stackproofs: Private proofs of stack and contract execution using ProtoGalaxy

Liam Eagen[1], Ariel Gabizon[2], Marek Sefranek[3], Patrick Towa[2], and Zachary J. Williamson[2]

[1]Alpen Labs
[2]Aztec Labs
[3]TU Wien

August 16, 2024

Abstract

The goal of this note is to describe and analyze a simplified variant of the zk-SNARK construction used in the Aztec protocol. Taking inspiration from the popular notion of Incrementally Verifiable Computation[Val08] (IVC) we define a related notion of *Repeated Computation with Global state* (RCG). As opposed to IVC, in RCG we assume the computation terminates before proving starts, and in addition to the local transitions some global consistency checks of the whole computation are allowed. However, we require the space efficiency of the prover to be close to that of an IVC prover not required to prove this global consistency. We show how RCG is useful for designing a proof system for a private smart contract system like Aztec.

1 Introduction

Incrementally Verifiable Computation (IVC) [Val08] and its generalization to Proof Carrying Data (PCD) [CT10] are useful tools for constructing space-efficient SNARK provers[BCCT12]. In IVC and PCD we always have an acyclic computation. However code written in almost any programming language *is* cyclic in the sense of often relying on internal calls – we start from a function A, execute some commands, go into a function B, execute its commands, and go back to A. When making a SNARK proof of such an execution, we typically linearize or "flatten" the cycle stemming from the internal call, in one of the following two ways.

1. The monolithic circuit approach: We "inline" all internal calls (as well as loops) into one long program without jumps.

2. The VM approach: Assume the code of A, B is written in some prespecified instruction set. The program is executed by initially writing the code of A, B into memory, and loading from memory and executing at each step the appropriate instruction according to a program counter. For example, the call to B is made by changing the counter to that of the first instruction of B. To prove correctness of the execution, all we need is a SNARK for proving correctness of a certain number of steps of a machine with this instruction set, and some initial memory state.

The second approach is more generic, while the first offers more room for optimization, so we'd want to use it in resource-constrained settings, e.g. client-side proving.

However, what if we're in a situation where A and B have already been "SNARKified" separately? Namely, there is a verification key attached to each one, and we are expected to use these keys specifically. This is what happens in the Aztec system.

The Aztec private contract system: Similar to Ethereum we have contracts, and the contracts have functions.[1] A function in a contract can internally call a different function in the same or a different contract. Moreover, while writing the code for the different functions and compiling them to circuits, we can't predict what function will be internally called by a given contract function. For example, a "send token" function could have an internal call to an "authorize" function. But the call to "authorize" need not be tied to a specific implementation and consequently, to a specific verification key – as different token holders are allowed to set their own "authorize" function.

The goal of the Aztec system is to enable constructing zero-knowledge proofs of such contract function executions. For this purpose, a contract is deployed by

1. Computing a verification key for each function of the contract.

2. Adding a commitment to the verification keys of the contract in a global "function tree". More accurately, a leaf of this tree is a hash of the contract address with a Merkle root of a tree whose leaves are the verification keys of that contract's functions.

Dealing with Global state The global state of the Aztec system is described by a set of notes, which are simply values in a field $\mathbb{F}$. Each note belongs to a certain contract. While running, a function can read, add or delete notes belonging to its contract. We can thus think of the notes as global variables shared between the different functions.

Assume all functions in this system return accept or reject. (We can always move the output into the arguments if a function is not of this form.) Here's a natural way to prove the mentioned execution: Put the arguments to B in the public inputs of both the circuits of A and B. Verify the proofs π_A, π_B for A, B; and check via the public inputs the same value was used in both proofs for the arguments of B.

This however, doesn't yet deal with the notes. During execution, note operations happened in a certain order. We can thus assign a *counter* equal to one for the first

[1] Detailed documentation of the Aztec protocol can be found here.

operation and increment the counter with each operation. We then need to check, for example, that if a note was read with a certain counter, it was indeed added with a smaller counter. We can include a description of the note operations performed by a function in the inputs of its circuit. This description will contain the operation type (add, del, read), the note value, and the counter. The issue is, what if A is reading a note that was added in the internal call to B? Checking the existence of an add operation with smaller counter requires a constraint *between* the inputs of both circuits. And for an execution consisting of more calls, this constraint can involve any two circuits in the call tree.

This brings us to the notion of *Repeated Computation with Global state* (RCG). In RCG we have a transition predicate taking us from one state to the next. We wish to prove we know a sequence of witnesses taking us from a legal initial state to a certain publicly known final state. This might remind the reader of the popular notion of *incrementally verifiable computation* (IVC). There are two differences.

- In RCG we are not interested in "incremental" proofs of one step, only in proofs for a whole sequence of transitions ending in a desired final state.
- In RCG we also have a *final predicate* checking a joint consistency condition between witnesses from all iterations.

One could ask, why not *only* have a final predicate that includes the transition checks? In other words, a monolithic circuit for the whole computation. The point is that in our use case the final predicate is applied to small parts of each iteration's witness – namely the note operations. As a result, the decomposition into a transition and final predicate can facilitate obtaining better prover efficiency, especially in terms of prover space. Roughly, we'll require space sufficient for storing the inputs to the final predicate, in addition to the space required to prove a single transition.

1.1 Related work

Recent work [Sou23, NDC+24] as well as ongoing work [Gro24] uses folding schemes [BCL+21, KST21] to break up proving statements about large computation into smaller statements. The objective being reducing prover memory and/or improving prover parallelism. These works have not formally defined a notion like RCG, and rather use the IVC terminology. We believe the RCG framework may be better suited for capturing the properties of these constructions. In terms of the concrete constructions, there are overlaps with this work, notably a two-pass over part of the witness to generate a random challenge. As alluded to earlier, one distinction is that these works start with a computation that is already linear, while here we start with cyclic computation and must "compile" it into a "linear" statement.

2 Preliminaries

2.1 Terminology and Conventions

We assume all algorithms described receive as an implicit parameter the security parameter λ. Similarly, all integer parameters in the paper are implicitly functions of λ, and of size at most $\mathsf{poly}(\lambda)$ unless explicitly stated otherwise.

Whenever we use the term *efficient*, we mean an algorithm running in time $\mathsf{poly}(\lambda)$. Furthermore, we assume an *object generator* $\mathcal{O}$ that is run with input λ before all protocols, and returns all fields and groups used. Specifically, in our protocol $\mathcal{O}(\lambda) = (\mathbb{F}, \mathbb{G}, g)$ where

- $\mathbb{F}$ is a field of **prime** size $r = \lambda^{\omega(1)}$.
- $\mathbb{G}$ is a group of size r.
- g is a uniformly chosen generator of $\mathbb{G}$.

We usually let the λ parameter be implicit, e.g. write $\mathbb{F}$ instead of $\mathbb{F}(\lambda)$. We denote by $\mathbb{F}_{<d}[X]$ the set of univariate polynomials over $\mathbb{F}$ of degree smaller than d. We write $\mathbb{G}$ additively.

We often denote by $[n]$ the integers $\{1, \ldots, n\}$. We use the acronym e.w.p. for "except with probability"; i.e. e.w.p. γ means with probability *at least* $1 - \gamma$.

Representing $\mathbb{G}$ Assume an injective function $R : \mathbb{G} \to \mathbb{F}^2$. Whenever we discuss $a \in \mathbb{G}$ we assume it is represented as $R(a)$. When we say for $b \in \mathbb{F}^2$ that $b \in \mathbb{G}$ we mean that there exists $a \in \mathbb{G}$ with $R(a) = b$.

2.2 Zero-Testing Assumption

Throughout this paper, we'll make use of a variant of the *Zero-Testing Assumption (ZTA)* from [LS24] given in Definition 2.1 as well as its generalization, the *t-variate Zero-Testing Assumption*, we introduce in Definition 2.2.

Definition 2.1. *Fix* $\mathsf{cm} : \mathbb{F}^M \to K$*, hash function* $\mathcal{H} : \{0,1\}^* \to \mathbb{F}$*, and integer* d*. Fix the family of functions* $\mathcal{D}$*. We say the tuple* (D, x, τ) *is a* degree d-relation for $(\mathcal{D}, \mathcal{H}, \mathsf{cm})$ *if*

1. $D \in \mathcal{D}$.
2. $f(X) := D(x, \tau)$ *is a non-zero element of* $\mathbb{F}_{\leq d}[X]$.
3. *Setting* $z := \mathcal{H}(\mathsf{cm}(x), \tau)$*, we have* $f(z) = 0$.

The Zero-Testing Assumption (ZTA) for $(\mathcal{D}, \mathcal{H}, \mathsf{cm}, d)$ *states that for any efficient* $\mathcal{A}$*, the probability that* $\mathcal{A}$ *outputs a degree* d*-relation for* $(\mathcal{D}, \mathcal{H}, \mathsf{cm})$ *is* $\mathsf{negl}(\lambda)$.

Definition 2.2 (t-variate ZTA)**.** *Fix* $\mathsf{cm} : \mathbb{F}^M \to K$*, hash function* $\mathcal{H} : \{0,1\}^* \to \mathbb{F}$*, and integer* d*. Fix the family of functions* $\mathcal{D}$*. We say the tuple* (D, x, τ) *is a* (t, d)-relation for $(\mathcal{D}, \mathcal{H}, \mathsf{cm})$ *if*

1. $D \in \mathcal{D}$.
2. $f(X_1, \ldots, X_t) := D(x, \tau)$ *is a non-zero element of* $\mathbb{F}_{\leq d}[X_1, \ldots, X_t]$.
3. *Setting* $z_i := \mathcal{H}(\mathsf{cm}(x), \tau, i)$ *for* $i \in [t]$*, we have* $f(z_1, \ldots, z_t) = 0$.

The t*-variate Zero-Testing Assumption (ZTA) for* $(\mathcal{D}, \mathcal{H}, \mathsf{cm}, d)$ *states that for any efficient* $\mathcal{A}$*, the probability that* $\mathcal{A}$ *outputs a* (t, d)*-relation for* $(\mathcal{D}, \mathcal{H}, \mathsf{cm})$ *is* $\mathsf{negl}(\lambda)$.

We prove that the univariate ZTA implies the t-variate ZTA.

Lemma 2.3. *Fix a family of functions* $\mathcal{D}$ *whose outputs are polynomials in* $\mathbb{F}_{\leq d}[X_1, \ldots, X_t]$*. Let* $\mathcal{D}_t$ *be a family of functions to be defined in the proof. Then the univariate ZTA for* $(\mathcal{D}_t, \mathcal{H}, \mathsf{cm}, d)$ *implies the* t*-variate ZTA for* $(\mathcal{D}, \mathcal{H}, \mathsf{cm}, d)$ *and* $t = \mathsf{poly}(\lambda)$.

Proof. Let $\mathcal{A}$ be an adversary against the t-variate ZTA that outputs the (t, d)-relation (D, x, τ) for $(\mathcal{D}, \mathcal{H}, \mathsf{cm})$. We construct the adversary $\mathcal{A}'$ against the univariate ZTA that outputs a degree-d relation for $(\mathcal{D}_t, \mathcal{H}, \mathsf{cm})$, where $\mathcal{D}_t$ will be the union of all the functions D_i defined in the following.

Write $f(X_1, \ldots, X_t) := D(x, \tau)$ as a polynomial in X_t over $\mathbb{F}[X_1, \ldots, X_{t-1}]$:

$$f(X_t) = \sum_{i=0}^{d} C_i(X_1, \ldots, X_{t-1}) X_t^i.$$

For $i \in [t]$, denote $z_i := \mathcal{H}(\mathsf{cm}(x), \tau, i)$. Suppose first that $f(z_1, \ldots, z_{t-1}, X_t) \not\equiv 0$. Then $\mathcal{A}'$ can output the degree-d relation (D_t, x, τ_t), where D_t is the function that computes $f_t(X) := f(z_1, \ldots, z_{t-1}, X)$ given x and $\tau_t := (\tau, t)$, deriving $z_1, \ldots, z_{t-1}$ via $\mathcal{H}$ as part of its operation. Note that $f_t(z_t) = 0$ with $z_t = \mathcal{H}(\mathsf{cm}(x), \tau_t)$.

Otherwise, there is a non-zero polynomial $C_i \in \mathbb{F}_{\leq d}[X_1, \ldots, X_{t-1}]$ which satisfies $C_i(z_1, \ldots, z_{t-1}) = 0$. If $C_i(z_1, \ldots, z_{t-2}, X_{t-1}) \not\equiv 0$, $\mathcal{A}'$ can output the degree-d relation (D_{t-1}, x, τ_{t-1}), where D_{t-1} is the function that computes $f_{t-1}(X) := C_i(z_1, \ldots, z_{t-2}, X)$ given x and $\tau_{t-1} := (\tau, t-1)$.

Recursively define degree-d relations (D_i, x, τ_i) until $i = 1$ and we have a univariate non-zero polynomial $C'_j \in \mathbb{F}_{\leq d}[X_1]$ with $C'_j(z_1) = 0$. In this base case, $\mathcal{A}'$ can output the degree-d relation (D_1, x, τ_1), where D_1 is the function that computes $f_1(X) := C'_j(X)$ given x and $\tau_1 := (\tau, 1)$, finishing the proof. □

3 The execution model

We present a formal framework for describing function executions enabling both internal function calls and global state.

We begin in Section 3.1 by introducing *record operations* which is our specific notion of operating on a global state. Record operations keep track of the computation steps at which records (roughly corresponding to notes in the introduction) were added, read and deleted. We then define what it means for a set of such operations to be *consistent*. For example, we want to enforce that a record can be deleted only if it was first added.

The eventual goal is to prove that a certain set of records is the output of a function execution (where that execution includes the function's internal calls). The proof should not reveal any information about which function was executed, as long as it belongs to a pre-defined set of legal functions, or about its arguments. To do so, it is sufficient to prove knowledge of an execution tree with the initial function at the root, the functions it calls at its children nodes and so forth. Section 3.2 defines a relation which formally captures what it means for an individual function to accept on its arguments, and Section 3.3 formally defines such execution trees.

Remark 3.1. *For simplicity, and in contrast to the introduction, we don't explicitly discuss contracts. We only model functions operating on a shared global state. As in Aztec each function can only operate on notes of its contract, this corresponds to a system with one contract. Capturing the general system mainly requires modelling the restriction that a function is only operating on the subset of notes belonging to its contract.*

3.1 Record operations

Records are pairs (v, c), where $\mathsf{v} \in \mathbb{F}$ is the *value*, and $\mathsf{c} \in [M]$ is a *counter*. A *record operation* has one of the following forms:

- $(\mathsf{add}, \mathsf{v}, \mathsf{c})$,
- $(\mathsf{del}, \mathsf{v}, \mathsf{c}_\mathsf{v}, \mathsf{c})$,
- $(\mathsf{read}, \mathsf{v}, \mathsf{c}_\mathsf{v}, \mathsf{c})$.

Here $\mathsf{v} \in \mathbb{F}$ is a value and $\mathsf{c}, \mathsf{c}_\mathsf{v} \in [M]$ are counters. c is interpreted as the counter of the current operation. c_v is interpreted as the counter of the operation where the note was added in the case of a read or del operation.

We say a sequence $\mathcal{O}$ of record operations of size M is *consistent* if

1. The counter values c are distinct in all elements of $\mathcal{O}$, and as a set equal to $[M]$.
2. The c_v fields in all del operations $(\mathsf{del}, \mathsf{v}, \mathsf{c}_\mathsf{v}, \mathsf{c}) \in \mathcal{O}$ are distinct.
3. If $(\mathsf{del}, \mathsf{v}, \mathsf{c}_\mathsf{v}, \mathsf{c}) \in \mathcal{O}$, then $\mathsf{c}_\mathsf{v} < \mathsf{c}$ and $(\mathsf{add}, \mathsf{v}, \mathsf{c}_\mathsf{v}) \in \mathcal{O}$.
4. If $(\mathsf{read}, \mathsf{v}, \mathsf{c}_\mathsf{v}, \mathsf{c}) \in \mathcal{O}$, then $\mathsf{c}_\mathsf{v} < \mathsf{c}$ and $(\mathsf{add}, \mathsf{v}, \mathsf{c}_\mathsf{v}) \in \mathcal{O}$.

Let V be a set of records. We say $\mathcal{O}$ is *has output* V if:

- $\mathcal{O}$ is consistent.
- $\mathsf{V} = \{(\mathsf{v}, \mathsf{c}) \mid (\mathsf{add}, \mathsf{v}, \mathsf{c}) \in \mathcal{O} \wedge \forall \mathsf{c}' \in [M], (\mathsf{del}, \mathsf{v}, \mathsf{c}, \mathsf{c}') \notin \mathcal{O}\}$. In words, V is the set of notes that were added and not deleted.

3.2 The Plonkish relation

Now we introduce a relation $\mathcal{R}_{app}$ describing the individual function executions tailored to make it convenient to later discuss an execution of a *sequence* functions calling each other. The executed function is represented in the instance by a single group element f. In the terminology of [GWC19], f is a commitment to the permutation and selector values of a specific $\mathcal{PLONK}$ circuit. In particular, $\mathcal{R}_{app}$ is a "universal" Plonkish relation where the circuit is not fixed but chosen in the instance.

Additionally, the instance adheres to a form containing both the record operations and the details of the inner calls of the individual function execution. We stress however, that the *interpretation* of these values as record operations and inner calls, only happens in the next section when we discuss valid executions trees; and doesn't manifest in the definition of $\mathcal{R}_{app}$. Some choices of constants – like args being of size four – are arbitrary.

We fix a polynomial $G : \mathbb{F}^8 \to \mathbb{F}$, and integers N, n that are implicit parameters in the following definition of relation $\mathcal{R}_{app}$.

$\mathcal{R}_{app}$ consists of all pairs $(\mathfrak{x}, \mathfrak{w})$ having the form

- $\mathfrak{x} = (\mathsf{f}, \mathsf{args}, \mathsf{c}, \mathsf{f}_1, \mathsf{args}_1, \mathsf{f}_2, \mathsf{args}_2, \mathcal{O})$ where $\mathsf{f}, \mathsf{f}_1, \mathsf{f}_2 \in \mathbb{G}, \mathsf{args} \in \mathbb{F}^4, \mathsf{c} \in \{0, 1, 2\}$;
- $\mathfrak{w} = (\mathsf{w}_\mathsf{f}, \omega)$ where
  - $\mathsf{w}_\mathsf{f} = (\mathsf{S}_1, \ldots, \mathsf{S}_4, \mathsf{q}_1, \ldots, \mathsf{q}_4)$,where $\mathsf{S}_j \in [|\mathfrak{x}| + N]^n, \mathsf{q}_j \in \mathbb{F}^n$ for each $j \in [4]$
  - $\omega \in \mathbb{F}^N$

such that

1. Setting $x = (\mathfrak{x}, \omega)$, for all $i \in [n]$
$$G(\mathsf{q}_{1,i}, \ldots, \mathsf{q}_{4,i}, x_{\mathsf{S}_{1,i}}, \ldots, x_{\mathsf{S}_{4,i}}) = 0.$$
2. $\mathsf{f} = \mathsf{cm}(\mathsf{w}_\mathsf{f})$.

3.3 Valid execution trees

By an *execution tree of length* n we mean a binary tree T with n vertices, whose nodes are labeled by pairs $(\mathfrak{x}, \mathfrak{w})$. Let F be a set of elements of $\mathbb{G}$. Given such T we say it is a *valid execution of length* n *with function set* F *and output* V if

1. For each $\mathsf{n} \in \mathsf{T}$, its label $(\mathfrak{x}, \mathfrak{w})$ is in $\mathcal{R}_{app}$.
2. For each $\mathsf{n} \in \mathsf{T}$, let $(\mathfrak{x}, \mathfrak{w})$ be its label. Let $\mathfrak{x} = (\mathsf{f}, \mathsf{args}, \mathsf{c}, \mathsf{f}_1, \mathsf{args}_1, \mathsf{f}_2, \mathsf{args}_2, \mathcal{O})$. Then
   - $\mathsf{f} \in$ F.
   - The number of its children is c.
   - For $i \in [\mathsf{c}]$, let $(\mathsf{f}^i, \mathsf{args}^i, \mathsf{c}^i, \mathsf{f}^i_1, \mathsf{args}^i_1, \mathsf{f}^i_2, \mathsf{args}^i_2, \mathcal{O}^i)$ denote the first component of n's i'th child's label. Then $\mathsf{f}_i = \mathsf{f}^i$ and $\mathsf{args}_i = \mathsf{args}^i$.

- Let $\mathcal{O}$ be the multi-set union of $\mathfrak{x}.\mathcal{O}$ over all nodes' labels $(\mathfrak{x}, \mathfrak{w})$. Then $\mathcal{O}$ has output V.

Given a set of group elements F say it has *Merkle root* $\mathbf{r}$ if $\mathbf{r}$ is the root of a Merkle tree with the elements of F at the leaves using some pre-determined encoding.

We define a relation $\mathcal{R}_{\text{exec}}$ capturing knowledge of an execution of bounded length with a certain output set of records. $\mathcal{R}_{\text{exec}}$ consists of the pairs $(\mathrm{x}_{\text{exec}}, \mathrm{w}_{\text{exec}})$ of the form

- $\mathrm{x}_{\text{exec}} = (\mathbf{r}, C, \mathsf{V})$,
- $\mathrm{w}_{\text{exec}} = (n, \mathsf{T})$,

such that $n \leq C$, and T is a valid execution tree of length n with function set F having Merkle root $\mathbf{r}$, and output set V.

4 Repeated Computation with Global state

Motivated by space-efficient proofs for $\mathcal{R}_{\text{exec}}$, we introduce the notion of *Repeated Computation with Global state* (RCG). RCGs enable us to deal separately with the local consistency of iterative steps of a transition function, and over-all consistency of a global state consisting of a part of each iteration's witness. We first define the general notion, and then in Section 4.1 show how to capture valid execution trees with it.

Defining RCG relations: An RCG relation is defined by a pair of functions $(F, \not\ell)$. We call $F(Z, W, Z^*, S) \to \{\mathsf{accept}, \mathsf{reject}\}$ the *transition predicate*. We informally think of

- Z as the public input and W as the private input of F.
- Z^* as the output of F (although the actual output is $\{\mathsf{accept}, \mathsf{reject}\}$).
- S as the part of the private input that will be used in the final predicate.

Let D_0, D_1, D_2 be the domains of Z, S, V respectively. $\not\ell$ is a function $\not\ell : D_0 \times D_1^* \times D_2 \to \{\mathsf{accept}, \mathsf{reject}\}$ called the *final predicate*.

The relation $\mathcal{R}_{F,\not\ell}$ is the set of pairs (x, w)

- $x = (z_{final}, C, \mathsf{V})$,
- $w = (n, z = (z_0, \ldots, z_n), w = (w_1 \ldots, w_n), s = (s_1, \ldots, s_n))$

such that

1. $z_0.\mathsf{init} = \mathsf{true}$.
2. $z_n = z_{final}$.

3. $n \leq C$.

4. For each $i \in [n]$, $F(z_{i-1}, w_i, z_i, s_i) = \mathsf{accept}$.

5. $\not\ell(z_n, s_1, \ldots, s_n, \mathsf{V}) = \mathsf{accept}$.

We say a zk-SNARK for $\mathcal{R}_{F,\not\ell}$ is *space-efficient* if given s and streaming access to z and w $\mathbf{P}$ requires space $O(|F| + |s| + \lambda \log n)$. Here $|F|$ is defined as $\mathscr{M} + n'$ where $f : \mathbb{F}^{\mathscr{M}} \to \mathbb{F}^{n'}$ is the PROTOGALAXY constraint function representing F (see Section 6.).

4.1 Valid executions as RCGs

We define a specific RCG relation $\mathcal{R}_{F,\not\ell}$ capturing valid execution trees as defined in Section 3.3. Loosely speaking, the transition function F will update a call stack of functions yet to be executed, and execute the function that is at the top of the stack. The final predicate $\not\ell$ will check the union of record operations from all iterations is consistent.

More precisely, define the function $F(Z, W, Z^*, S) \to \{\mathsf{accept}, \mathsf{reject}\}$ as follows.

- $Z = (g, \mathbf{r}, \mathsf{init})$ where g is a stack of elements of the form $(\mathsf{f}, \mathsf{args})$, $\mathbf{r}$ is a root of a Merkle tree, and init a boolean.
- $Z^* = (g^*, \mathbf{r}^*, \mathsf{init}^*)$ has the same form.
- $W = (\mathbf{p}, \mathbf{x}, \mathfrak{w})$
- S is a set of record operations.

Under this notation $F(Z, W, Z^*, S) = \mathsf{accept}$ if and only if

1. If $\mathsf{init} = \mathsf{true}$, g contains exactly one element.
2. Setting $\mathfrak{x} = (\mathbf{x}, S)$, we have $(\mathfrak{x}, \mathfrak{w}) \in \mathcal{R}_{app}$.
3. Denoting $g[0] = (\mathsf{f}, \mathsf{args})$, we have $\mathsf{f} = \mathbf{x}.\mathsf{f}$ and $\mathsf{args} = \mathbf{x}.\mathsf{args}$.
4. $\mathbf{p}$ is a Merkle path from f to $\mathbf{r}$.
5. $\mathbf{r} = \mathbf{r}^*$.
6. g^* is the result of popping $(\mathsf{f}, \mathsf{args})$ from g and then pushing the $\mathfrak{x}.\mathsf{c}$ elements $(\mathfrak{x}.\mathsf{f}_i, \mathsf{args}_i)$ for $i \in [\mathfrak{x}.\mathsf{c}]$.

Denote by g_{empty} the empty stack. We define $\not\ell(z_n, s_1, \ldots, s_n, \mathsf{V})$ to output accept if and only if

1. $z_n.g = g_{empty}$,
2. Defining $\mathcal{O}$ as the multi-set union of $s_1, \ldots, s_n$ it is as well-formed set of record operations with output V.

We show that proving knowledge of a witness for an instance of $\mathcal{R}_{\text{exec}}$ can be reduced to proving knowledge of a witness for an instance of $\mathcal{R}_{F,\ell}$.

Lemma 4.1. *There is an efficiently computable and efficiently invertible map φ such that the following holds. Let* F *be a set of function commitments with Merkle root* $\mathbf{r}$*. Fix positive integers n, C with $n \leq C$. Define $z_{final} = (g_{empty}, \mathbf{r}, \mathsf{false})$. Let* T *be an execution tree of length n.*

Then $((\mathbf{r}, C, \mathsf{V}), \mathsf{T}) \in \mathcal{R}_{\text{exec}}$ if and only if $((z_{final}, C, \mathsf{V}), \varphi(\mathsf{T})) \in \mathcal{R}_{F,\ell}$.

Proof. We describe the operation of φ. Given T of length n let $(\mathfrak{x}_1, \mathfrak{w}_1), \ldots, (\mathfrak{x}_n, \mathfrak{w}_n)$ be the labels of its nodes according to DFS order. Define a sequence of stacks $g_0, \ldots, g_n$ according to the sequence of labels.

Namely, g_0 is the stack containing only $(\mathfrak{x}_1.\mathsf{f}, \mathfrak{x}_1.\mathsf{args})$. And for each $i \in [n]$, g_i is the stack obtained by popping $g_{i-1}[0]$ and adding $(\mathfrak{x}_i.\mathsf{f}_j, \mathfrak{x}_i.\mathsf{args}_j)$ for $j \in [\mathfrak{x}_i.\mathsf{c}]$. Now, define $z_0 = (g_0, \mathbf{r}, \mathsf{true})$ and for each $i \in [n]$, $z_i = (g_i, \mathbf{r}, \mathsf{false})$.

To proceed we need to refer to the record operations in each instance separately. For this purpose, for each $i \in [n]$ denote $\mathfrak{x}_i = (\mathbf{x}_i, \mathcal{O}_i)$. For each $i \in [n]$, let $\mathbf{p}_i$ be the path from $\mathfrak{x}_i.\mathsf{f}$ to $\mathbf{r}$. Define for each $i \in [n]$, $w_i = (\mathbf{p}_i, \mathbf{x}_i, \mathfrak{w}_i)$, $s_i = \mathcal{O}_i$. Finally set $z = (z_0, \ldots, z_n), w = (w_1, \ldots, w_n), s = (s_1, \ldots, s_n)$ and $\varphi(\mathsf{T}) = (n, z, w, s)$. Given this definition of φ the statement of the lemma is straightforward to check. □

5 Removing the global state via rational identities

We give rational identities which are equivalent to the consistency of record operations as formally defined in Section 3.1. They arise from ideas similar to those used in "log-derivative lookups" [Eag22, Hab22]. Using these identities, we then define a new RCG relation $\mathcal{R}_{F^*,\ell^*}$ capturing valid execution trees, i.e., $\mathcal{R}_{\text{exec}}$. The advantage of $\mathcal{R}_{F^*,\ell^*}$ over $\mathcal{R}_{F,\ell}$ from Section 4.1 is that the final predicate is "trivial" in the sense of only depending on the output of the final iteration. As we'll see in Section 8, this makes constructing a zk-SNARK for it more convenient.

Claim 5.1. *Assume $\mathbb{F}$ has characteristic larger than $M+1$. Let* V *be a set of records and $\mathcal{O} = \{(op_i, \mathsf{v}_i, \mathsf{c}_{\mathsf{v}i}, \mathsf{c}_i)\}_{i \in [M]}$ be a set of record operations (defining $\mathsf{c}_{\mathsf{v}i} = 0$ when $op_i = \mathsf{add}$) with $\mathsf{c}_{\mathsf{v}i} < \mathsf{c}_i$ for all $i \in [M]$. Then $\mathcal{O}$ has output V if and only if the following rational function identities hold:*

1. $$\sum_{(\mathsf{v},\mathsf{c}) \in \mathsf{V}} \frac{1}{X + \mathsf{v}Y + \mathsf{c}} = \sum_{i \in [M], op_i = \mathsf{add}} \frac{1}{X + \mathsf{v}_i Y + \mathsf{c}_i} - \sum_{i \in [M], op_i = \mathsf{del}} \frac{1}{X + \mathsf{v}_i Y + \mathsf{c}_{\mathsf{v}i}}.$$

2. *For some $m \in \mathbb{F}^M$, we have*
$$\sum_{i \in [M], op_i = \mathsf{add}} \frac{m_i}{X + \mathsf{v}_i Y + \mathsf{c}_i} = \sum_{i \in [M], op_i = \mathsf{read}} \frac{1}{X + \mathsf{v}_i Y + \mathsf{c}_{\mathsf{v}i}}.$$

3.

$$\sum_{i\in[M]} \frac{1}{X+\mathsf{c}_i} = \sum_{i\in[M]} \frac{1}{X+i}.$$

Proof. We focus on the only if direction. That is, if $\mathcal{O}$ doesn't have output V one of the three identities should not hold. Let $R_{v,c} := \frac{1}{X+vY+c}$. The main fact we use is that the rational functions $\{R_{v,c}\}_{(v,c)\in\mathbb{F}^2}$ are linearly independent. Thus, $\sum_{v,c} a_{v,c}R_{v,c} = \sum b_{v,c}R_{v,c}$ implies $a_{v,c} = b_{v,c}$ for each $(v,c) \in \mathbb{F}^2$. The event of $\mathcal{O}$ not having output V means one of the following occurs.

1. The multi-set of counters $\{\mathsf{c}_i\}_{i\in[M]}$ doesn't equal $\{1,\ldots,M\}$. In this case, the LHS of the third identity will not have all one coefficients for the elements $\{R_{0,\mathsf{c}_i}\}_{i\in[M]}$ and so cannot equal the RHS.

   Note that when we are not in this case the counters in $\mathcal{O}$ are all distinct, which we assume for the next cases.

2. For some $\mathsf{v},\mathsf{c}_\mathsf{v},\mathsf{c}$, $(\mathsf{del},\mathsf{v},\mathsf{c}_\mathsf{v},\mathsf{c}) \in \mathcal{O}$ but $(\mathsf{add},\mathsf{v},0,\mathsf{c}_\mathsf{v}) \notin \mathcal{O}$; or for some $\mathsf{v},\mathsf{c}_\mathsf{v},\mathsf{c}_1 \neq \mathsf{c}_2$, $(\mathsf{del},\mathsf{v},\mathsf{c}_\mathsf{v},\mathsf{c}_1),(\mathsf{del},\mathsf{v},\mathsf{c}_\mathsf{v},\mathsf{c}_2) \in \mathcal{O}$: In the first identity RHS, we will have $R_{\mathsf{v},\mathsf{c}_\mathsf{v}}$ with coefficient in the range $\{-1,\ldots,-M\}$, while in the LHS it has coefficient one or zero.

3. For some $\mathsf{v},\mathsf{c}_\mathsf{v},\mathsf{c}$, $(\mathsf{read},\mathsf{v},\mathsf{c}_\mathsf{v},\mathsf{c}) \in \mathcal{O}$ but $(\mathsf{add},\mathsf{v},0,\mathsf{c}_\mathsf{v}) \notin \mathcal{O}$: In the second identity RHS $R_{\mathsf{v},\mathsf{c}_\mathsf{v}}$ will have a coefficient in the range $\{1,\ldots,M\}$ while in the LHS it has coefficient zero.

4. V is *not* equal to the set V' of (v,c) for which $(\mathsf{add},\mathsf{v},0,\mathsf{c}) \in \mathcal{O}$ but $(\mathsf{del},\mathsf{v},\mathsf{c},\mathsf{c}') \notin \mathcal{O}$. We look at the first identity. V' is precisely the set of (v,c) with coefficient one on the RHS, while V is the set of (v,c) with coefficient one on the LHS. Hence the first identity cannot hold in this case.

□

Defining $\mathcal{R}_{F^*,\mathscr{f}^*}$ We now define the RCG relation $\mathcal{R}_{F^*,\mathscr{f}^*}$ which will permit efficient proofs that record operations are consistent. The idea is to evaluate the above rational identities at a random point, and have the transition predicate incrementally check the evaluation is correct by a running sum. For the random evaluation point to be generated in manner that is sound but also allow for an incremental computation of the rational identity summand by summand, we compute it via a hash chain. This chain includes at each step the new set of record operations and the last hash output, which is in essence a commitment to all the record operations in previous steps. This idea also appears in [Sou23, NDC+24, Gro24]. We proceed with the formal definition.

Let F denote the function $F(Z_F, W_F, Z_F^*, S_F) \to \{\mathsf{accept},\mathsf{reject}\}$ from Section 4.1. Let

$k := |S_F|$, $\mathsf{cm} : \mathbb{F}^{5k} \to \mathbb{G}$, and set $M := kn$.

Define the function $F^* : (Z, W, Z^*) \to \{\mathsf{accept}, \mathsf{reject}\}$:

- $Z = (Z_F, \hbar, \jmath, \alpha, \beta, \varepsilon)$.
- $Z^* = (Z_F^*, \hbar^*, \jmath^*, \alpha^*, \beta^*, \varepsilon^*)$.
- $W = (W_F, S_F, m)$

Under this notation $F^*(Z, W, Z^*) = \mathsf{accept}$ if and only if

1. $F(Z_F, W_F, Z_F^*, S_F) = \mathsf{accept}$.
2. If $Z_F.\mathsf{init} = \mathsf{true}$, $\hbar = \emptyset$ and $\jmath = 0$.
3. Let $S_F = \{(op_i, \mathsf{v}_i, \mathsf{c}_{\mathsf{v}i}, \mathsf{c}_i)\}_{i\in[k]}$. We have $\mathsf{c}_{\mathsf{v}i} < \mathsf{c}_i$ for all $i \in [k]$, and $\jmath^* = \jmath +$
$$\sum_{i\in[k];op_i=\mathsf{add}} \frac{1+\varepsilon m_i}{\alpha+\beta\mathsf{v}_i+\ \mathsf{c}_i} - \sum_{i\in[k];op_i=\mathsf{read}} \frac{\varepsilon}{\alpha+\beta\mathsf{v}_i+\mathsf{c}_{\mathsf{v}i}} - \sum_{i\in[k];op_i=\mathsf{del}} \frac{1}{\alpha+\beta\mathsf{v}_i+\mathsf{c}_{\mathsf{v}i}} + \sum_{i\in[k]} \frac{\varepsilon^2}{\alpha+\mathsf{c}_i}.$$
4. $\alpha^* = \alpha, \beta^* = \beta, \varepsilon^* = \varepsilon$.
5. $\hbar^* = \mathcal{H}(\hbar, \mathsf{cm}(S_F, m))$.

$\mathscr{f}^*(Z, \mathsf{V}) = \mathsf{accept}$ if and only if

1. $\jmath = \sum_{(\mathsf{v},\mathsf{c})\in\mathsf{V}} \frac{1}{\alpha+\beta\mathsf{v}+\mathsf{c}} + \sum_{i\in[M]} \frac{\varepsilon^2}{\alpha+i}$,
2. $\mathcal{H}(\hbar, \mathsf{V}, 1) = \alpha$, $\mathcal{H}(\hbar, \mathsf{V}, 2) = \beta$, $\mathcal{H}(\hbar, \mathsf{V}, 3) = \varepsilon$.

We show that $\mathcal{R}_{F^*,\mathscr{f}^*}$ captures $\mathcal{R}_{F,\mathscr{f}}$ and consequently valid executions.

Lemma 5.2. *There is an efficiently computable and efficiently invertible map φ such that the following holds. Let* F *be a set of function commitments with Merkle root* $\mathbf{r}$*. Fix positive integers n, C with $n \le C$. Fix some $\alpha, \beta, \varepsilon \in \mathbb{F}$ and set of records V. Define $z_{final} = (g_{empty}, \mathbf{r}, \mathsf{false}), z^*_{final} := (z_{final}, \hbar, \jmath, \alpha, \beta, \varepsilon)$ and set $\phi := (z^*_{final}, C, \mathsf{V})$. Suppose an efficient adversary $\mathcal{A}$ outputs ω such that $(\phi, \omega) \in \mathcal{R}_{F^*,\mathscr{f}^*}$.*

Set $d := 3M + |\mathsf{V}| + 1$. Define $\mathsf{cm}'((s_1, m_1), \ldots, (s_n, m_n)) := h_n$, where $h_0 := \emptyset$ and $h_i := \mathcal{H}(h_{i-1}, \mathsf{cm}(s_i, m_i))$ for $i \in [n]$. Let $\mathcal{D}$ be a function family to be defined in the proof. Assume the 3-variate ZTA holds for $(\mathcal{D}, \mathcal{H}, \mathsf{cm}', d)$. Then e.w.p. $\mathsf{negl}(\lambda)$, $((z_{final}, C, \mathsf{V}), \varphi(\omega)) \in \mathcal{R}_{F,\mathscr{f}}$.

Proof. Given a witness $\omega = (n, z', w')$ with $z' = (z'_0, \ldots, z'_n), w' = (w'_1, \ldots, w'_n)$ output by $\mathcal{A}$, denote $z'_i = (z_i, \hbar_i, \jmath_i, \alpha_i, \beta_i, \varepsilon_i), w'_i = (w_i, s_i, m_i)$. Define $\mathscr{w} = \varphi(\omega)$ as $\mathscr{w} := (n, (z_0, \ldots, z_n), (w_1, \ldots, w_n), (s_1, \ldots, s_n))$. From $(\phi, \omega) \in \mathcal{R}_{F^*,\mathscr{f}^*}$, we know $\mathscr{w}$ satisfies

the transition constraints, namely for $i \in [n]$, $F(z_{i-1}, w_i, z_i, s_i) = \mathsf{accept}$. We also know that $z_0.\mathsf{init} = \mathsf{true}$, $z_n = z_{final}$, and $n \leq C$. It is left to show that e.w.p. $\mathsf{negl}(\lambda)$ $\not\in(z_n, s_1, \ldots, s_n, \mathsf{V}) = \mathsf{accept}$.

From $((z^*_{final}, C, \mathsf{V}), \omega) \in \mathcal{R}_{F^*, \not\in^*}$ we know the equations from Claim 5.1 hold at $\alpha, \beta, \varepsilon$. That is, defining the rational function

$$r(X,Y,Z) := \sum_{(\mathsf{v},\mathsf{c})\in\mathsf{V}} \frac{1}{X+\mathsf{v}Y+\mathsf{c}} - \sum_{i\in[M],op_i=\mathsf{add}} \frac{1}{X+\mathsf{v}_iY+\mathsf{c}_i} + \sum_{i\in[M],op_i=\mathsf{del}} \frac{1}{X+\mathsf{v}_iY+\mathsf{c}_{\mathsf{v}i}}$$
$$+ Z\left(\sum_{i\in[M],op_i=\mathsf{add}} \frac{(m_1\|\ldots\|m_n)_i}{X+\mathsf{v}_iY+\mathsf{c}_i} - \sum_{i\in[M],op_i=\mathsf{read}} \frac{1}{X+\mathsf{v}_iY+\mathsf{c}_{\mathsf{v}i}}\right)$$
$$+ Z^2\left(\sum_{i\in[M]} \frac{1}{X+\mathsf{c}_i} - \sum_{i\in[M]} \frac{1}{X+i}\right),$$

we have $r(\alpha, \beta, \varepsilon) = 0$. If any of the three rational identities from Claim 5.1 does not hold, we have $r(X, Y, Z) \not\equiv 0$. Let $f \in \mathbb{F}_{\leq d}[X, Y, Z]$ denote the resulting non-zero degree-d polynomial when multiplying $r(X, Y, Z)$ with all of its denominators. Note that $f(\alpha, \beta, \varepsilon) = 0$. Define D as the function that computes $f(X, Y, Z)$ given $x := ((s_1, m_1), \ldots, (s_n, m_n))$ and $\tau := \mathsf{V}$. Set $\mathcal{D} := \{D\}$. Then we can define an efficient adversary $\mathcal{A}'$ against the 3-variate ZTA for $(\mathcal{D}, \mathcal{H}, \mathsf{cm}', d)$ that outputs the $(3, d)$-relation (D, x, τ), which has probability $\mathsf{negl}(\lambda)$. □

6 Non-interactive folding schemes

We fix a vector space K over $\mathbb{F}$, and an $\mathbb{F}$-linear function $\mathsf{cm} : \mathbb{F}^M \to K$ that will be an implicit parameter in Definition 6.1. We say a relation $\mathcal{R}$ is cm-*compatible* if every element of $\mathcal{R}$ has the form $(\mathsf{cm}(\omega), \omega)$. We say $\mathcal{R}$ is cm-*extendable* if every element of $\mathcal{R}$ has the form $((\mathsf{cm}(\omega), \tau), \omega)$ for some ω, τ.

Definition 6.1. *Fix relations $\mathcal{R}$ and $\mathcal{R}_{\mathfrak{acc}}$ that are respectively* cm*-compatible and* cm*-extendable. An $(\mathcal{R} \mapsto \mathcal{R}_{\mathfrak{acc}})$-*folding scheme *is a pair of algorithms $(\mathbf{P}, \mathbf{V})$ such that*

1. $\mathbf{P}$ *on input $(\Phi, \phi'; \omega, \omega')$ produces a pair (Φ^*, ω^*) and proof π.*
2. $\mathbf{V}$ *on input $(\Phi, \phi', \Phi^*, \pi)$ outputs* accept *or* reject *such that*
   (a) ***Completeness:** If $(\Phi, \omega) \in \mathcal{R}_{\mathfrak{acc}}$, $(\phi', \omega') \in \mathcal{R}$ and $\mathbf{P}(\Phi, \phi'; \omega, \omega') = (\pi, \Phi^*, \omega^*)$, then with probability $1 - \mathsf{negl}(\lambda)$, $(\Phi^*, \omega^*) \in \mathcal{R}_{\mathfrak{acc}}$ and $\mathbf{V}(\Phi, \phi', \Phi^*, \pi) = \mathsf{accept}$.*
   (b) ***Knowledge soundness given extractable commitments:***
   For any efficient $\mathcal{A}$ the probability of the following event is $\mathsf{negl}(\lambda)$: $\mathcal{A}$ outputs $(\phi, \tau), \phi', (\phi^*, \tau^*), \omega, \omega', \omega^*, \pi$ such that
      i. $\mathsf{cm}(\omega) = \phi, \mathsf{cm}(\omega') = \phi', \mathsf{cm}(\omega^*) = \phi^*$,

ii. $\mathbf{V}(\phi, \phi', \phi^*, \pi) = \mathsf{accept}$,
iii. $((\phi^*, \tau^*), \omega^*) \in \mathcal{R}_{\mathfrak{acc}}$,
iv. $((\phi, \tau), \omega) \notin \mathcal{R}_{\mathfrak{acc}}$ *or* $(\phi', \omega') \notin \mathcal{R}$.

Remark 6.2. *The justification for requiring only knowledge soundness given extractable commitments is as follows. We assume the Algebraic Group Model [FKL18] and use a commitment scheme based on linear combination of group elements like [KZG10]. In this model with such a commitment scheme an adversary $\mathcal{A}$ must output ω, with $\mathsf{cm}(\omega) = \phi$ whenever it outputs some $\phi \in K$. For more details see [FKL18] or Section 2.2 of [GWC19], as well as Section 7 of this paper.*

6.1 Relations for folding schemes

We define a more general satisfiability relation than in [EG23]. We have as parameters, integers $n, \mathscr{M}, d$ and an $\mathbb{F}$-vector space K. We have a

- *Constraint function* $f : \mathbb{F}^{\mathscr{M}} \to \mathbb{F}^n$ which is a vector of n $\mathscr{M}$-variate polynomials of degree $\leq d$,
- *Instance predicate* $\mathsf{f} : K \to \{\mathsf{accept}, \mathsf{reject}\}$,
- *Commitment function* $\mathsf{cm} : \mathbb{F}^{\mathscr{M}} \to K$ which is $\mathbb{F}$-linear and assumed to be collision resistant.

Given $(f, \mathsf{f}, \mathsf{cm})$ we define a relation $\mathcal{R}_{f,\mathsf{f},\mathsf{cm}}$ consisting of all pairs (ϕ, ω) such that

1. $f(\omega) = 0^n$.
2. $\mathsf{f}(\phi) = \mathsf{accept}$.
3. $\phi = \mathsf{cm}(\omega)$.

The relation $\mathcal{R}^{\mathsf{rand}}$: For brevity, let $\mathcal{R} = \mathcal{R}_{f,\mathsf{f},\mathsf{cm}}$. As in [EG23], we define the "randomized relaxed" version of $\mathcal{R}, \mathcal{R}^{\mathsf{rand}}$. First, some required notation. Let $t := \log n$. For $i \in [n]$, let $S \subset \{0, \ldots, t-1\}$ be the set such that $i - 1 = \sum_{j \in [S]} 2^j$. We define the t-variate polynomial pow_i as

$$\mathsf{pow}_i(X_0, \ldots, X_{t-1}) := \prod_{\ell \in S} X_\ell.$$

Note that if $\boldsymbol{\beta} = (\beta, \beta^2, \beta^4, \ldots, \beta^{2^{t-1}})$, $\mathsf{pow}_i(\boldsymbol{\beta}) = \beta^{i-1}$.

Given the above notation, $\mathcal{R}^{\mathsf{rand}}$ consists of the pairs (Φ, ω) with $\Phi = (\phi, \boldsymbol{\beta}, e)$ such that

1. $\phi = \mathsf{cm}(\omega)$.

2. $\boldsymbol{\beta} \in \mathbb{F}^t, e \in \mathbb{F}$ and we have

$$\sum_{i \in [n]} \mathsf{pow}_i(\boldsymbol{\beta}) f_i(\omega) = e.$$

(Here, f_i denotes the i'th output coordinate of f.)

6.2 The PROTO**galaxy** scheme

Deviating from [EG23], we explicitly present PROTO**GALAXY** as a *non-interactive* folding scheme, and for the special case of folding $k = 1$ instances. We define $Z(X) := X \cdot (1-X)$. We assume below $\mathcal{H}$ is a function mapping arbitrary strings to elements of $\mathbb{F}^*$.

$\underline{\mathbf{P}_{PG}(\Phi = (\phi, \boldsymbol{\beta}, e), \phi_1; \omega, \omega_1):}$

1. Compute $\delta = \mathcal{H}(\Phi, \phi_1)$. Define $\boldsymbol{\delta} := (\delta, \delta^2, \ldots, \delta^{2^{t-1}}) \in \mathbb{F}^t$.
2. Compute the polynomial

$$F(X) := \sum_{i \in [n]} \mathsf{pow}_i(\boldsymbol{\beta} + X\boldsymbol{\delta}) f_i(\omega).$$

   (Note that $F(0) = \sum_{i \in [n]} \mathsf{pow}_i(\boldsymbol{\beta}) f_i(\omega) = e$.)
3. Denote the non-constant coefficients of F by $a := (F_1, \ldots, F_t)$.
4. Compute $\alpha = \mathcal{H}(\Phi, \phi_1, a)$.
5. Compute $\boldsymbol{\beta}^* \in \mathbb{F}^t$ where $\boldsymbol{\beta}^*_i := \boldsymbol{\beta}_i + \alpha \cdot \boldsymbol{\delta}_i$.
6. Define the polynomial $G(X)$ as

$$G(X) := \sum_{i \in [n]} \mathsf{pow}_i(\boldsymbol{\beta}^*) f_i(X \cdot \omega + (1 - X)\omega_1).$$

7. Compute the polynomial $K(X)$ such that

$$G(X) = F(\alpha)X + Z(X)K(X).$$

8. Let $b := (K_0, \ldots, K_{d-2})$ be the coefficients of $K(X)$.
9. Compute $\gamma = \mathcal{H}(\Phi, \phi_1, a, b)$.
10. Compute

$$e^* := F(\alpha)\gamma + Z(\gamma)K(\gamma).$$

Finally, output

- the instance $\Phi^* = (\phi^*, \boldsymbol{\beta}^*, e^*)$, where

$$\phi^* := \gamma \cdot \phi + (1-\gamma)\phi_1,$$

- the witness $\omega^* := \gamma \cdot \omega + (1-\gamma) \cdot \omega_1$,
- and the proof $\pi := (a, b)$.

$\underline{\mathbf{V}_{PG}(\Phi, \phi_1, \Phi^*, \pi = (a,b)) :}$

1. Check that $\mathsf{f}(\phi_1) = \mathsf{accept}$. Output reject otherwise.
2. Compute $\delta, \alpha, \boldsymbol{\beta}^*, \gamma$ as in the prover algorithm given Φ, ϕ_1, a, b.
3. Check that Φ^* is computed as in the prover algorithm. Output accept iff this is the case.

For the knowledge soundness analysis we'll use a variant of the Zero-Testing Assumption from [LS24], see Definition 2.1.

Theorem 6.3. *Set* $d' := \max\{n, d\}$. *Denote* $\mathsf{cm}'((\omega, \boldsymbol{\beta}, e), \omega_1) := ((\mathsf{cm}(\omega), \boldsymbol{\beta}, e), \mathsf{cm}(\omega_1))$. *Let* $\mathcal{D}$ *be a family of four functions to be defined in the proof. Assume that* cm *is collision resistant and the ZTA holds for* $(\mathcal{D}, \mathcal{H}, \mathsf{cm}', d')$. *Then* ProtoGalaxy *is a* $(\mathcal{R} \mapsto \mathcal{R}^{\mathsf{rand}})$*-folding scheme.*

Proof. The main thing to prove is knowledge soundness given extractable commitments. Fix any efficient $\mathcal{A}$. Let E be the event that $\mathcal{A}$ outputs $\Phi = (\phi, \boldsymbol{\beta}, e), \phi_1, \Phi^* = (\phi^*, \boldsymbol{\beta}^*, e^*)$, $\omega, \omega_1, \omega^*, \pi$ such that

1. $\mathsf{cm}(\omega) = \phi, \mathsf{cm}(\omega_1) = \phi_1, \mathsf{cm}(\omega^*) = \phi^*$,
2. $\mathbf{V}_{PG}(\Phi, \phi_1, \Phi^*, \pi) = \mathsf{accept}$,
3. $(\Phi^*, \omega^*) \in \mathcal{R}^{\mathsf{rand}}$,
4. $(\Phi, \omega) \notin \mathcal{R}^{\mathsf{rand}}$ or $(\phi_1, \omega_1) \notin \mathcal{R}$.

According to Definition 6.1, knowledge soundness is equivalent to E having probability $\mathsf{negl}(\lambda)$ for any such $\mathcal{A}$. We construct an efficient $\mathcal{A}'$ that runs $\mathcal{A}$, and when E occurs outputs either a collision of cm or a degree d'-relation for $(\mathcal{H}, \mathsf{cm})$. By the theorem assumption this implies E is contained in two events of probability $\mathsf{negl}(\lambda)$, and must have probability $\mathsf{negl}(\lambda)$ itself.

Assume we are in event E. Using linearity of cm, when E occurs we have

$$\mathsf{cm}(\gamma\omega + (1-\gamma)\omega_1) = \gamma\phi + (1-\gamma)\phi_1 = \phi^* = \mathsf{cm}(\omega^*).$$

Thus, if $\omega^* \neq \gamma\omega + (1-\gamma)\omega_1$, $\mathcal{A}'$ can output $(\omega^*, \gamma\omega + (1-\gamma)\omega_1)$ as a collision of cm. Now assume that $\omega^* = \gamma\omega + (1-\gamma)\omega_1$. Suppose $\pi = (a, b)$, with $a = (a_1, \ldots, a_t), b =$

$(b_0, \ldots, b_{d-2})$. Define $F_0(X) := e + \sum_{i\in[t]} a_i X^i, K'(X) := \sum_{i=0}^{d-2} b_i X^i$. Let $\delta, \boldsymbol{\delta}, \alpha, \boldsymbol{\beta}^*, \gamma$ be computed as in the prover description given a, b. Define the polynomials

$$F'(X) := F_0(X) - \sum_{i\in[n]} \mathsf{pow}_i(\boldsymbol{\beta} + X\boldsymbol{\delta}) f_i(\omega),$$
$$G'(X) := F_0(\alpha)X + Z(X)K'(X) - \sum_{i\in[n]} \mathsf{pow}_i(\boldsymbol{\beta}^*) f_i(X\omega + (1-X)\omega_1).$$

Since $((\phi^*, \boldsymbol{\beta}^*, e^*), \omega^*) \in \mathcal{R}^{\mathsf{rand}}$ and $\mathbf{V}_{PG}(\Phi, \phi', \Phi^*, \pi) = \mathsf{accept}$,

$$\begin{aligned} G'(\gamma) &= F_0(\alpha)\gamma + Z(\gamma)K'(\gamma) - \sum_{i\in[n]} \mathsf{pow}_i(\boldsymbol{\beta}^*) f_i(\gamma\omega + (1-\gamma)\omega_1) \\ &= e^* - \sum_{i\in[n]} \mathsf{pow}_i(\boldsymbol{\beta}^*) f_i(\omega^*) = e^* - e^* = 0. \end{aligned}$$

Set $x := ((\omega, \boldsymbol{\beta}, e), \omega_1)$ and $\tau_1 := (a, b)$. If $G' \not\equiv 0$, $\mathcal{A}'$ outputs the degree d'-relation (D_1, x, τ_1), where D_1 is the function that computes $G'(X)$ given x, τ_1.

Assume now that $G' \equiv 0$.

If $(\phi_1, \omega_1) \notin \mathcal{R}$, using $Z(0) = 0$ we have

$$G'(0) = -\sum_{i\in[n]} \mathsf{pow}_i(\boldsymbol{\beta}^*) f_i(\omega_1) = 0.$$

Define the polynomial $A(X) := \sum_{i\in[n]} f_i(\omega_1)\mathsf{pow}_i(\boldsymbol{\beta}_1 + X\delta, \boldsymbol{\beta}_2 + X\delta^2, \ldots, \boldsymbol{\beta}_t + X\delta^{2^{t-1}})$. We have $A(\alpha) = 0$. Suppose first that $A(X) \not\equiv 0$. Then setting D_2 to be the function that computes $A(X)$ given x and $\tau_2 := a$, $\mathcal{A}'$ can output the degree n relation (D_2, x, τ_2). Now assume $A(X) \equiv 0$. Define the polynomial

$$B(X, Y) := \sum_{i\in[n]} f_i(\omega_1)\mathsf{pow}_i(\boldsymbol{\beta}_1 + XY, \boldsymbol{\beta}_2 + XY^2, \ldots, \boldsymbol{\beta}_t + XY^{2^{t-1}}).$$

We have that $B(X, \delta) \equiv 0$. Write $B(X, Y)$ as a polynomial in X over $\mathbb{F}[Y]$:

$$B(X) = \sum_{i=0}^{t} C_i(Y) X^i.$$

Because we're in the case $(\phi_1, \omega_1) \notin \mathcal{R}$, B is a combination of the n linearly independent polynomials $\left\{\mathsf{pow}_i(\boldsymbol{\beta}_1 + XY, \boldsymbol{\beta}_2 + XY^2, \ldots, \boldsymbol{\beta}_t + XY^{2^{t-1}})\right\}_{i\in[n]}$ with at least one non-zero coefficient, and so $B(X, Y) \not\equiv 0$. This means one of the polynomials C_i is non-zero, while $C_i(\delta) = 0$. We can use this to let $\mathcal{A}'$ output the degree n relation (D_3, x, τ_3), where D_3 is the function that computes C_i given x and $\tau_3 := \emptyset$.

Now, assume that $(\Phi, \omega) \notin \mathcal{R}^{\mathsf{rand}}$. As we're still assuming $G' \equiv 0$, we have

$$G'(1) = F_0(\alpha) - \sum_{i\in[n]} \mathsf{pow}_i(\boldsymbol{\beta}^*) f_i(\omega) = 0.$$

But we also have $F'(\alpha) = G'(1)$ and so $F'(\alpha) = 0$. On the other hand,

$$F'(0) = e - \sum_{i \in [n]} \mathsf{pow}_i(\boldsymbol{\beta}) f_i(\omega) \neq 0.$$

Setting $\tau_4 := a$ and D_4 to be the function that computes F' given x, τ_4, we have that (D_4, x, τ_4) is a degree $\log n$ relation that $\mathcal{A}'$ can output in this case. Setting $\mathcal{D} = \{D_1, D_2, D_3, D_4\}$ we have proven knowledge soundness under the theorem assumptions. □

7 Adversaries supporting recursive extraction

Following [LS24], we define a model of "recursive extraction" for our analysis in Section 8. We assume our commitment function $\mathsf{cm} : \mathbb{F}^{\mathscr{M}} \to K$ is surjective. We assume the existence of an efficiently computable injective representation function $R : K \to \mathbb{F}^{\mathscr{l}}$. Whenever an adversary $\mathcal{A}$ outputs $a \in K$ we assume it is represented as $R(a)$. When analyzing knowledge soundness of our zk-SNARK in the next section, we put the following limitation on $\mathcal{A}$. Say $\mathcal{A}$ outputs a vector v over $\mathbb{F}$. If there is an index i such that $(v_i, \ldots, v_{i+\mathscr{l}-1}) = R(a)$ for some $a \in K$, then $\mathcal{A}$ must also output $\omega \in \mathbb{F}^{\mathscr{M}}$ such that $\mathsf{cm}(\omega) = a$. We call such $\mathcal{A}$ a *recursive algebraic adversary.*

This assumption is motivated by a "recursive" interpretation of the Algebraic Group Model[FKL18] as done in [LS24]. For illustration, consider first the case where $\mathsf{cm} : \mathbb{F}^{n} \to \mathbb{G}$ is defined as $\mathsf{cm}(\omega) = \langle \omega, V \rangle$ for a fixed vector $V \in \mathbb{G}^n$ derived in a setup procedure. I.e. cm is a Pedersen commitment in this case. In this case, the AGM forces $\mathcal{A}$ when outputting $a \in \mathbb{G}$ to also output ω such that $\langle \omega, V \rangle = \mathsf{cm}(\omega) = a$. [LS24] now raise the idea that if a prespecified subset of indices of ω is a valid representation of $b \in \mathbb{G}$, it is reasonable to demand of an algebraic $\mathcal{A}$ to also output ω_2 with $\langle \omega_2, V \rangle = b$.

Our concrete choice for cm when using ProtoGalaxy, is of the following form.[2] We have a setup procedure outputting a vector of group elements $V \in \mathbb{G}^n$ as output. We have some fixed partition of our input $\omega \in \mathbb{F}^{\mathscr{M}}$ into continuous segments of size either one or n. To get $\mathsf{cm}(\omega)$ we operate segment-wise. If the segment ω_i is of size one we simply append ω_i to the output. If a segment $s = (\omega_i, \ldots, \omega_{i+n-1})$ is of size n we append $\langle s, V \rangle$ to the output. In particular the output $\mathsf{cm}(\omega)$ is a mixture of $\mathbb{F}$ and $\mathbb{G}$ elements, and accordingly the space K is a direct sum of spaces $\mathbb{V}_i \in \{\mathbb{F}, \mathbb{G}\}$. It follows that an algebraic adversary outputting $c \in K$ must output ω with $\mathsf{cm}(\omega) = c$ as the AGM forces it to send an opening $s \in \mathbb{F}^n$ to each $a \in \mathbb{G}$.

[2] cm is of this form because it is derived from an interactive protocol where the prover messages are in committed form, but verifier challenges are in the clear. See [BC23, EG23] for more details.

8 The main construction

We say an RCG relation $\mathcal{R}_{F,\ell}$ is *trivial* if the S variable is not used in F, and accordingly ℓ depends only on (z_{final}, V). We assume in this section $R_{F,\ell}$ is trivial, thinking of it as $\mathcal{R}_{F^*,\ell^*}$ from Section 5.

8.1 The extended function F'

As in [KST21, BCL+21], we first describe an extended function F' that executes the folding verifier in addition to an iteration of F. Loosely speaking, extending F into F' is what enables bootstrapping a folding scheme into an IVC or RCG. We describe the function arguments first.

$\mathfrak{x} = (z, \mathsf{count}, \hbar)$

- z – output for F
- count – counter of IVC step
- $\hbar$ – hash of accumulator

$\mathfrak{w} = (\Phi^*, \Phi, \mathfrak{x}_0, w, \pi)$

- Φ^* – current accumulator instance
- Φ – previous accumulator instance
- $\mathfrak{x}_0$ – instance (of F') to be accumulated.
- w – private input for F
- π – proof for ProtoGalaxy verifier

$F'(\mathfrak{x}, \mathfrak{w}) = \mathsf{accept}$ if and only if:

1. $F(\mathfrak{x}_0.z, w, z) = \mathsf{accept}$
2. If $\mathsf{count} > 1$:
   (a) $\mathcal{H}(\Phi^*) = \hbar$.
   (b) $\mathfrak{x}_0.\hbar = \mathcal{H}(\Phi)$.
   (c) $\mathfrak{x}_0.\mathsf{count} = \mathsf{count} - 1$.
   (d) $\mathbf{V}_{PG}(\Phi, \mathfrak{x}_0, \pi, \Phi^*) = \mathsf{accept}$.
3. If $\mathsf{count} = 1$:
   - $\mathfrak{x}_0.z.\mathsf{init} = \mathsf{true}$

Let $\mathcal{R}, \mathcal{R}^{\mathsf{rand}}$ be relations for the PROTOGALAXY version of the function F' above.[3]

For readability, from now on we modify notational conventions and denote an accumulator by acc, accumulator witness by $\mathbf{w}$-acc, instance by inst, and instance witness by $\boldsymbol{w}$-inst.

8.2 The relation $\mathcal{R}_{\mathsf{fin}}$

We define the relation $\mathcal{R}_{\mathsf{fin}}$ of pairs $(\boldsymbol{x}, \boldsymbol{w})$ such that

- $\boldsymbol{x} = (\mathsf{acc}, \mathsf{V}, C, z_{final})$
- $\boldsymbol{w} = (\mathbf{w}\text{-}\mathsf{acc}, \mathsf{acc}_0, \mathsf{inst}, \pi)$

such that

1. $(\mathsf{acc}, \mathbf{w}\text{-}\mathsf{acc}) \in \mathcal{R}^{\mathsf{rand}}$.
2. $\mathbf{V}_{PG}(\mathsf{acc}_0, \mathsf{inst}, \pi, \mathsf{acc}) = \mathsf{accept}$.
3. $\mathcal{H}(\mathsf{acc}_0) = \mathsf{inst}.\hbar$.
4. $\mathsf{inst}.z = z_{final}$.
5. $\mathsf{inst}.\mathsf{count} \leq C$.

Let $(\mathbf{P}_{\mathsf{fin}}, \mathbf{V}_{\mathsf{fin}})$ be a zk-SNARK for $\mathcal{R}_{\mathsf{fin}}$.

8.3 Main construction

We now describe the full prover and verifier for a given trivial RCG relation $\mathcal{R}_{F,\mathscr{f}}$. Later we review how to use this to get a zk-SNARK for the relation $\mathcal{R}_{\text{exec}}$ of valid executions, given the reductions of previous sections.

$\underline{\mathbf{P}(x, w)}$:

1. Let $\boldsymbol{x} = (z_{final}, C, \mathsf{V}), \boldsymbol{w} = (n, (z_0, \ldots, z_n), (w_1, \ldots, w_n))$. Recall that $(\boldsymbol{x}, \boldsymbol{w}) \in \mathcal{R}_{F,\mathscr{f}}$ implies $F(z_{i-1}, w_i, z_i) = \mathsf{accept}$ for each $i \in [n]$, $z_0.\mathsf{init} = \mathsf{true}, z_n = z_{final}$ and $\mathscr{f}(z_{final}, \mathsf{V}) = \mathsf{true}$.
2. Choose instance $\mathsf{inst}_0 = (z_0, \mathsf{count}_0, \hbar_0)$ for arbitrary values $\mathsf{count}_0, \hbar_0$. Choose acc_0 arbitrarily. Let $\mathsf{inst}_1 = (z_1, 1, \hbar_1)$, $\boldsymbol{w}$-$\mathsf{inst}_1 = (\mathsf{acc}_0, \mathsf{acc}_0, \mathsf{inst}_0, w_1, \pi_0)$ for arbitrary values $\hbar_1, \mathsf{acc}_0, \pi_0$. (We can choose some values arbitrarily as they aren't constrained in F' when $\mathsf{count} = 1$.)

[3] In an updated version of the paper we will give more details on how to efficiently implement F' as a PROTOGALAXY relation.

3. Let acc_1 be a randomly chosen satisfiable accumulator. Namely, $\mathsf{acc}_1 = (\mathsf{cm}(\mathbf{w}\text{-}\mathsf{acc}_1), \boldsymbol{\beta}, e)$ where $\mathbf{w}\text{-}\mathsf{acc}_1$ and $\boldsymbol{\beta}$ are chosen randomly[4], and e is set to $e = \sum_{i\in[n]} \mathsf{pow}_i(\boldsymbol{\beta}) f_i(\mathbf{w}\text{-}\mathsf{acc}_1)$.

4. For each $2 \leq i \leq n$, compute

   (a) $(\mathsf{acc}_i, \mathbf{w}\text{-}\mathsf{acc}_i, \pi_i) = \mathbf{P}^{\mathrm{PG}}_{\mathsf{cm},n,f}(\mathsf{acc}_{i-1}, \mathbf{w}\text{-}\mathsf{acc}_{i-1}, \mathsf{inst}_{i-1}, \boldsymbol{w}\text{-}\mathsf{inst}_{i-1})$

   (b) $\mathsf{inst}_i = (z_i, i, \mathcal{H}(\mathsf{acc}_i))$,

   (c) $\boldsymbol{w}\text{-}\mathsf{inst}_i = (\mathsf{acc}_i, \mathsf{acc}_{i-1}, \mathsf{inst}_{i-1}, w_i, \pi_i)$

5. $(\mathsf{acc}, \mathbf{w}\text{-}\mathsf{acc}, \pi^*) = \mathbf{P}^{\mathrm{PG}}_{\mathsf{cm},n,f}(\mathsf{acc}_n, \mathbf{w}\text{-}\mathsf{acc}_n, \mathsf{inst}_n, \boldsymbol{w}\text{-}\mathsf{inst}_n)$.

6. Let $\pi_{\mathsf{fin}} = \mathbf{P}_{\mathsf{fin}}(x, w)$ where

   - $x = (\mathsf{acc}, \mathsf{V}, C, z_{final})$
   - $w = (\mathbf{w}\text{-}\mathsf{acc}, \mathsf{acc}_n, \mathsf{inst}_n, \pi^*)$

7. Output $\pi = (\mathsf{acc}, \pi_{\mathsf{fin}})$.

$\underline{\mathbf{V}(x, \pi)}$:

1. Parse x as $(z_{final}, C, \mathsf{V})$, π as $(\mathsf{acc}, \pi_{\mathsf{fin}})$.

2. If $\mathscr{f}(z_{final}, \mathsf{V}) = \mathsf{reject}$ output reject.

3. Let $x := (\mathsf{acc}, z_{final}, C, \mathsf{V})$.

4. Return $\mathbf{V}_{\mathsf{fin}}(x, \pi_{\mathsf{fin}})$.

Theorem 8.1. *Let $\mathcal{R}_{F,\mathscr{f}}$ be a trivial RCG relation. Assume that $\mathcal{H}$ is collision resistant, and that the assumptions in Theorem 6.3 hold. Then $(\mathbf{P}, \mathbf{V})$ is a space-efficient zk-SNARK for $\mathcal{R}_{F,\mathscr{f}}$.*

Proof. The main thing to prove is knowledge soundness. Let $\mathcal{A}$ be a recursive algebraic adversary.

We define the following extractor algorithm $\mathbf{E}$:

1. Given (x, π), use the SNARK extractor to output $w = (\mathbf{w}\text{-}\mathsf{acc}, \mathsf{acc}^*, \mathsf{inst}^*, \pi^*)$.

2. Let $n := \mathsf{inst}^*.\mathsf{count}$. Denote $\mathsf{acc}_n := \mathsf{acc}^*, \mathsf{inst}_n := \mathsf{inst}^*$.

3. If $(x, w) \notin \mathcal{R}_{\mathsf{fin}}$ output *empty* and abort. Otherwise $\mathsf{acc}_n, \mathsf{inst}_n \in K$ and let $\mathbf{w}\text{-}\mathsf{acc}_n, \boldsymbol{w}\text{-}\mathsf{inst}_n$ be the elements output by $\mathcal{A}$ such that $\mathsf{cm}(\mathbf{w}\text{-}\mathsf{acc}_n) = \mathsf{acc}_n$ and $\mathsf{cm}(\boldsymbol{w}\text{-}\mathsf{inst}_n) = \mathsf{inst}_n$.

4. For $i = n-1, \ldots, 1$:

[4] We only need $\mathsf{cm}(\mathbf{w}\text{-}\mathsf{acc}_1)$ to be uniformly distributed over the image of cm. According to cm's structure, it may suffice to choose only a small subset of $\mathbf{w}\text{-}\mathsf{acc}_1$'s coordinates randomly, and set the rest to zero.

(a) If $(\mathsf{inst}_{i+1}, \boldsymbol{w}\text{-}\mathsf{inst}_{i+1}) \notin \mathcal{R}$, output empty and abort. Otherwise, as $(\mathsf{inst}_{i+1}, \boldsymbol{w}\text{-}\mathsf{inst}_{i+1}) \in \mathcal{R}$ implies $\mathsf{acc}_i, \mathsf{inst}_i \in K$, let $\mathbf{w}\text{-}\mathsf{acc}_i, \boldsymbol{w}\text{-}\mathsf{inst}_i$ be the elements output by $\mathcal{A}$ such that $\mathsf{acc}_i = \mathsf{cm}(\mathbf{w}\text{-}\mathsf{acc}_i), \mathsf{inst}_i = \mathsf{cm}(\boldsymbol{w}\text{-}\mathsf{inst}_i)$.

(b) Parse inst_i as $(z_i, \mathsf{count}_i, \hbar_i)$. Parse $\boldsymbol{w}\text{-}\mathsf{inst}_i$ as $(\mathsf{acc}'_i, \mathsf{acc}_{i-1}, \mathsf{inst}_{i-1}, w_i, \pi_i)$. Note that through this parsing we have in particular defined z_i, w_i and inst_{i-1}.

5. Define $z_0 := \mathsf{inst}_0.z$.

6. Output $w := (n, (z_0, \ldots, z_n), (w_1, \ldots, w_n))$.

Let D be the event that $\mathcal{A}$ outputs (x, π) such that $\mathbf{V}(x, \pi) = \mathsf{accept}$. If D has probability $\mathsf{negl}(\lambda)$ we are done. Otherwise, in order to prove knowledge soundness, we need to show that

$$\Pr[(x, w) \notin \mathcal{R}_{F,\ell} \mid D] = \mathsf{negl}(\lambda).$$

Assume from now we are in the space conditioned on D. Let E_0 be the event $(\boldsymbol{x}, \boldsymbol{w}) \notin \mathcal{R}_{\mathsf{fin}}$. For $i \in [n]$, let E_i be the union of the following events:

1. $(\mathsf{inst}_i, \boldsymbol{w}\text{-}\mathsf{inst}_i) \notin \mathcal{R}$.

2. $(\mathsf{acc}_i, \mathbf{w}\text{-}\mathsf{acc}_i) \notin \mathcal{R}$.

3. $\mathcal{H}(\mathsf{acc}_i) \neq \mathsf{inst}_i.\mathcal{H}$.

We first argue that if $E_0, \ldots, E_n$ *don't* occur, $(x, w) \in \mathcal{R}_{F,\ell}$. Denote $x = (z_{final}, C, \mathsf{V})$ and $\boldsymbol{x} = (\mathsf{acc}, x)$. Examining the definition of $\mathcal{R}_{F,\ell}$, the statement $(x, w) \in \mathcal{R}_{F,\ell}$ is equivalent to the following conditions.

1. $\ell(z_{final}, \mathsf{V}) = \mathsf{true}$: $\mathbf{V}$ checks this directly so we know it holds.

2. $\mathsf{inst}_n.z = z_{final}$ and $n \leq C$: Follows from $(\boldsymbol{x}, \boldsymbol{w}) \in \mathcal{R}_{\mathsf{fin}}$.

3. For all $i \in [n]$ $F(z_{i-1}, w_i, z_i) = \mathsf{accept}$: Follows directly from $(\mathsf{inst}_i, \boldsymbol{w}\text{-}\mathsf{inst}_i) \in \mathcal{R}$ for each $i \in [n]$.

4. $z_0.\mathsf{init} = \mathsf{true}$: For this we first prove by backwards induction on $i = n, n-1 \ldots, 1$ that $\mathsf{inst}_i.\mathsf{count} = i$. By our definition of inst_n this holds for $i = n$. For the induction step, assume it holds for i. Since $(\mathsf{inst}_i, \boldsymbol{w}\text{-}\mathsf{inst}_i) \in \mathcal{R}$ we have $\mathsf{inst}_{i-1}.\mathsf{count} = \boldsymbol{w}\text{-}\mathsf{inst}_{i-1}.\mathsf{inst}.\mathsf{count} = \mathsf{inst}_i.\mathsf{count} - 1 = i - 1$.

   In particular, $\mathsf{inst}_1.\mathsf{count} = 1$ which implies when $(\mathsf{inst}_1, \boldsymbol{w}\text{-}\mathsf{inst}_1) \in \mathcal{R}$ that $z_0.\mathsf{init} = \mathsf{inst}_0.z.\mathsf{init} = \mathsf{true}$.

It is left to show that the probability of $E_0 \cup E_1 \ldots \cup E_n$ is $\mathsf{negl}(\lambda)$. From the knowledge-soundness of the zk-SNARK for $\mathcal{R}_{\mathsf{fin}}$ we have $\Pr[E_0] = \mathsf{negl}(\lambda)$. (Recall we are in the event D and assuming its probability is non-negligible.)

For this purpose, we prove by backwards induction on i, for each $i \in [n]$, that the probability that E_i occurs given $E_0, E_{i+1}, \ldots, E_n$ *didn't* is $\mathsf{negl}(\lambda)$.

For $i = n$, assume E_0 didn't occur, hence $(x, w) \in \mathcal{R}_{\mathsf{fin}}$. In this case $(\mathsf{acc}, \mathbf{w}\text{-}\mathsf{acc}) \in \mathcal{R}^{\mathsf{rand}}$ and $\mathbf{V}_{PG}(\mathsf{inst}_n, \mathsf{acc}_n, \pi^*, \mathsf{acc}) = \mathsf{accept}$. Define $\mathcal{A}_n$ to be the algorithm that executes $\mathbf{E}$ and outputs $\mathsf{acc}_n, \mathsf{inst}_n, \mathsf{acc}, \mathbf{w}\text{-}\mathsf{acc}_n, \boldsymbol{w}\text{-}\mathsf{inst}_n, \mathbf{w}\text{-}\mathsf{acc}, \pi^*$. From the knowledge soundness of PROTOGALAXY applied to $\mathcal{A}_n$ we have that the probability conditioned on $\neg E_0$ that E_n occurred is $\mathsf{negl}(\lambda)$.

Now assume the induction hypothesis for some $i > 1$. We have by the hypothesis that the event $D_i := \neg(E_0 \cup E_i \ldots \cup E_n)$ has probability $1 - \mathsf{negl}(\lambda)$. Assume that we are in D_i. From $(\mathsf{inst}_i, \boldsymbol{w}\text{-}\mathsf{inst}_i) \in \mathcal{R}$ we have that $\mathsf{inst}_i.\hbar = \mathcal{H}(\mathsf{acc}'_i) = \mathcal{H}(\mathsf{acc}_i)$. From the collision resistance of $\mathcal{H}$, we thus have that the probability that $\mathsf{acc}_i \neq \mathsf{acc}_i$ is $\mathsf{negl}(\lambda)$. If $\mathsf{acc}_i = \mathsf{acc}'_i$ we have from $(\mathsf{inst}_i, \boldsymbol{w}\text{-}\mathsf{inst}_i) \in \mathcal{R}$ that $\mathbf{V}(\mathsf{acc}_{i-1}, \mathsf{inst}_{i-1}, \pi_i, \mathsf{acc}_i) = \mathsf{accept}$. Define $\mathcal{A}_i$ to be the algorithm that executes $\mathbf{E}$ and outputs $\mathsf{acc}_{i-1}, \mathsf{inst}_{i-1}, \mathsf{acc}_{i-1}, \mathbf{w}\text{-}\mathsf{acc}_{i-1}$, $\boldsymbol{w}\text{-}\mathsf{inst}_{i-1}, \mathbf{w}\text{-}\mathsf{acc}_i, \pi_i$. From the knowledge soundness of PROTOGALAXY applied to $\mathcal{A}_i$ we have that the probability that $(\mathsf{inst}_{i-1}, \boldsymbol{w}\text{-}\mathsf{inst}_{i-1}) \notin \mathcal{R}$ or $(\mathsf{acc}_{i-1}, \mathbf{w}\text{-}\mathsf{acc}_{i-1}) \notin \mathcal{R}^{\mathsf{rand}}$ given D_i is $\mathsf{negl}(\lambda)$. Noting that $(\mathsf{inst}_i, \boldsymbol{w}\text{-}\mathsf{inst}_i) \in \mathcal{R}$ also implies $\mathcal{H}(\mathsf{acc}_{i-1}) = \mathsf{inst}_{i-1}.\hbar$, $\Pr[E_{i-1} \mid D_i] = \mathsf{negl}(\lambda)$; which is the desired inductive statement.

□

8.4 Constructing proofs for $\mathcal{R}_{\text{exec}}$

Going through the reductions of previous sections, we get the following zk-SNARK for the relation $\mathcal{R}_{\text{exec}}$ from Section 3.3 describing valid executions.

$\underline{\mathbf{P}(\mathrm{x}_{\text{exec}}, \mathrm{w}_{\text{exec}})}$:

1. Let $\mathrm{x}_{\text{exec}} = (\mathbf{r}, C, \mathsf{V}), z_{final} = (g_{empty}, \mathbf{r}, \mathsf{false})$ and $x = (z_{final}, C, \mathsf{V})$. Let φ be the map in Lemma 4.1, and compute $w = \varphi(\mathrm{w}_{\text{exec}})$.
2. Let $w = (n, z, w, s)$. As in Lemma 5.2 define for each $i \in [n]$ $m_i \in \mathbb{F}^k$ such that $m_{i,j}$ is the number of times a record is read when $s_{i,j}$ is an add operation, and zero otherwise.
3. Compute $h_n = \mathsf{cm}'(s, m)$, where cm' is as in Lemma 5.2. Compute $(\alpha, \beta, \varepsilon) = (\mathcal{H}(\hbar, \mathsf{V}, 1), \mathcal{H}(\hbar, \mathsf{V}, 2), \mathcal{H}(\hbar, \mathsf{V}, 3))$.
4. Define $z^*_{final} = (z_{final}, h_n, \alpha, \beta, \varepsilon), x^* = (z^*_{final}, C, \mathsf{V})$. Let $w^* = \varphi^*(w)$ where φ^* is the mapping from Lemma 5.2.
5. Compute the proof π^* for $(x^*, w^*) \in \mathcal{R}_{F^*, \mathscr{F}^*}$ according to $\mathbf{P}$ in Section 8.3.
6. Output $\pi = ((h_n, \alpha, \beta, \varepsilon), \pi)$.

$\underline{\mathbf{V}(\mathrm{x}_{\text{exec}}, \pi)}$:

1. Parse π as $(\hbar, \alpha, \beta, \varepsilon, \pi^*)$. Parse x_{exec} as $(z_{final}, C, \mathsf{V})$. Define $x = ((z_{final}, \hbar, \alpha, \beta, \varepsilon), C, \mathsf{V})$.
2. Let $\mathbf{V}'$ be $\mathbf{V}$ from Section 8.3. Return $\mathbf{V}'(x, \pi^*)$.

9 General Final predicate

We sketch how to deal with a *general* final predicate $\mathscr{f}$, rather than the one described so far for checking record operations. A more detailed explanation will appear in an updated version of the paper.

The basic idea is that a general final predicate reduces to the concatenation predicate. Namely, suppose we can prove an element $\mathscr{s}$ is the commitment to the concatenation of the $\{s_i\}$ from all n iterations. Then we can modify the final zk-SNARK in Section 8 to also check that $\mathscr{f}(z_{final}, s_1, \ldots, s_n, \mathsf{V}) = \mathsf{accept}$.

For this purpose, instead of the function F^* in section 5 we add PROTOGALAXY constraints to the transition function F to obtain at iteration i a vector S_i that is the concatenation of S_{i-1} with s_i.

9.1 Efficient Proofs of Concatenation with PROTOgalaxy

Given a sequence consisting of vectors $\{s_i\}_{i\in[n]}$ of field elements of length k and a bound C on n, we can define a system of linear protogalaxy constraints that a concatenation was correctly carried out. We highlight that it is important the constraint system does not depend on the current iteration i, or actual sequence length n, only on the bound C. The idea is that if we append the new elements *at the beginning* of the vector, the concatenation constraints can be written as linear constraints that indeed only depend on k, C but not i or n.

Fix vector $b \in \mathbb{F}^k$ and vectors $a, a' \in F^{kC}$. For each $j \in [kC]$ we define the constraint $f_j(a, b, a')$ as follows.

$$f_j(a, b, a') := \begin{cases} a'_j - b_j & i \leq k \\ a'_j - a_{j-k} & j > k \end{cases}.$$

In words, the constraints $\{f_j\}$ enforce that a' consists of the k elements of b followed by the first $(C-1)k$ elements of a. Additionally, we will constrain (using the init variable) that S_0 is the zero vector of length kC. In the i'th iteration, our transition function F will check for each $j \in [kC]$ the constraint $f_j(S_i, s_i, S_{i+1}) = 0$.

One can now show inductively that when starting from the zero vector, under the assumptions that $n \leq C$ and $|s_i| \leq k$ for each $i \in [n]$, we end up with $S_n = (s_1 \| \ldots \| s_n)$ possibly padded with zeroes.

Prover efficiency If the stack is folded in order, then even in the new randomized instance, the zero padding is preserved. That is, higher order elements above ki are still zero after folding. So the prover only needs to perform one field multiplication per stack element to compute the folded witness. These constraints are also linear so they do not contribute to any of the error terms in the $G(X)$ polynomial.

For all $j > k(i+1)$ notice that $f_j(S_i, s_i, S_{i+1}) = 0$ identically, so they do not even contribute to the cost of computing the $F(X)$ polynomial. Assuming all of these constraints are organized adjacently in the highest order indices of the overall protogalaxy constraint system, there is a straightforward modification of the $F(X)$ computation which incurs

no cost for constraints that are identically zero. In brief, if an entire subtree consists of leaves that are zero, then the prover can avoid any computation for that subtree. Therefore, the total prover computation depends only on the size of the vector S_{i+1}.

References

[BC23] B. Bünz and B. Chen. Protostar: Generic efficient accumulation/folding for special sound protocols. *IACR Cryptol. ePrint Arch.*, page 620, 2023.

[BCCT12] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for SNARKs and proof-carrying data. Cryptology ePrint Archive, Paper 2012/095, 2012. `https://eprint.iacr.org/2012/095`.

[BCL+21] B. Bünz, A. Chiesa, W. Lin, P. Mishra, and N. Spooner. Proof-carrying data without succinct arguments. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part I*, volume 12825 of *Lecture Notes in Computer Science*, pages 681–710. Springer, 2021.

[CT10] A. Chiesa and E. Tromer. Proof-carrying data and hearsay arguments from signature cards. In *Innovations in Computer Science - ICS 2010, Tsinghua University, Beijing, China, January 5-7, 2010. Proceedings*, pages 310–331. Tsinghua University Press, 2010.

[Eag22] Liam Eagen. Bulletproofs++. *IACR Cryptol. ePrint Arch.*, page 510, 2022.

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

[FKL18] G. Fuchsbauer, E. Kiltz, and J. Loss. The algebraic group model and its applications. In *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part II*, pages 33–62, 2018.

[Gro24] J. Groth. Memory checking in folding-based zkvms - jens groth (nexus), 2024. `https://www.youtube.com/watch?v=xnzjC_9vUzs`.

[GWC19] A. Gabizon, Z. J. Williamson, and O. Ciobotaru. PLONK: permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. *IACR Cryptology ePrint Archive*, 2019:953, 2019.

[Hab22] U. Haböck. Multivariate lookups based on logarithmic derivatives. *IACR Cryptol. ePrint Arch.*, page 1530, 2022.

[KST21] A. Kothapalli, S. T. V. Setty, and I. Tzialla. Nova: Recursive zero-knowledge arguments from folding schemes. *IACR Cryptol. ePrint Arch.*, page 370, 2021.

[KZG10] A. Kate, G. M. Zaverucha, and I. Goldberg. Constant-size commitments to polynomials and their applications. pages 177–194, 2010.

[LS24] Hyeonbum Lee and Jae Hong Seo. On the security of nova recursive proof system. Cryptology ePrint Archive, Paper 2024/232, 2024. `https://eprint.iacr.org/2024/232`.

[NDC+24] Wilson Nguyen, Trisha Datta, Binyi Chen, Nirvan Tyagi, and Dan Boneh. Mangrove: A scalable framework for folding-based SNARKs. Cryptology ePrint Archive, Paper 2024/416, 2024. `https://eprint.iacr.org/2024/416`.

[Sou23] L. Soukhanov. Folding endgame, 2023. `https://zkresear.ch/t/folding-endgame/106`.

[Val08] P. 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*, volume 4948 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2008.