stackproofs: Private proofs of stack and contract execution using PROTO**GALAXY**

July 22, 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[?] (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 desigining a proof system for a private smart contract system like Aztec.

1 Introduction

Incrementally Verifiable Computation (IVC) [?] and its generalization to Proof Carrying Data (PCD) [?] are useful tools for constructing space-efficient SNARK provers. 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. 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, we can't predict specifically 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 "authorize" is not tied to one 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 While running, a contract can read, add or delete notes belonging to it. We can thus think of the notes as global variables shared between the different functions.

Let us think of all functions in this system as having multiple arguments, and returning 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.

However, this doesn't yet deal with the notes. During native execution, note operations happened at a certain order. We can think of these operations as having timestamps incremented by one with each operation. We need to check, for example, that if a note was read in a certain timestamp, it was indeed created in an earlier timestamp. We can have the note operations - $\{add, read, delete\}$ - performed by a function be included in the public inputs of its circuit. The issue is, what if A is reading a note that was added in the internal call to B? Checking the existance of an add operation with smaller timestamp than that of the read requires a constraint between the inputs of both circuits.

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 allow 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? The point is that in our usecase 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

mangrove, nexus, Jens's talk.

2 Preliminaries

2.1 Terminology and Conventions

We assume our field $\mathbb{F}$ is of prime order. We denote by $\mathbb{F}_{<d}[X]$ the set of univariate polynomials over $\mathbb{F}$ of degree smaller than d. We assume all algorithms described receive as an implicit parameter the security parameter λ. Similarly, we assume all integer parameters in the paper are implicitly functions of λ.

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 prime field of super-polynomial 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, i.e. write $\mathbb{F}$ instead of $\mathbb{F}(\lambda)$. 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 Relations of the app functions

Define a relation including the selectors Fixed polynomial $f(x_1, \ldots, x_S)$ $S = n + d + \ell$ Relation $\mathcal{R}_{app}$ of $\mathsf{cm1}, \ldots, \mathsf{cm}_S$

3 The execution model:

We present a formal framework that will be convenient for our proof system of what it means to prove an execution where functions can call each other, and there is global state. For this, we first introduce record operations which is our specific notion of operating on a global state.

3.1 Record operations

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

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

Here $v \in \mathbb{F}$ is a note value and $c, vc \in [n]$ are counters. c is interpreted as the counter of the current operation, and vc 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 n is *consistent* if

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

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

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

3.2 The plonkish relation

Now we introduce a relation describing the individual function executions tailored to make it convenient in the next section to discuss an execution of a sequence functions calling each other. Some choices of constants - like the maximal number of inner calls being two, are abitrary. We require the instance to adhere to a form containing both the record operations and the details of the inner calls (though they will be interpreted as such only in the next section when we discuss valid executions).

We fix a polynomial $G : \mathbb{F}^8 \to \mathbb{F}$ that is an implicit parameter in the following definition of the 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 *exeuction tree of length* n we mean a tree T with n vertices of maximal degree two, 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
   - $\mathsf{f} \in \mathrm{F}$.
   - 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

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

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

An RCG relation is defined by a pair of functions (F, f).
We call $F(Z, W, S, Z^*) \to \{\mathsf{accept}, \mathsf{reject}\}$ the *transition predicate.*

We informally think of

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

Let D_1, D_2 be the domains of S, V respectively. f is a function $\mathsf{f} : D_1^* \times D_2 \to \{\mathsf{accept}, \mathsf{reject}\}$ called the *final predicate.*

The relation $\mathcal{R} = \mathcal{R}_{F,\mathsf{f}}$ consists of pairs (x, ω) such that $\mathsf{x} = (z_{\mathrm{final}}, C, \mathsf{V}), \omega = (n, z = (z_0, \ldots, z_n), w = (w_1 \ldots, w_n), s = (s_1, \ldots, s_n))$ such that

- $z_0.\mathsf{init} = \mathsf{true}$.
- $z_n = z_{\mathrm{final}}$.
- $n \leq C$.
- For each $i \in [n]$, $F(z_{i-1}, w_i, s_i, z_i) = \mathsf{accept}$.
- $\mathsf{f}(s_1, \ldots, s_n, \mathsf{V}) = \mathsf{accept}$.

we say a zk-SNARK for $\mathcal{R}$ is *space-efficient* if given s and streaming access to z and w $\mathbf{P}$ requires space $O(|F| + |s| + \log n)$.

4.1 Valid executions as RCG

We now reduce valid executions as defined in Section 3.3 to RCG's.

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. Denoting $g[0] = (\mathsf{f}, \mathsf{args})$, we have $\mathsf{f} = \mathbf{x}.\mathsf{f}$ and $\mathsf{args} = \mathbf{x}.\mathsf{args}$.
3. Setting $\mathfrak{x} = (\mathbf{x}, S)$, we have $(\mathfrak{x}, \mathfrak{w}) \in \mathcal{R}_{app}$.
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 $i \in [\mathfrak{x}.\mathsf{c}]$.

We define the function f on input $(s_1, \ldots, s_n, \mathsf{V})$ to output accept if and only if each s_i is a well-formed set of record operations, and when defining $\mathcal{O}$ as the multi-set union of $s_1, \ldots, s_n$ it has output V.

In the lemma below, we denote by g_{empty} the empty stack.

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 $z_{\text{final}} = (g_{\text{empty}}, \mathbf{r}, \mathsf{false})$. 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 $((z_{\text{final}}, C, \mathsf{V}), \phi(\mathsf{T})) \in \mathcal{R}_{F,\mathsf{f}}$.

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 $g[0]$ and adding $(\mathfrak{x}_i.\mathsf{f}_j, \mathsf{args}_j)$ for $j \in [\mathfrak{x}_i.\mathsf{c}]$.

Now, define $z_0 = (g_0, \mathbf{r}, \mathsf{true})$ and for each $i \in [n]$, $z_i = (g_i, \mathbf{r}, \mathsf{false})$.

We now 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 $\phi(\mathsf{T}) = (n, z, w, s)$. Given this definition of ϕ the statement of the lemma is straightforward to check. $\square$

5 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 the following definition.

Definition 5.1. *Fix relations $\mathcal{R}$ and $\mathcal{R}_{\text{acc}}$. An $(\mathcal{R} \mapsto \mathcal{R}_{\text{acc}})$*-accumulation/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* $(\Phi, \omega) \in \mathcal{R}_{\text{acc}}, (\phi', \omega') \in \mathcal{R}$, *and* $\mathbf{P}(\Phi, \phi'; \omega, \omega') = (\pi, \Phi^*, \omega^*)$ *then with probability one* $(\Phi^*, \omega^*) \in \mathcal{R}_{\text{acc}}$ *and* $\mathbf{V}(\Phi, \phi', \Phi^*, \pi) = \mathsf{accept}$.
   (b) ***Knowledge soundness in the Algebraic Group Model:***
   For any efficient $\mathcal{A}$ *the probability of the following event is* $\mathsf{negl}(\lambda)$*:* $\mathcal{A}$ *outputs* $(\phi, \tau), \phi', (\phi^*, \tau^*), \omega, \omega', \omega^*$ *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. $((\phi^*, \tau^*), \omega^*) \in \mathcal{R}_{\text{acc}}$,
      iv. $((\phi, \tau), \omega) \notin \mathcal{R}_{\text{acc}}$ *or* $(\phi', \omega') \notin \mathcal{R}$.

5.1 Relations for folding schemes

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

- *Constraint function* $f : \mathbb{F}^M \to \mathbb{F}^n$ which is a vector of n degree d polynomials,
- *Instance predicate* $\mathsf{f} : K \to \{\mathsf{accept}, \mathsf{reject}\}$,
- *Commitment function* $\mathsf{cm} : \mathbb{F}^M \to K$ 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

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

The relation $\mathcal{R}^{\mathsf{rand}}$: For brevity, let $\mathcal{R} = \mathcal{R}_{f,\mathsf{f},\mathsf{cm}}$. As in [?], 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 $S \subset \{0, \dots, t-1\}$ be the set such that $i - 1 = \sum_{j \in [S]} 2^j$. We define the t-variate polynomial pow_i as

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

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

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.)

5.2 The Protogalaxy scheme

Deviating from [?], we explicitly present Protogalaxy as a *non-interactive* folding scheme, for the special case of folding $k = 1$ instances.

We define $Z(X)$

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

Compute the polynomial

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

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

Denote the non-constant coefficients of F by $a := (F_1, \ldots, F_t)$.

Compute $\alpha = \mathcal{H}(\delta, a)$

Compute $\boldsymbol{\beta}^* \in \mathbb{F}^t$ where $\boldsymbol{\beta}^*_i := \boldsymbol{\beta}_i + \alpha \cdot \boldsymbol{\delta}_i$.

Define the polynomial $G(X)$ as

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

Compute the polynomial $K(X)$ such that

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

Let $b := (K_0, \ldots, K_{d-2})$ be the coefficients of $K(X)$.

Compute $\gamma = \mathcal{H}(\alpha, b)$.

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

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

1. Compute $\alpha, \beta, \delta, \gamma$ as in the prover algorithm given Φ, ϕ_1, a, b.
2. Check that Φ^* is as computed 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 form [**?**].

Definition 5.2. *Fix* $\mathsf{cm} : \mathbb{F}^M \to K$*, hash function* $\mathcal{H}$*, parameter* d*. The Zero-Testing Assumption (ZTA) for* $(\mathcal{H}, \mathsf{cm}, d)$ *states the following.*

For any efficient $\mathcal{A}$ *and efficiently computable function* D *into* $\mathbb{F}_{<d}[X]$*, the probability of* $\mathcal{A}$ *outputting* x, τ *such that, when setting*

- $f(X) = D(x, \tau)$
- $z = \mathcal{H}(\mathsf{cm}(x), \tau)$

we have that f *is not the zero polynomial but* $f(z) = 0$*, is* $\mathsf{negl}(\lambda)$*.*

We extend cm to work

Theorem 5.3. *Set* $d' := \max\{\log n, d\}$*. Denote by* $\mathsf{cm}^{\oplus 2}$ *the function* $\mathsf{cm}^{\oplus 2}(\omega_1, \omega_2) := (\mathsf{cm}(\omega_1), \mathsf{cm}(\omega_2))$*. Assume that* cm *is collision resistant and the ZTA holds for* $(\mathcal{H}, \mathsf{cm}^{\oplus 2}, d')$ *Then* PROTO**GALAXY** *has knowledge soundness in the Algebraic Group Model.*

Proof. Let E be the event that $\mathcal{A}$ has output $\Phi = (\phi, \boldsymbol{\beta}, e), \phi_1, \Phi^* = (\phi^*, \boldsymbol{\beta}^*, e^*), \omega, \omega_1, \omega^*, \pi$ such that

1. $\mathsf{cm}(\omega) = \phi, \mathsf{cm}(\omega_1) = \phi_1, \mathsf{cm}(\omega^*) = \phi^*$.
2. $\mathbf{V}(\phi, \phi_1, \phi^*, \pi) = \mathsf{accept}$
3. $(\Phi^*, \omega^*) \in \mathcal{R}^{\mathsf{rand}}$
4. $(\phi_0, \omega) \notin \mathcal{R}^{\mathsf{rand}}$ or $(\phi_1, \omega_1) \notin \mathcal{R}$.

According to Definition 5.1 knowledge soundness is equivalent to E having probability $\mathsf{negl}(\lambda)$ for any $\mathcal{A}$. We construct an efficient $\mathcal{A}'$ that when E occurs outputs either a collision of cm or an exception to ZTA. Hence, by the theorem assumption E is contained in two events having 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

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

Thus, if $\omega^* \neq \gamma\omega + (1-\gamma)\omega'$, $\mathcal{A}'$ can output $(\omega^*, \gamma\omega + (1-\gamma)\omega')$ as a collision of cm. Now assume that $\omega^* = \gamma\omega + (1-\gamma)\omega'$. Suppose $\pi = (a, b)$, with $a = (a_1, \dots, a_t), b = (b_0, \dots, b_{d-2})$. 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 $\alpha, \beta, \delta, \gamma, \boldsymbol{\beta}, \boldsymbol{\delta}$ be computed as in the prover description given a, b. Define the polynomials

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

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

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

$$F'(\alpha)\gamma + Z(\gamma)K'(\gamma) = e^* = \sum_{i\in[n]} \mathsf{pow}_i(\boldsymbol{\beta}^*) f(\omega^*).$$

This precisely means that $G'(\gamma) = 0$.

If $(\phi_1, \omega_1) \notin \mathcal{R}$ then (the honest) G will have a $(1-X)\beta^* f(w_1)$ factor that can't be canceled out by $Z(X)K(X)$ so G' can't be the zero polynomial (see PG paper analysis if not clear).

And hence taking $d = (w, w_1, \beta, \alpha, \delta, K), D(d) = G'$ we get a contradiction to ZTA. (A little care needs to be taken to define h and C so that we really get $h(C(w, w_1, \beta, \alpha, \delta, K)) = \gamma$).

Now assume $((\phi, \beta, e), w) \notin R_{rand}$. And that $G' \equiv 0$ - otherwise we can contradict ZTA via G' as above.

Thus, if we write $G'(X) = c_1 X + c_2(1-X) + Z(X)K'(X)$ for some constants c_1, c_2 and polynomial K' we must have $c_1, c_2, K' = 0$. Looking at the equation of G' we have $c_1 = F(\alpha) - \beta^* f(w)$.

Note that $c_1 = F'(\alpha)$, so $F'(\alpha) = 0$.

We also have $F'(0) = e - \beta f(w) \neq 0$, and thus F' is not the zero polynomial.

Hence, in this case $d = (w, \beta, e, w_1, \delta)$ $D(d) = F'$ is a counterexample to ZTA.

□

5.3 Proving record ops via log-derivative

6 On protogalaxy

6.1 Introspective constraints

Constraints f_i on ω, simply low degree polynomials, but have the ability to refer to components of $\mathsf{cm}(m_j)$ Function F should have "introspection" ability to look at commitments.

commitment function will output two representations of $\mathsf{cm}(w)$ - in $\mathbb{G}$ and in $\mathbb{F}$ and

or PI will include $\mathbb{F}$ representation. which will then be part of w. $\mathbf{V}$ will check representations match.

6.2 Proving $\textsc{Proto}$**galaxy** under zero-testing assumption

7 The Algebraic Group Model with recursive extraction

As in [FKL18], we assume when $\mathcal{A}$ outputs $a \in \mathbb{G}$ it outputs a vector $c \in \mathbb{F}^n$ with $< c, \mathsf{srs} >= a$. We fix some mapping $G : \mathbb{F}^4 \to \mathbb{G} \cup \{*\}$. $*$ means the input doesn't correspond to $\mathbb{G}$ element. We assume that this is the representation $\mathcal{A}$ uses for elements of $\mathbb{G}$.

For our security proof, we require a notion of "recursive extraction" used by [?]: Specifically, we assume that if c output by $\mathcal{A}$ along with a, thought of as $c \in (\mathbb{F}^4)^{n/4}$, contains an element c_i with $G(c_i) = a \in \mathbb{G}$, then $\mathcal{A}$ outputs $c' \in \mathbb{F}^n$ with $< c', \mathsf{srs} >= c$. The same holds for c'. Thus $\mathcal{A}$ must continue outputting representations until reaching one where no element correponds to $a \in \mathbb{G}$.

8 Transforming F into F'

Describing F^* **public input:** z - output for F G - global state for F $count$ - counter of IVC step h - supposed hash of accumulator **private input:** acc - current accumulator instance acc_{prev} - previous accumulator instance $inst$ - instance (of F') to be accumulated. w - private input for F π - proof for protogalaxy verifier

Set $X := (z, G, count, h)$, $W := (acc, acc_{prev}, inst, w, \pi)$

$F'(X, W) =$ accept if and only if:

1. $hash(acc) = h$.
2. $V_{PG}(acc_{prev}, inst, \pi, acc) =$ acc

3. $inst.h = hash(acc_{prev})$.
4. If $count > 0$:
    (a) $inst.G = G$.
    (b) $inst.count = count - 1$.
    (c) $F(false, G, inst.z, w, z) = \text{acc}$
5. If $count = 0$:
    (a) $F(true, G, inst.z, w, z) = \text{acc}$

Acknowledgements

References

[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.