarthy, and Eric W Smith. “Formal Verification of Zero-Knowledge Circuits”. In: *arXiv preprint arXiv:2311.08858* (2023) (cit. on p. 14).

[EG23] Liam Eagen and Ariel Gabizon. “ProtoGalaxy: Efficient ProtoStar-style folding of multiple instances”. In: *Cryptology ePrint Archive* (2023) (cit. on p. 17).

[GHK23] Aarushi Goel, Mathias Hall-Andersen, and Gabriel Kaptchuk. “Dora: Processor Expressiveness is (Nearly) Free in Zero-Knowledge for RAM Programs”. In: *Cryptology ePrint Archive* (2023) (cit. on pp. 6, 10).

[KT23] Tohru Kohrita and Patrick Towa. “Zeromorph: Zero-Knowledge Multilinear-Evaluation Proofs from Homomorphic Univariate Commitments”. In: *Cryptology ePrint Archive* (2023) (cit. on p. 10).

[KS23a] Abhiram Kothapalli and Srinath Setty. “CycleFold: Folding-scheme-based recursive arguments over a cycle of elliptic curves”. In: *Cryptology ePrint Archive* (2023) (cit. on pp. 3, 5, 9, 16–18, 30, 33).

[KS23b] Abhiram Kothapalli and Srinath Setty. “HyperNova: Recursive arguments for customizable constraint systems”. In: *Cryptology ePrint Archive* (2023) (cit. on pp. 3, 4, 9, 15, 17, 18, 23, 27, 30, 32).

[NBS23] Wilson Nguyen, Dan Boneh, and Srinath Setty. “Revisiting the Nova Proof System on a Cycle of Curves”. In: *Cryptology ePrint Archive* (2023) (cit. on pp. 5, 14, 16).

[OWBB23] Alex Ozdemir, Riad S Wahby, Fraser Brown, and Clark Barrett. “Bounded Verification for Finite-Field-Blasting (In a Compiler for Zero Knowledge Proofs)”. In: *Cryptology ePrint Archive* (2023) (cit. on p. 14).

[STW23a] Srinath Setty, Justin Thaler, and Riad Wahby. “Customizable constraint systems for succinct arguments”. In: *Cryptology ePrint Archive* (2023) (cit. on pp. 5, 9, 15, 18, 25).

[STW23b] Srinath Setty, Justin Thaler, and Riad Wahby. “Unlocking the lookup singularity with Lasso”. In: *Cryptology ePrint Archive* (2023) (cit. on p. 10).

[ZGGX23] Tianyu Zheng, Shang Gao, Yu Guo, and Bin Xiao. “KiloNova: Non-Uniform PCD with Zero-Knowledge Property from Generic Folding Schemes”. In: *Cryptology ePrint Archive* (2023) (cit. on pp. 6, 9, 10, 17).

[ZZD23] Zibo Zhou, Zongyang Zhang, and Jin Dong. “Proof-Carrying Data from Multi-folding Schemes”. In: *Cryptology ePrint Archive* (2023) (cit. on pp. 9, 17, 18, 28, 36).