One-Shot Native Proofs of Non-Native Operations in Incrementally Verifiable Computations

Tohru Kohrita[1], Patrick Towa[2] and Zachary J. Williamson[2]

[1] Nil Foundation
[2] Aztec Labs

Abstract. Proving non-native operations is still a bottleneck in existing incrementally verifiable computations. Prior attempts to solve this issue either simply improve the efficiency of proofs of non-native operations or require folding instances in each curve of a cycle. This paper shows how to avoid altogether in-circuit proofs of non-native operations in the incremental steps, and only record them in some auxiliary proof information. These operations are proved *natively* at the end of the computation, at the cost of only a small constant number (four or five) of non-native field multiplications to go from a non-native operation record to a native one. To formalise the security guarantees of the scheme, the paper introduces the concept of incrementally verifiable computation with auxiliary proof information, a relaxation of the standard notion of incrementally verifiable computation. The knowledge-soundness now guarantees the correctness of a computation only if the piece of information attached to a proof is valid. This new primitive is thus only to be used if there is an efficient mechanism to verify the validity of that information. This relaxation is exactly what enables a construction which does not require in-circuit proofs of non-native operations during the incremental part of the computation. Instantiated in the Plonk arithmetisation, the construction leads to savings in circuit-gate count (compared to standard folding-based constructions) of at least one order of magnitude, and that can go up to a factor of 50.

1 Introduction

1.1 Context

The modern approach to proving the correctness of a computation is to do so incrementally. The computation is first decomposed into sequential steps. Then, to prove the correctness of the overall computation, the idea is to sequentially prove the execution of a single step starting from the output of the previous step, assuming it to have already been proved correct. This is what is known as Incrementally Verifiable Computation (IVC) [26]. The main advantage of this approach is that the computation and memory requirements for the prover are

only proportional to the size of a single step, rather the size of the whole computation as with monolithic Succinct Non-interactive Arguments of Knowledge (SNARKs).

This granularity enables several practical applications such as proving the execution of a virtual machine which runs instructions from a pre-determined instruction-set architecture, e.g., RISC-V or the instruction-set of Ethereum Virtual Machine. These instructions are equivalent to instructions in an assembly language. The full specification of a virtual machine also includes the number of registers and the main-memory type (e.g., read and write, read/write-only, static or dynamic). At each IVC step, the prover then shows that a single instruction has been correctly executed and that the new memory state is consistent with the previous state and the executed instruction.

Another recent application of incremental computation are proofs [8] that a public transition of a state machine stems from the private execution of a function in a given public set. The execution of a function is considered private if the proof reveals nothing else than the state transition (and what can be inferred from it). In particular, the proof reveals no information about the executed function nor its arguments.

The classical approach [3,26] to building an IVC scheme for a given function (e.g., the transition function of a virtual machine) is to recursively use a proof system that is universal (i.e., it can be applied to any circuit of a given maximum size) and has succinct verification. At each step, the prover shows that the result at current step is the result of a single application of the function to a result at the previous step, and that it knows a proof of correctness of this previous result that the verifier of the previous step (instantiated with the proof-system verifier) would accept. The downside of this approach is that in its instantiation with the most efficient SNARKs, the verifier performs ElliptiC-Curve operations and computes pairings. These types of operations, especially pairings, are enforced with a large number of high-degree circuit gates, and the larger this amount is, the slower the prover gets.

Accumulation schemes [5] and folding schemes introduced in Nova [17] gave a solution to IVC that significantly improves the efficiency of the prover as there are no pairing equations to prove. They are primitives which reduce the task of checking the membership of two (or more) instances to an NP language to the task of checking the membership of a single instance. To build an IVC scheme from a folding scheme, the idea is to fold at each step an instance that attests to the correct computation at the current step and an instance (an accumulator) that attests to the correct execution of all the previous steps. At the end of the computation, the prover need only prove knowledge of a witness to the membership of the last accumulator. The advantage of using folding schemes compared to the previous approach is that the verifier in existing (pre-quantum) constructions only performs ECC operations; it does not do any pairing computation. It means that when the IVC prover shows that it correctly folded two instances, it must only prove ECC operations.

However, given a group of points on elliptic-curve defined over a prime-order field $\mathbb{F}_q$, if r denotes its (prime) order, proving a group addition in an arithmetic circuit defined over $\mathbb{F}_r$ requires emulating $\mathbb{F}_q$ operations in $\mathbb{F}_r$. This emulation is what is usually referred to as foreign-field or non-native field operations. If $r < q$ as it is for instance the case for the BN254 used on the Ethereum blockchain, proving group operations in an arithmetic circuit incurs a large number of gates, and the prover computation scales at least linearly with this number.

The CycleFold scheme [16] addresses this issue by leveraging an idea reminiscent of a paper by Ben-Sasson, Chiesa, Tromer and Virza [2]. It uses 2-cycles of elliptic curves, i.e., it assumes that there is another group of points on an elliptic-curve defined over $\mathbb{F}_r$ that has order q. To be more precise, it requires half cycles, as this second group need not be pairing friendly. Applied to a folding-based IVC construction, it requires two instance pairs: one pair per curve. The operations in the $\mathbb{F}_q$ curve induced by the folding of two instances (consisting of group elements in that $\mathbb{F}_q$ curve) are accumulated in the instance represented by group elements in the $\mathbb{F}_r$ curve, and vice versa. There are at least two downsides to this approach. First, folding-instances on each curve are now necessary instead of a single one, and even if the second instance is for a small circuit, its validity must also be checked at the end of the computation. Moreover, if the paradigm is instantiated with folding schemes like Protostar [4] or ProtoGalaxy [7] in which the folding verifier performs field operations, the $\mathbb{F}_q$ field operations required to check the folding of the instances in the $\mathbb{F}_r$ curve (i.e., the curve defined over $\mathbb{F}_r$) are non-native to the $\mathbb{F}_r$ field of the group in the $\mathbb{F}_q$ curve, although the correct folding of these instances is constrained in $\mathbb{F}_r$. Delegating these non-native field operations to a subsequent IVC step while enforcing their consistency with the current step is non-trivial and not addressed. The paper leaves as open task the formalisation of a compiler that applies to any folding scheme described over a single curve.

The WARPfold paper [24] aims instead at efficiently simulating non-native arithmetic in IVC schemes based on the Hypernova [18] and Protostar [4] folding schemes, but does not obviate non-native field operations.

1.2 Contributions

This paper introduces an IVC scheme which does not require *any* non-native operations to be proved at any IVC step. The idea is to record (in $\mathbb{F}_r$) at each step the group operations to do, and to only prove them *natively* at the end of the computation. This record is committed and the commitment is given with the IVC proof. The delicate part is to prove that the recorded operations are consistent with the operations required to, for instance fully verify a folding, without introducing new non-native operations. This is precisely what the construction in Section 4 achieves. Importantly, the size of the proof and the verifier computation do *not* scale with the total size of the operation record. The size of the circuit for which the verifier checks proofs only depends on the number of recorded non-native operations at the current step. The instantiation of the

scheme in the Plonk [10] arithmetisation is given in Section 5 and called Goblin Plonk.

The prover eventually only performs non-native operations at the end of the computation to translate into $\mathbb{F}_q$ the operations recorded in $\mathbb{F}_r$. Per scalar multiplication, this translation only requires a constant (four or five) number of multiplications in $\mathbb{F}_q$ to be proved over $\mathbb{F}_r$. In comparison, proving a scalar multiplication in an $\mathbb{F}_r$ circuit would require a number of $\mathbb{F}_q$ multiplications that scales linearly with the binary length of r (which is around 2λ, if λ denotes the security parameter). Section 6 shows how to prove the correctness of this translation, and how to prove the overall validity of the record.

As a proof at a given IVC step now attests to the validity of the result only if the recorded operations are valid, the scheme is not a standard IVC scheme per se. It is rather an IVC with some auxiliary information attached to the proof, and the IVC output is valid only if that information is valid, in addition to the IVC proof. In other words, the knowledge-soundness requirements are relaxed. Section 3 formally defines IVC with auxiliary proof information. It is worth noting that this new notion captures what is tacitly done in recent work such as Mangrove [20] or Stackproofs [8], in which IVC outputs are valid only if some information at the end satisfies a predicate.

Section 3 also describes a framework that encompasses the paradigms behind existing IVC constructions, whether from SNARKs or folding schemes. This framework allows to describe the construction in Section 4 in a way that is not tied to a particular pre-existing standard IVC scheme, and rather shows that the construction applies to all (though not exactly in a black-box manner). The focus can then be only put on the handling of auxiliary proof information, without having to prove the knowledge soundness of the construction anew whenever it is applied to a particular incremental scheme.

Section 7 gives an estimation of the number of gates in a Plonk circuit entailed by the instantiation in Section 5 and the validity proof in Section 6. Compared to a standard folding-based IVC scheme, the estimation shows savings of at least one order of magnitude, which may go up to a factor of 50.

2 Preliminaries

2.1 Notation and Convention

All algorithms are assumed to return an empty string, symbolised by $\varnothing$, whenever they are run on an input not in their defined input sets. Whenever an algorithm returns $\varnothing$, it is said to abort or to return an empty output. An algorithm is termed "efficient" if its run-time is a polynomial function of its input size. Probabilistic algorithms which run in Polynomial Time are referred to as PPT algorithms. Given a PPT algorithm A and binary strings x and r, $\mathrm{A}(x;r)$ denotes the output of A on input x and random string r.

2.2 Bilinear-Group Structures

An asymmetric bilinear group structure consists of a tuple $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$, with p a prime integer, $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ groups of order p, and $e\colon \mathbb{G}_1 \times \mathbb{G}_2 \mapsto \mathbb{G}_T$ a non-degenerate bilinear map. Generators of $\mathbb{G}_1$ and $\mathbb{G}_2$ are respectively denoted $[1]_1$ and $[1]_2$. Define $[1]_T \coloneqq e([1]_1, [1]_2)$. For any $x \in \mathbb{F}$, $[x]_i$ is defined as $x \cdot [1]_i$ for all $i \in \{1, 2, T\}$. A bilinear group structure is of type 3 if there is no efficiently computable homomorphism from $\mathbb{G}_2$ to $\mathbb{G}_1$. All pairings considered herein are of type 3.

2.3 Proof Systems

A proof system for an NP-relation generator R consists of a set-up algorithm $\textsc{Setup}(R) \to (par, \tau)$ that returns public parameters and a trapdoor (which may be an empty string) on the input of a relation $R \leftarrow \mathrm{R}\left(1^\lambda\right)$, and of a pair

$$(\textsc{Prove}(par, x, w), \textsc{Vf}(par, x))$$

of interactive algorithms. Generator R may for instance call on a generator of bilinear group structures, and the NP relation for which proofs are computed is defined over the generated bilinear group structure.

2.3.1 Properties. The proof systems considered in this paper are expected to be complete and knowledge sound. A proof system is complete if Vf accepts any interaction with Prove on a common instance x if the latter is given an input w such that $(x, w) \in R$. Knowledge soundness (also called extractability) requires the existence of a probabilistic algorithm, called extractor, that runs in expected polynomial time and computes a witness for any instance for which a prover makes the verifier accept with a probability above a certain threshold. This threshold is called knowledge-soundness error. The knowledge-soundness error is a function of the security parameter and the size of the instance. The extractor is given black-box access to the prover algorithm and also has control over its random tape. These properties are formally defined in Appendix A.1.

2.3.2 Universal Proof Systems. A universal proof system can be defined as a proof system for a generator which returns relations of the form

$$\{((C, x), w)\colon C(x, w) = 0\},$$

with C being an arithmetic circuit of a size determined by the security parameter. To be accurate, the relations (e.g., Plonkish relations) are distinct but still NP-complete, so a pair (C, x) can be mapped in polynomial-time to an instance of the relation.

These proof systems are sometimes defined with an additional key-generation algorithm which takes as input parameters and a circuit C, and returns proving and verification keys. The verification key is a succinct representation of C.

2.4 Polynomial Commitments

A polynomial-commitment scheme allows a party to commit to a polynomial and to later convince another party of evaluations of the committed polynomial.

2.4.1 Syntax. Given a field $\mathbb{F}$, a scheme to commit to univariate polynomials consists of a set of algorithms as defined below.

Setup $\left(1^\lambda, N_{\max} \in \mathbb{N}_{\geq 1}\right) \to par$: generates public parameters on the input of a security parameter encoded in unary and of a positive integer $N_{\max}$. The latter indicates a strict upper-bound on the maximum degree of the polynomials that are committed to, i.e., the polynomials to be committed to are of degree at most $N_{\max} - 1$. It is here tacitly assumed that the set-up algorithm also expects an auxiliary input which may for instance specify the basis (e.g., monomial or Lagrange) in which the polynomials to be committed to are represented. For simplicity, this input is omitted from the syntax. To lighten the notation, the parameters are given as an implicit input to the algorithms to follow whenever they are clear from the context.

Com $\left(f \in \mathbb{F}[X]^{<N_{\max}}\right) \to (C, r)$: computes a commitment to f (represented as a tuple of at most $N_{\max}$ field elements) and a piece of de-commitment information r, which typically is a random value used to compute the commitment.

Open $(C, f, r) \to b \in \{0, 1\}$: returns a bit indicating whether C is a valid commitment to f w.r.t. the de-commitment information r. The algorithm is said to accept if it returns 1 and to reject otherwise.

Eval : is a proof system for the language

$$\left\{(C, u, v) : \exists \left(f \in \mathbb{F}[X]^{<N_{\max}}, r\right) \ \textsc{Open}(C, f, r) = 1 \text{ and } f(u) = v\right\}.$$

The bound $N_{\max}$ on the degree of the witness f is here determined by *par*.

2.4.2 Hiding KZG Commitments. Polynomial-delegation schemes are similar to polynomial-commitment schemes, except that the evaluations for which proofs are computed are also committed. Zhang, Genkin, Katz, Papdopoulos and Papamanthou [28] proposed a polynomial-delegation scheme which is similar in spirit to the pairing-based polynomial-commitment scheme due to Kate, Zaverucha and Goldberg [13]. The scheme which follows is a variation of standard KZG commitments and is inspired by their construction. Kohrita and Towa [14] proved it to be knowledge-sound under the q-DLOG assumption.

Setup $\left(1^\lambda, N_{\max} \in \mathbb{N}_{\geq 1}\right)$:

$\mathbb{G} \coloneqq (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e) \leftarrow \mathrm{G}\left(1^\lambda\right)$

$srs \leftarrow \left([1]_1, [\tau]_1, \ldots, \left[\tau^{N_{\max}-1}\right]_1, [\xi]_1, [1]_2, [\tau]_2, [\xi]_2\right)$ for $\tau, \xi \leftarrow_{\$} \mathbb{F}^*$

Return $par \leftarrow (\mathbb{G}, srs)$.

Com $(f \coloneqq (a_0, \ldots, a_{N-1}))$:

$r \leftarrow_{\$} \mathbb{F}$

$C \leftarrow a_0 \cdot [1]_1 + a_1 \cdot [\tau]_1 + \cdots + a_{N-1} \cdot \left[\tau^{N-1}\right]_1 + r \cdot [\xi]_1 = [f(\tau)]_1 + r \cdot [\xi]_1$

Return (C, r).

Open $(C, f \coloneqq (a_0, \ldots, a_{N-1}), r)$:

$$C \stackrel{?}{=} a_0 \cdot [1]_1 + a_1 \cdot [\tau]_1 + \cdots + a_{N-1} \cdot \left[\tau^{N-1}\right]_1 + r \cdot [\xi]_1.$$

Eval:

Prove $\rightarrow$ **Vf** :

$q \leftarrow f - v : (X - u)$

$s \leftarrow_{\$} \mathbb{F}$

$\pi \leftarrow [q(\tau)]_1 + s \cdot [\xi]_1$

$\delta \leftarrow r \cdot [1]_1 - s \cdot [\tau]_1 + (s \cdot u) \cdot [1]_1 = [r - s(\tau - u)]_1$

Output (π, δ)

Vf : $e\left(C - v \cdot [1]_1, [1]_2\right) \stackrel{?}{=} e\left(\pi, [\tau]_2 - u \cdot [1]_2\right) + e\left(\delta, [\xi]_2\right).$

2.5 Folding Schemes

A folding scheme [17] is a cryptographic primitive that reduces the task of verifying the membership of two (or more) instances to an NP language to that of verifying the membership of a single instance. Folding schemes are reminiscent of accumulation schemes [5].

Formally, a folding scheme for an NP-relation generator R consists of set-up algorithm $\textsc{Setup}(R) \rightarrow (par, \tau)$ that returns public parameters on the input of a relation $R \leftarrow \mathrm{R}\left(1^{\lambda}\right)$, (let L denote the corresponding language) and of a pair $(\textsc{Prove}(par, (x_0, x_1), (w_0, w_1)), \textsc{Vf}(par, (x_0, x_1)) \rightarrow (w, x)$ of interactive algorithms. Both algorithms run on a couple (x_0, x_1) of instances. Algorithm Prove is also given witnesses w_0 and w_1 for the membership of x_0 and x_1 to L. At the end of the interaction, Prove returns a witness w and Vf returns an instance x. Instance x is referred to as the folding of x_0 and x_1[3]. A folding scheme is only useful in practice if the computational costs of the verifier added to the costs to verify that (x, w) is in R are strictly less than the costs to verify that (x_0, w_0) and (x_1, w_1) are in R.

2.5.1 Properties. A folding scheme should be complete and knowledge sound. The completeness property requires that $(x, w) \in R$ if w_0 and w_1 are valid witness for x_0 and x_1. A folding scheme is knowledge sound if there exists an

[3] The terminology of proof systems ("prover," "verifier," and "proof" in the case of non-interactive schemes) is perhaps unfitting for folding schemes. A folding verifier does not accept or reject; it simply returns a new instance computed from its input instances and the messages exchanged with a prover. The terms "(folded) instance generator," "witness generator" and "folding string" might be better suited.

extractor that returns valid witnesses w_0 and w_1 for the memberships of x_0 and x_1, when given rewinding access to any prover that convinces the verifier and returns a valid witness with a probability above a certain threshold. That threshold is the knowledge-soundness error. Formal definitions of these properties are given in Appendix A.2.

2.5.2 Comparison with the Nova Definition. The definition of folding schemes in the Nova series of papers [15, 17, 18] additionally includes a key-generation algorithm that takes parameters as input and returns proving and verification keys. That is because the constructions they give only allow to fold instances for the same R1CS structure, which essentially means for the same circuit. (The Nova paper [17] splits an R1CS instance as classically defined between the matrices which define a circuit, now called strcuture, and the rest of the instance that is just called instance.)

Instances of a Plonk relation can be folded with the Protostar [4] and ProtoGalaxy [7] folding schemes. As explained in Section 2.3.2, that relation can be mapped to a relation of the form $\{((C, x), w) \colon C(x, w) = 0\}$, so ProtoGalaxy folds pairs of the form (C, x). (Circuit C is described by Plonk selectors.) It means that ProtoGalaxy actually folds different circuits and their inputs. In contrast, a folding instance in Nova is not a pair (C, x) that consists of a circuit and its instance, but rather only of a circuit instance x. That is, the Nova folding scheme is for relations of the form $\{(x, w) \colon C(x, w) = 0\}$, with the relation being parametrised by a circuit C. The relation for ProtoGalaxy is not parametrised by a fixed circuit, and it is for this reason that the above definition of folding schemes does not involve a key-generation algorithm.

3 Definitions

This section first gives a classical definition of Incrementally Verifiable Computation (IVC) for (ternary) NP relations. It next proposes an alternative definition from which the rationale behind all existing constructions can be explained in a simple and unified manner. It then introduces IVC with auxiliary proof information, which essentially is IVC with a relaxed knowledge soundness notion. This relaxation is precisely what will later enable constructions that are more efficient that what the standard IVC definition permits.

3.1 Incrementally Verifiable Computation

Existing definitions of IVC schemes are given for NP functions $F \colon (x, w) \mapsto y$ which implies that to be able to repeat applications of F, its output set must be included in its left-input set. However, most functions for which IVC schemes are built are in fact binary-output functions (i.e., predicates), and $\{0, 1\}$ is not necessarily a subset of its left-input set. This realisation motivates the following definitions which is given for general NP relations. It is worth noting that the

definition of proof-carrying data (a generalisation of IVC) due to Bünz, Chiesa, Mishra and Spooner [6] is similarly given for predicates.

To show the definition of IVC for NP relations captures that for NP functions which need not be predicates, consider a generator $\lambda \mapsto \mathrm{F}\left(1^{\lambda}\right)$ of NP functions with arbitrary input and output sets. An IVC scheme for F can be defined as an IVC scheme for the relation generator R that runs $F \leftarrow \mathrm{F}\left(1^{\lambda}\right)$ and returns relation

$$R_F \coloneqq \{((x, y), w) \colon F(x, w) = y\}.$$

Definition. Consider an NP-relation generator R. An Incrementally Verifiable Computation (IVC) scheme for R consists of a set of algorithms as defined hereafter.

Setup $(R) \to par$: on the input of a relation $R \leftarrow \mathrm{R}\left(1^{\lambda}\right)$, it generates public parameters. The NP language corresponding to R is further denoted L.

Prove$(par, (i, x_0, x_{i+1}), x_i, \pi_i, w_i) \to \pi_{i+1}$: given a non-negative integer i, a proof π_i (in case $i \geq 1$) that there exist sequences $x_1, \ldots, x_{i-1}$ (only if $i \geq 1$, and just x_1 if $i = 1$) and $w_0, \ldots, w_{i-1}$ such that

$$((x_0, x_1), w_0), \ldots, ((x_{i-1}, x_i), w_{i-1}) \in R,$$

and a witness w_i, this algorithm computes a proof π_{i+1} that there exist sequences $x_1, \ldots, x_i$ and $w_0, \ldots, w_i$ such that

$$((x_0, x_1), w_0), \ldots, ((x_i, x_{i+1}), w_i) \in R.$$

Vf$(par, (i + 1, x_0, x_{i+1}), \pi_{i+1}) \to \{0, 1\}$: given a non-negative integer i, this algorithm deterministically returns a bit indicating whether π_{i+1} is a valid proof.

The integers that Prove and Vf receive as input (i and $i + 1$ respectively) indicate the number of IVC steps that have already been executed when the algorithms are run. The number of executed IVC steps is here understood as the number of number of IVC proofs that have already been computed.

Alternative Definition. An IVC scheme can equivalently be defined as a sequence of knowledge-sound proof systems $\left(P^{(i)}, V^{(i+1)}\right)_{i \in \mathbb{N}}$ (sharing the same parameters), each for an NP language $L^{(i)}$. Let

$$L^{(0)}_{\mathrm{IVC}} \coloneqq L = \{(x_0, x_1) \colon \exists w_0 \ ((x_0, x_1), w_0) \in R\}$$

and

$$\begin{aligned} L^{(i)}_{\mathrm{IVC}} \coloneqq \{ \, (x_0, x_{i+1}) \colon \exists \, (w_0, x_1, w_1 \ldots, x_i, w_i) \\ ((x_0, x_1), w_0) \, , \ldots, ((x_i, x_{i+1}), w_i) \in R \ \} \end{aligned}$$

for all positive integers i. For all $i \in \mathbb{N}$, Language $L^{(i)}$ must include $L^{(i)}_{\mathrm{IVC}}$ (to be able to compute valid proof for any instance in $L^{(i)}_{\mathrm{IVC}}$), and must be such that it is hard to compute in PPT an instance in $L^{(i)}$ that is not in $L^{(i)}_{\mathrm{IVC}}$.

Note that $V^{(0)}$ is not defined. That is because x_0 is the starting point of the computation and there is thus no proof π_0 to verify. Moreover, $V^{(i+1)}$ verifies proof π_{i+1} computed by $P^{(i)}$ and which attests to the validity of x_{i+1}.

Nevertheless, the prover in the definition of IVC is given as witness not a tuple $(w_0, x_1, w_1, \ldots, x_i, w_i)$ but rather a previous output x_i, a proof π_i that it stems from a computation that starts from x_0 and a witness w_i for the membership of (x_i, x_{i+1}) to $L^{(i)}$. This expresses the idea that it should be possible to carry on an IVC from any step as long as the output is proved to be correct; there should be no need to know intermediate computation results or witnesses. To achieve it, existing IVC constructions define a sequence of predicates $\left(F^{(i)}\right)_{i\in\mathbb{N}}$ and build a sequence $\left(P^{(i)}, V^{(i+1)}\right)_{i\in\mathbb{N}}$ of knowledge-sound proof systems for the NP languages (parametrised by parameters *par*)

$$L^{(i)} \coloneqq \left\{ (x_0, x_{i+1}) \colon \exists (x_i, \rho_i, w_i)\ F^{(i)}\left(par, (x_0, x_{i+1}), x_i, \rho_i, w_i\right) = 1 \right\}.$$

These sequences of proof systems share the same parameter-generation algorithm, which is omitted for simplicity, and the proving and verification algorithms are run on the same parameters throughout the sequence. Note that as $P^{(i)}$ is given π_i as witness input, ρ_i should be efficiently and deterministically computable from (par, x_0, x_i, π_i) if $V^{(i)}$ accepts it.

Function $F^{(0)}$ generally is a predicate that is satisfied if its second and third arguments are equal and if $((x_0, x_1), w_0) \in R$ (input ρ_0 is the empty string). Language $L^{(0)}$ is then simply $L^{(0)}_{\mathrm{IVC}}$. For any positive i, predicates $F^{(i)}$ are defined so that an instance (x_0, x_{i+1}) is in $L^{(i)}$ if there exists (x_i, π_i, w_i) such that $((x_i, x_{i+1}), w_i) \in R$ and that $V^{(i)}$ accepts on input (par, x_0, x_i, π_i). It is the case for any (x_0, x_{i+1}) in $L^{(i)}_{\mathrm{IVC}}$ (it suffices to run the previous provers) and therefore, $L^{(i)}_{\mathrm{IVC}} \subseteq L^{(i)}$.

Conversely, languages $L^{(i)}$ are such that if a polynomial-time algorithm can compute a proof π_{i+1} that an instance (x_0, x_{i+1}) is in $L^{(i)}$, then that instance is likely in

$$\left\{ (x_0, x_{i+1}) \colon \exists (x_i, \pi_i, w_i)\ \ ((x_i, x_{i+1}), w_i) \in R \right. \\ \left. \text{and } V^{(i)}\left(par, (x_0, x_i), \pi_i\right) = 1 \right\}.$$

Moreover, a valid proof π_i can be efficiently extracted from the algorithm which computes π_{i+1} and on an instance determined by ρ_i. The knowledge soundness of the proof systems imply that (x_0, x_{i+1}) is likely to be in $L^{(i)}_{\mathrm{IVC}}$. In other words, it is hard to efficiently compute an instance in $L^{(i)} \setminus L^{(i)}_{\mathrm{IVC}}$.

Recursive SNARK Bootstrapping. In the case of recursive SNARK bootstrapping [3], for any positive integer i, function $F^{(i)}$ accepts if and only if $((x_i, x_{i+1}), w_i) \in R$ and $V^{(i)}$ accepts on $(par, (x_0, x_i), \pi_i =: \rho_i)$. More precisely,

the construction assumes that a universal SNARK (P, V) is given. Its parameters are those on which the proof systems in the sequence are run. $\left(P^{(0)}, V^{(1)}\right)$ is defined as (P, V) applied to a circuit that $((x_0, x_1), w_0)$ satisfies if and only if it is in R. Proof system $\left(P^{(1)}, V^{(2)}\right)$ is now (P, V) applied to a circuit that $((x_0, x_2), (x_1, \pi_1, w_1))$ satisfies if and only if $((x_1, x_2), w_1)$ is in R and $V^{(1)}$ accepts proof π_1 for instance (x_0, x_1). In general, $\left(P^{(i)}, V^{(i+1)}\right)$ is (P, V) applied to a circuit that $((x_0, x_{i+1}), (x_i, \pi_i, w_i))$ satisfies if and only if $((x_i, x_{i+1}), w_i)$ is in R and $V^{(i)}$ accepts proof π_i for instance (x_0, x_i). Although it is always the same universal SNARK that is used, the circuit to which it is applied changes at each iteration.

The construction of Proof-Carrying Data (a generalisation of IVC) due to Bitansky, canetti, Chiesa and Tromer [3] is actually expressed somewhat differently. The authors define a random-access machine that they call PCD machine, and of which the input is equivalent to $(i, (x_0, x_{i+1}), x_i, \pi_i, w_i)$ in the notation herein. The machine accepts if $((x_i, x_{i+1}), w_i) \in R$ and if the SNARK verifier would accept π_i as a proof that x_i is the result of i invocation the PCD machine from x_0. This definition is circular as the definition of the PCD machine depends on the PCD machine itself. However, an application of Kleene's recursion theorem resolves this apparent contradiction. In contrast, the inductive definition in the previous paragraph is not circular and conveys the same idea. One of the IVC constructions of Valiant in his seminal paper on IVC [26, Construction 2] is similarly sequential.

Folding-Based Constructions. In the case of IVC schemes built from folding schemes [17], function $F^{(i)}$ mainly checks that $((x_i, x_{i+1}), w_i) \in R$, that a claimed instance matches the folding of two instances, and that these two instances are valid. Their validity attests to the correct execution thus far. More precisely, IVC proofs

$$\pi_i \coloneqq ((U_i, V_i), (u_i, v_i), r_i)$$

consist of two instance-witness pairs that are in the folding-scheme relation and some randomness. Each instance is of the form (C, x), with C denoting an arithmetic circuit and x an instance for that circuit. Instance x is an input-output pair (I, O) of the function of which C attests to the correct computation. The first instance attests to the correct execution of functions $F^{(j)}$ for all $j < i - 1$, and the second attest to the correct execution of $F^{(i-1)}$. These functions mainly run one IVC step and ensure that the input to the corresponding step is the correct output from all the previous steps. From an IVC proof π_i as above, the IVC prover folds the two instances U_i and u_i into U_{i+1}, and computes a non-interactive folding proof $\pi_{\mathrm{FS},i+1}$. Next, it computes a hash h_{i+1} (with a function H) of the function output at the current steps with randomness r_{i+1}. This hash acts as a compressing commitment to the outputs, so as to bound the size of the public-output part of the folding instances, while chaining the computation by making U_{i+1} part of the public outputs of u_{i+1}. The prover then sets

$$\rho_i \leftarrow ((U_i, u_i), U_{i+1}, \pi_{\mathrm{FS},i+1}, r_i, r_{i+1}, h_{i+1}).$$

Formally, function $F^{(0)}$ simply checks that $((x_0, x_1), w_0) \in R$ (U_0, u_0 and U_1 are undefined and not part of its inputs) and that $h_1 = H(par, 1, (x_0, x_1), r_1)$. The parameters *par* given as input to the hash function H are parameters for a folding scheme FS. Function $F^{(1)}$ enforces that $((x_1, x_2), w_1) \in R$, that $u_1.C = F^{(0)}$, that

$$u_1.x.\mathrm{O} = H(par, 1, (x_0, x_1), r_1),$$

and that $U_2 \coloneqq u_1$ is part of the public output of instance u_2, i.e., that

$$h_2 = H(par, 2, (x_0, x_2), u_1, r_2).$$

For $i \geq 2$,

$F^{(i)}\,(par, (x_0, x_{i+1}), x_i, \rho_i, w_i)$:

Return 1 if

$((x_i, x_{i+1}), w_i) \in R$

$u_i.C = F^{(i-1)}$

$\mathrm{FS.V_F}\,(par, (U_i, u_i), \pi_{\mathrm{FS},i+1}) = U_{i+1}$

$u_i.x.\mathrm{O} = H(par, i, (x_0, x_i), U_i, r_i)$

$h_{i+1} = H(par, i+1, (x_0, x_{i+1}), U_{i+1}, r_{i+1})$

Else return 0.

This explanation departs from that of the Nova paper. The Nova IVC scheme only defines a single "augmented function" which takes $(par, i, (x_0, x_{i+1}), x_i, \rho_i, w_i)$ as argument, tests whether i is 0 or at least 1, and returns what $F^{(i)}$ does on $(par, (x_0, x_{i+1}), x_i, \rho_i, w_i)$. There is still a slight discrepancy as in the Nova paper, the prover returns an empty accumulator U_1 at step 0, and that is folded with u_1 at step 1. In that case $F^{(1)} = F^{(2)}$ and that is why the case distinction on i is only between 0 and otherwise, rather than 0, 1 and otherwise. However, folding an empty accumulator U_1 with u_1 is equivalent to setting $U_2 \leftarrow u_1$ as above.

Note that using a single function of which index i is a field element, as it is the case therein, means that the IVC scheme is only defined for i bounded by the field size. The sequential definition of functions $F^{(i)}$ does not have this restriction, although this difference is irrelevant in practical IVC applications when the field size is large, e.g., 2^{256}.

Besides, defining a single augmented function may seem circular as the folding instances attests to the correct execution of the augmentation function being defined. Yet, as the set in which to which the instances belong is fixed when parameters are generated, the augmented function can just be defined w.r.t. elements in these sets, and it is only once the function is defined that these elements are interpreted as folding instances. The previous sequential definition once again avoids this apparent circularity issue in the definition.

Notice that function $F^{(i)}$ also includes a check that $u_i.C = F^{(i-1)}$, i.e., that the circuit in instance u_i is indeed the previous function $F^{(i-1)}$ in the sequence.

(Function $F^{(i-1)}$ is here conflated with the circuit that attests to its correct computation.) The Nova augmented function does not have such a test because the Nova folding scheme folds instances for a particular circuit, and its knowledge-soundness definition guarantees that the extractor returns witnesses for the two input instances for the *same* circuit. Recall from Section 2.5.2that a folding instance in Nova is not a pair consisting of a circuit and its instance (C, x), but only a circuit instance x. In particular, only the IVC verifier checks at the end that the folding instance-witness pairs in the proof are valid for the circuit of the augmented function, i.e., that $((F', U_{i+1}), V_{i+1})$ and $((F', u_{i+1}), v_{i+1})$ are valid, with F' denoting the Nova augmented function. If that is the case, by the knowledge soundness of the Nova folding scheme, the extractor returns witnesses V_i and v_i for folding instances U_i and u_i (deduced from v_{i+1}) such that $((F', U_i), V_i)$ and $((F', u_i), v_i)$ are valid. It is thus guaranteed at the next extraction step that u_i is an instance for the same augmented function.

If the Nova IVC scheme were instead instantiated with a folding scheme that folds circuits and their instances, like Protostar [4] and ProtoGalaxy [7], it would be necessary for the augmented function to check at each step i that circuit $u_i.C$ is indeed F', for the IVC scheme to be knowledge-sound.

Limitations and Alternative. Despite its elegance and simplicity, this approach to building IVC incurs significant costs as it requires $P^{(i)}$ to prove a statement about a SNARK or folding verifier. Even though folding schemes greatly reduced these costs compared to the SNARK-bootstrapping approach (because folding verifiers do not perform pairing operations), folding-scheme verifiers [4, 17] typically performs ElliptiC-Curve (ECC) operations. Unfortunately, ECC operations induce a large number of high-degree constraints in the circuit of which $P^{(i)}$ proves the satisfiability, even with the most practical circuit arithmetisations [10, 22]; and the higher the number and degree of constraints, the slower provers are. That is because the arithmetisation is done in the scalar field of the ECC group, which means that arithmetising ECC operations requires to simulate operations from the *base* field of the ECC in its scalar field.

This observation motivates the following idea: instead of proving the correctness of expensive verifier operations at each step, simply accumulate (at each step and) into some information auxiliary to the IVC proof constraints to prove the input-output statements of these expensive operations. Proving the correctness of expensive operations can thereby be deferred and proved at once for multiple steps[4]; and the IVC proof now only guarantees the correctness of a computation output *up to the validity of* the expensive operations accumulated in *the auxiliary information*. To formalise this idea, the next section introduces IVC with auxiliary proof information.

[4] Section 6 shows how to prove the validity of these operations in their native field, which is why the prover is eventually faster than in standard IVC constructions.

3.2 Incrementally Verifiable Computation with Auxiliary Proof Information

Incrementally verifiable computation with auxiliary proof information relaxes the soundness requirement of IVC insofar as a proof attests to the validity of a computation only if some auxiliary proof information is in a predetermined language. The elements of that language are later qualified as valid. The relaxation enables constructions that are more efficient than those of classical IVC, as operations that are expensive in the proof computation can be included in the auxiliary proof information rather than proved at each step. Assuming the existence of an efficient scheme to prove the validity of auxiliary information, those expensive operations can eventually be proved at once after a desired number of computation steps. Together with the relaxed IVC proof, a validity proof for the auxiliary information attests to the validity of the computation.

Definition. Consider a generator G which returns the description of an NP relation R and of a family of NP relations $R_{aux}^{(i)}$ defined for all positive integers i. (Although there might be infinitely many distinct relations in the family, the description should be finite.) Denote by $L_{aux}^{(i)}$ the language corresponding NP language. Relation R is the main relation for which the IVC is computed. Language $L_{aux}^{(i)}$ consists of the pieces of auxiliary proof information that are considered valid after i computation steps.

An Incrementally Verifiable Computation (IVC) scheme for G consists of a set of algorithms as defined below.

Setup $\left(R, \left(R_{aux}^{(i)}\right)_{i\geq 1}\right) \rightarrow par$: generates public parameters on the input of a relation and a sequence of relations

$$\left(R, \left(R_{aux}^{(i)}\right)_{i\geq 1}\right) \leftarrow \mathrm{G}\left(1^{\lambda}\right).$$

Prove$(par, (i, x_0, x_{i+1}), x_i, \pi_i, aux_i, st_i, w_i) \rightarrow (\pi_{i+1}, aux_{i+1}, st_{i+1})$: receives as input a non-negative integer i, a proof π_i (in case $i \geq 1$) that either $aux_i \notin L_{aux}^{(i)}$ or there exist sequences $x_1, \ldots, x_{i-1}$, (only if $i \geq 1$, and just x_1 if $i = 1$) and $w_0, \ldots, w_{i-1}$ such that

$$((x_0, x_1), w_0), \ldots, ((x_{i-1}, x_i), w_{i-1}) \in R,$$

a state st_i (only if $i \geq 1$) and a witness w_i. This algorithm computes an auxiliary piece of proof information aux_{i+1} as well as a proof π_{i+1} that either $aux_{i+1} \notin L_{aux}^{(i+1)}$ or there exist sequences $x_1, \ldots, x_i$ and $w_0, \ldots, w_i$ such that

$$((x_0, x_1), w_0), \ldots, ((x_i, x_{i+1}), w_i) \in R,$$

and updates the prover state to st_{i+1}. State st_i is a witness for the membership of aux_i to $L_{aux}^{(i)}$ and contains information that the prover needs to

compute aux_{i+1}, and state st_{i+1} information that the prover needs at the next step.

$\mathbf{V_F}(par, (i+1, x_0, x_{i+1}), \pi_{i+1}, aux_{i+1}) \to \{0,1\}$: given a non-negative integer i, this algorithm deterministically returns a bit indicating whether π_{i+1} is a valid proof w.r.t. aux_{i+1}.

Formal definitions of completeness and knowledge soundness are given in Appendix B.

Remark that standard IVC is a special case of IVC with auxiliary proof information in which the auxiliary information and the prover state are empty at all steps.

Alternative Definition. Similarly to classical IVC, IVC with auxiliary proof information can be defined as a sequence of knowledge-sound proof systems $\left(P^{(i)}, V^{(i+1)}\right)_{i\in\mathbb{N}}$ (sharing the same parameters), each for an NP language $L^{(i)}$. Let

$$L^{(0)}_{\mathrm{IVC-aux}} = \left\{(x_0, x_1, aux_1) \colon aux_1 \notin L^{(1)}_{aux} \text{ or } \exists w_0 \ ((x_0, x_1), w_0) \in R\right\}$$

and

$$L^{(i)}_{\mathrm{IVC-aux}} \coloneqq \{\ (x_0, x_{i+1}, aux_{i+1}) \colon aux_{i+1} \notin L^{(i+1)}_{aux} \text{ or}$$
$$\exists (w_0, x_1, w_1, \ldots, x_i, w_i)\ \ ((x_0, x_1), w_0), \ldots, ((x_i, x_{i+1}), w_i) \in R\ \}$$

for all positive integers i. The prover state is (witnesses and) not part of the instance as the verifier does not have access to it. For all $i \in \mathbb{N}$, language $L^{(i)}$ must include the language of instances $(x_0, x_{i+1}, aux_{i+1})$ for which there exist intermediate computation results (if $i \geq 1$) and valid witnesses[5]. Language $L^{(i)}$ should also be such that it is hard to compute in PPT an instance in $L^{(i)}$ for which there do not exist intermediate witnesses and computation results.

The next section then shows how to build an IVC with auxiliary proof information from a standard IVC that follows the paradigm highlighted in Section 3.1.

[5] To be able to compute valid proofs for any such instance. This is not required for instances with an invalid aux_{i+1} and for which intermediate computation results and valid witnesses do not exist.

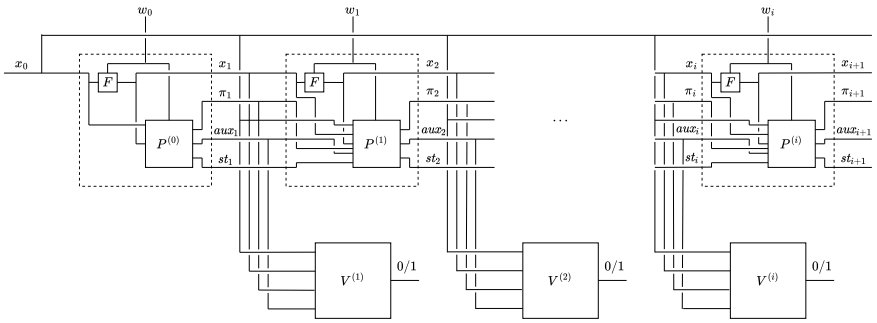

Fig. 1. Illustration of incrementally verifiable computation with auxiliary proof information for a generator of relations of the form $\{((x, y), w) \colon F(x, w) = y\}$.

4 Construction

This section presents an IVC scheme with auxiliary proof information with no expensive operation to be proved at any step. It is built from any IVC scheme that follows the same model that existing constructions do, and from other standard cryptographic primitives.

Decomposition of Predication Functions. As explained in Section 3.1, each of the existing constructions of IVC schemes can be viewed as a sequence $\left(P^{(i)}, V^{(i+1)}\right)_{i\in\mathbb{N}}$ of knowledge-sound proof systems for NP languages

$$L^{(i)} \coloneqq \left\{(x_0, x_{i+1}) \colon \exists(x_i, \rho_i, w_i)\ F^{(i)}\left(par, (x_0, x_{i+1}), x_i, \rho_i, w_i\right) = 1\right\},$$

given a sequence $\left(F^{(i)}\right)_{i\in\mathbb{N}}$ of predicates. These sequences are such that if (x_0, x_{i+1}) is in $L^{(i)}$, then a proof π_i such that

$$V^{(i)}\left(par, (x_0, x_i), \pi_i\right) = 1$$

can be extracted from a membership witness.

Let i_0 be the computation step from which function $F^{(i)}$ contains expensive operations. That is, i_0 is a non-negative integer such that for all $i \geq i_0$, functions $F^{(i)}$ can be written as a composition $F^{(i)}_{k-1} \circ \cdots \circ F^{(i)}_0$ of $k \geq 2$ functions that are alternately inexpensive and expensive, or expensive and inexpensive. A function is here considered "expensive" if it incurs a number of circuit gates or a degree of gate constraints in a chosen arithmetisation that is above a desired threshold. This threshold is at least high enough to consider a check that $((x_i, x_{i+1}), w_i) \in R$ as inexpensive[6]. The goal is to defer execution proofs of expensive operations by accumulating their inputs and outputs into some auxiliary information, and only prove inexpensive operations at each IVC step.

[6] As $L^{(0)}_{\mathrm{IVC}} = L$, function $F^{(0)}$ typically only checks that $((x_0, x_1), w_0) \in R$, which is considered inexpensive, so $i_0 > 0$. Note that in the case of recursive SNARK

Integer k can be assumed to be 2 without loss of generality because the techniques to follow readily extend to larger values of k. Function $F_0^{(i)}$ can be assumed to be inexpensive and $F_1^{(i)}$ expensive for the same reason. For instance, in the case of recursive SNARK bootstrapping, $F_0^{(i)}$ may include only the field operations performed by the verifier and $F_1^{(i)}$ may represent its ECC operations. (As ECC operations can be simulated with field operations, the decomposition need not be unique.) In other words, function $F^{(i)}$ is further written as $F_1^{(i)} \circ F_0^{(i)}$, with $F_0^{(i)}$ inexpensive and $F_1^{(i)}$ expensive. Language $L^{(i)}$ can then be rewritten as

$$\Big\{ (x_0, x_{i+1}) \colon \exists (x_i, \rho_i, w_i, \sigma_i)\ F_0^{(i)}\,(par, (x_0, x_{i+1}), x_i, \rho_i, w_i) = \sigma_i \\ \text{and } \sigma_i \in L_{F_1^{(i)}} \Big\},$$

with $L_{F_1^{(i)}}$ denoting the language of inputs accepted by $F_1^{(i)}$. Let $R_{F_1^{(i)}}$ stand for the corresponding NP relation. Language $L^{(i)}$ is then

$$\Big\{ (x_0, x_{i+1}) \colon \exists (x_i, \rho_i, w_i, \sigma_i, w_{\sigma,i})\ F_0^{(i)}\,(par, (x_0, x_{i+1}), x_i, \rho_i, w_i) = \sigma_i \\ \text{and } (\sigma_i, w_{\sigma,i}) \in R_{F_1^{(i)}} \Big\}.$$

Construction Outline. The idea of the construction is to only prove the inexpensive part $F_0^{(i)}$ of $F^{(i)}$ at each step i, and accumulate their outputs in a running commitment. To prove that this accumulation was properly done, the prover separately commits to the output of $F_0^{(i)}$ at the current step, and proves that the new accumulator opens to the concatenation of the previous accumulator and the current output of $F_1^{(i)}$. This proof is then to be verified at the next IVC iteration. However, verifying a concatenation proof may itself incur expensive operations. The concatenation verifier algorithm is thus itself split into expensive and inexpensive parts. The prover at the next iteration only proves the inexpensive part and concatenates its output with the output of $F_0^{(i)}$ before proving that the accumulation was correctly done.

Building Blocks. The construction makes use of the following algorithms.

Γ : a scheme to commit to vectors of a maximum length, say $\ell_{\max}$, given as input to its parameter-generation algorithm. It will be used to commit to instance-witness pairs relations to be determined below. The bound on the maximum vector length will in turn imply a bound on the maximum number of IVC steps that can be proved.

bootstrapping, $F^{(1)}$ already implements $V^{(1)}$ which is generally considered expensive, and thus $i_0 = 1$. However, in the case of folding-based IVC, $F^{(1)}$ is inexpensive as there is no check of a folding verifier (see Section 3.1), but $F^{(2)}$ is expensive, and therefore $i_0 = 2$.

$\Pi_{\|}$: a non-interactive proof system to prove that a committed vector is the concatenation of two other committed vectors. That is, a proof system for the relation

$$\Big\{ \big((C_0,(C_1,\ell),C_2),(A_k,r_k)_{k=0}^2\big) : \Gamma.\textsc{Open}\,(C_k,A_k,r_k)=1 \\ \text{and } (A_0 = A_1 \parallel A_2 \text{ or } \text{length}(A_1)\neq\ell \text{ or } \ell+\text{length}(A_2)>\ell_{\max}) \Big\}.$$

Algorithms $\Pi_{\|}.\textsc{Prove}$ and $\Pi_{\|}.\textsc{Vf}$ are further denoted $\textsc{Prove}_{\|}$ and $\textsc{Vf}_{\|}$. Note that a proof guarantees concatenation only if the length of the vector committed in C_1 is of the length given as part of the public instance. (This relaxation is to enable efficient instantiations.) This proof system should therefore only be used if there is another mechanism to ascertain that this length requirement is satisfied. The condition $\ell+\text{length}(A_2)>\ell_{\max}$ will be enforced by the predicates to follow.

As $\textsc{Vf}_{\|}$ may itself incur operations that are expensive to be verified in-circuit, the function this algorithm computes is also decomposed into two parts, $\textsc{Vf}_{\|,0}$ and $\textsc{Vf}_{\|,1}$, the former inexpensive and the latter expensive, such that

$$\textsc{Vf}_{\|} = \textsc{Vf}_{\|,1} \circ \textsc{Vf}_{\|,0}.$$

In fact, with the upcoming instantiation in mind, function $\textsc{Vf}_{\|,1}$ is further decomposed into two functions $\textsc{Vf}'_{\|,1}$ and $\textsc{Vf}''_{\|,1}$. In the instantiation, $\textsc{Vf}'_{\|,1}$ performs ECC operations and $\textsc{Vf}''_{\|,1}$ performs pairing computations. Further denote by $L_{\textsc{Vf}'_{\|,1}}$ the input-output language of $\textsc{Vf}'_{\|,1}$ and by $R_{\textsc{Vf}'_{\|,1}}$ the corresponding relation. The language of strings that $\textsc{Vf}''_{\|,1}$ accepts is supposed to be in the complexity class P (in practice, it would be a set of ECC points which satisfy a pairing equation). The auxiliary proof information will then (also) contain the outputs of $\textsc{Vf}'_{\|,1}$ from each IVC step. However, for the auxiliary information not to grow indefinitely, the outputs of $\textsc{Vf}'_{\|,1}$ at each iteration is accumulated into a constant-size piece of information. That can of course be done with a commitment scheme, as with the outputs of $F_0^{(i)}$, but to further reduce proving costs, the instantiation leverages the linear structure of the set of ECC points. More precisely, it computes a random linear combination of a vector of ECC points which accumulates the outputs of $\textsc{Vf}'_{\|,1}$ in all the previous rounds, and its output in the current round. This property is captured in the general case by the existence of an algorithm as follows.

$\textbf{Acc}(T_{i-1},\bar{\tau}_i)\mapsto T_i$: a PPT algorithm such that if T_i is in $L_{\textsc{Vf}''_{\|,1}}$ (the language of strings that $\textsc{Vf}''_{\|,1}$ accepts) then, except with probability $\varepsilon_{\textsc{Acc}}$, inputs T_{i-1} and $\bar{\tau}_i$ are as well. Probably $\varepsilon_{\textsc{Acc}}$ may depend on the algorithm which computes the triple. Conversely, if T_{i-1} and $\bar{\tau}_i$ are in $L_{\textsc{Vf}''_{\|,1}}$, then T_i is as well with probability 1. The function it computes should be inexpensive. Although this algorithm is probabilistic, the predicates to come, which are not probabilistic, must check its correct execution. The randomness $\textsc{Acc}$

uses is then to be derived deterministically, while remaining unpredictable to any efficient prover. The construction thus uses a hash function to do so.

H : a hash function, modelled as a random oracle, that is used to derive randomness for algorithm ACC by hashing its inputs. Define

$$R_{\text{Acc}} \coloneqq \{(((T', \bar{\tau}), T), w_{\text{Acc}}) \colon \text{Acc}\,(T', \bar{\tau}; H\,(T', \bar{\tau})) = T\}\,.$$

The auxiliary proof information further contains a witness $w_{\text{Acc},i}$ for the correct accumulation of T_{i-1} and $\bar{\tau}_i$ into T_i.

$\mathbf{R}$: an NP-relation generator.

$\left(F^{(i)} = F_1^{(i)} \circ F_0^{(i)}\right)_{i \in \mathbb{N}}$: a sequence of predicates parametrised by a relation R in the range of R.

$\left(P^{(i)}, V^{(i+1)}\right)_{i \in \mathbb{N}}$: a sequence of knowledge-sound proof systems for

$$\Big\{ (x_0, x_{i+1}) \colon \exists (x_i, \rho_i, w_i, \sigma_i, w_{\sigma,i})\ F^{(i)}\,(par, (x_0, x_{i+1}), x_i, \rho_i, w_i) = \sigma_i \\ \text{and } (\sigma_i, w_{\sigma,i}) \in R_{F_1^{(i)}} \Big\}.$$

Main Predicate. The IVC proof computed at step i is a satisfiability proof for a predicate $G^{(i)}$. In addition to IVC input and output x_i and x_{i+1}, it takes as input the auxiliary pieces of information aux_i and aux_{i+1} computed at steps i and $i+1$. These pieces of information each include three commitments:

- a commitment C_1 to the concatenation of the outputs of F_0, of the input-outputs of $\text{VF}'_{\|,1}$ (and witnesses for their memberships to $L_{F_1^{(i)}}$ and $L_{\text{VF}'_{\|,1}}$) at all previous steps before the last one, and of an accumulation of $\text{VF}'_{\|,1}$ outputs up to two steps ago (together with a witness which attests that its accumulation with the output of $\text{VF}'_{\|,1}$ at the previous round results in the accumulation of all the $\text{VF}'_{\|,1}$ outputs up to the previous step),
- a commitment C_2 to the output, input-output and accumulation at the last step, and
- a commitment C_0 to the concatenation of the openings to C_1 and C_2, i.e., to the outputs, input-outputs and accumulation at all the previous steps.

The length of the concatenated table committed in C_1 is also given. It is necessary to ensure that the opening to C_0 is the concatenation of the openings to C_1 and C_2, because $\Pi_{\|}$ does so only if the length of the opening to C_1 has the length declared in the instance. Lastly, they contain a concatenation proof $\pi_{\|}$ as well as an accumulation T of the $\text{VF}'_{\|,1}$ outputs at all the steps before the previous one. The prover state consists of a piece of de-commitment information for C_0, of an opening A_1 to C_1 and a piece of de-commitment information r_1, and an opening $A_2 \coloneqq ((\sigma, w_\sigma), (\tau = (\tilde{\tau}, \bar{\tau}), w_\tau), (T', w_{\text{Acc}}))$ to C_2 and a piece of

de-commitment information r_2. That is to say,

$$\begin{aligned} aux_i &\coloneqq \left(C_0^{(i-1)}, \left(C_1^{(i-1)}, \ell_{i-1}\right), C_2^{(i-1)}, \pi_{\|,i}, T_{i-1}\right) \\ st_i &\coloneqq \left(r_0^{(i-1)}, A_1^{(i-1)}, r_1^{(i-1)}, A_2^{(i-1)}, r_2^{(i-1)}\right), \end{aligned}$$

and similarly for aux_{i+1} and st_{i+1}.

The predicate essentially checks consistency between commitments $C_0^{(i-1)}$ and $C_1^{(i)}$, guarantees that the output of the inexpensive part $F_0^{(i)}$ of predicate $F^{(i)}$ and the input-output pair of $\mathrm{VF}'_{\|,1}$ are indeed committed in $C_2^{(i-1)}$, that the new length ℓ_i is indeed the sum of ℓ_{i-1} and the length of $A_2^{(i-1)}$, that this new length does not exceed a bound $\ell_{\max}$ dictated by the parameters of the commitment scheme, that T' in $A_2^{(i)}$ is consistent with T_{i-1} in aux'_i, and that T_i is the correct accumulation of T_{i-1} and of the $\mathrm{VF}'_{\|,1}$ output.

What precedes only holds true for $i \geq i_0 + 4$. That is because there are no expensive operations before step i_0, and consequently no auxiliary proof information computed before step i_0, no concatenation proof computed before step $i_0 + 1$, no $\mathrm{VF}'_{\|,1}$ output before step $i_0 + 2$ and no accumulation of $\mathrm{VF}'_{\|,1}$ outputs before step $i_0 + 3$. In the first $i_0 + 4$ steps, the auxiliary proof information has the following form (the bracket notation indicates commitments):

$$\begin{aligned} st_1 = aux_1 = \cdots = st_{i_0} = aux_{i_0} &\leftarrow \varnothing \\ st_{i_0+1} &\leftarrow A_2^{(i_0)} = (\sigma_{i_0}, w_{\sigma,i_0}) =: A_0^{(i_0)}, r_2^{(i_0)} =: r_0^{(i_0)} \\ aux_{i_0+1} &\leftarrow C_0^{(i_0)} = C_2^{(i_0)} = \left[A_2^{(i_0)}; r_2^{(i_0)}\right] \\ st_{i_0+2} &\leftarrow r_0^{(i_0+1)}, A_1^{(i_0+1)} = A_0^{(i_0)}, r_0^{(i_0)}, A_2^{(i_0+1)} = (\sigma_{i_0+1}, w_{\sigma,i_0+1}), r_2^{(i_0+1)} \\ aux_{i_0+2} &\leftarrow \left[A_1^{(i_0+1)}, A_2^{(i_0+1)}; r_0^{(i_0+1)}\right], \left[A_2^{(i_0)}; r_2^{(i_0)}\right], \ell_{i_0+1}, \left[A_2^{(i_0+1)}; r_2^{(i_0+1)}\right] \\ &\quad \pi_{\|,i_0+2} \\ st_{i_0+3} &\leftarrow r_0^{(i_0+2)}, A_1^{(i_0+2)} = \left(A_1^{(i_0+1)}, A_2^{(i_0+1)}\right), r_0^{(i_0+1)} \\ &\quad A_2^{(i_0+2)} = ((\sigma_{i_0+2}, w_{\sigma,i_0+2}), (\tau_{i_0+2}, w_{\tau,i_0+2})), r_2^{(i_0+2)} \\ aux_{i_0+3} &\leftarrow \left[A_1^{(i_0+2)}, A_2^{(i_0+2)}; r_0^{(i_0+2)}\right], \left[A_1^{(i_0+2)}; r_0^{(i_0+1)}\right], \ell_{i_0+2}, \left[A_2^{(i_0+2)}; r_2^{(i_0+2)}\right] \\ &\quad \pi_{\|,i_0+3}, T_{i_0+2} \\ st_{i_0+4} &\leftarrow r_0^{(i_0+3)}, A_1^{(i_0+3)} = \left(A_1^{(i_0+2)}, A_2^{(i_0+2)}\right), r_0^{(i_0+2)} \\ &\quad A_2^{(i_0+3)} = ((\sigma_{i_0+3}, w_{\sigma,i_0+3}), (\tau_{i_0+3}, w_{\tau,i_0+3}), (T_{i_0+2}, w_{\mathrm{Acc},i_0+3})), \\ &\quad \pi_{\|,i_0+4}, T_{i_0+3}, r_2^{(i_0+3)} \\ aux_{i_0+4} &\leftarrow \left[A_1^{(i_0+3)}, A_2^{(i_0+3)}; r_0^{(i_0+3)}\right], \left[A_1^{(i_0+3)}; r_0^{(i_0+2)}\right], \ell_{i_0+3}, \left[A_2^{(i_0+3)}; r_2^{(i_0+3)}\right] \\ &\quad \pi_{\|,i_0+4}, T_{i_0+3}. \end{aligned}$$

For $i \geq i_0 + 3$, define

$G^{(i)}\left(par, (x_0, x_{i+1}, aux_{i+1}), A_2^{(i)}, r_2^{(i)}, x_i, \rho_i, aux_i, A_2^{(i-1)}, r_2^{(i-1)}, w_i\right) \to \{0,1\}$:

Return 1 if

$$
\begin{aligned}
&F_0^{(i)}\left(par, (x_0, x_{i+1}), x_i, \rho_i, w_i\right) = \sigma_i \\
&\Gamma.\textsc{Open}\left(C_2^{(i-1)}, A_2^{(i-1)}, r_2^{(i-1)}\right) = 1 \\
&\Gamma.\textsc{Open}\left(C_2^{(i)}, A_2^{(i)}, r_2^{(i)}\right) = 1 \\
&C_0^{(i-1)} = C_1^{(i)} \\
&\mathrm{VF}_{\|,0}\left(par_{\|}, \left(C_0^{(i-1)}, \left(C_1^{(i-1)}, \ell_{i-1)}\right), C_2^{(i-1)}\right), \pi_{\|,i}\right) = \tilde{\tau}_i \\
&\ell_i = \ell_{i-1} + \mathrm{length}\left(A_2^{(i-1)}\right) \\
&\ell_i \leq \ell_{\max} \\
&T_{i-1} = A_2^{(i)}.T_i'
\end{aligned}
$$

Else return 0,

with

$$A_2^{(i)} \coloneqq ((\sigma_i, w_{\sigma,i}), (\tau_i = (\tilde{\tau}_i, \bar{\tau}_i), w_{\tau,i}), (T_i', w_{\mathrm{ACC},i}))\,.$$

The second and third conditions in the predicate are easily proved in the Plonk arithmetisation: constrain the inputs of $G^{(i)}$ corresponding to $A_2^{(i-1)}$ to be equal to the values committed in $C_2^{(i-1)}$; the latter is interpreted as a commitment to extra columns of a Plonk execution trace. The same can be done for $A_2^{(i)}$ and $C_2^{(i)}$.

For $i < i_0$, function $G^{(i)}$ returns 1 if and only if $F^{(i)}$ does so on the same inputs. Function $G^{(i_0)}$ ensures that the output of $F_0^{(i_0)}$ is committed in $C_1^{(i_0)}$. $G^{(i_0+1)}$ additionally enforces consistency between $C_0^{(i_0)}$ and $C_1^{(i_0+1)}$, and that ℓ_{i_0+1} in aux_{i_0+2} matches the length of $A_2^{(i_0)}$ in aux_{i_0+1} (this latter check is crucial to later ensure correct concatenation). Function $G^{(i_0+2)}$ further guarantees that the output of $F_0^{(i_0+2)}$ and the input-output of $\mathrm{VF}'_{\|,1}$ are committed in $C_2^{(i_0+2)}$, that the ℓ_{i_0+2} in aux_{i_0+3} is the sum of ℓ_{i_0+1} and of the length of $A_2^{(i_0+1)}$ in aux_{i_0+2}, and that T_{i_0+2} is the output of $\mathrm{VF}'_{\|,1}$.

A precise definition of $G^{(i)}$ would require its input set to be completely specified. Yet, vectors $A_2^{(i-1)}$ and $A_2^{(i)}$ that it receives as input might in practice be of variable size. For a given $\ell_{\max}$ in the *par* input of $G^{(i)}$, the set to which $A_2^{(i-1)}$ and $A_2^{(i)}$ respectively belong is simply defined as the union of the sets of vectors of size 1 to $\ell_{\max}$. The input set of $G^{(i)}$ is then defined as the union over $\ell_{\max} \in \mathbb{N}_{\geq 1}$ of the input set for a given $\ell_{\max}$.

Now that the sequence of predicates $G^{(i)}$ has been defined, the construction will also make use of the following primitives.

$\left(P_0^{(i)}, V_0^{(i+1)}\right)_{i\in\mathbb{N}}$: a sequence of knowledge-sound proof systems for the inexpensive parts of functions $\left(F^{(i)}\right)_{i\in\mathbb{N}}$, i.e., for

$$\left\{(x_0, x_{i+1})\colon \exists(x_i, \rho_i, w_i, \sigma_i)\ F_0^{(i)}\left(par, (x_0, x_{i+1}), x_i, \rho_i, w_i\right) = \sigma_i\right\}$$

for all $i \geq i_0$. This is easily obtained from the previous sequence by simply removing the constraints to enforce that $(\sigma_i, w_{\sigma,i}) \in R_{F_1^{(i)}} = 1$. In the Plonk arithmetisation for instance, it simply means setting the selectors for these constraints to 0. Let $\left(P_0^{(i)}, V_0^{(i+1)}\right) = \left(P^{(i)}, V^{(i+1)}\right)$ for all $i < i_0$. This assumption is the reason why the construction is not entirely black-box, though it holds for known practical applications.

$\left(Q^{(i)}, W^{(i+1)}\right)_{i\in\mathbb{N}}$: a sequence of knowledge-sound proof systems for the languages of instances accepted by functions $\left(G^{(i)}\right)_{i\in\mathbb{N}}$, i.e., for

$$\left\{ (x_0, x_{i+1}, aux_{i+1})\colon \exists \left(A_2^{(i)}, r_2^{(i)}, x_i, \rho_i, aux_i, A_2^{(i-1)}, r_2^{(i-1)}, w_i\right) \right.$$
$$\left. G^{(i)}\left(par, (x_0, x_{i+1}, aux_{i+1}), A_2^{(i)}, r_2^{(i)}, x_i, \rho_i, aux_i, A_2^{(i-1)}, r_2^{(i-1)}, w_i\right) = 1 \right\}.$$

This sequence can be obtained from sequence $\left(P_0^{(i)}, V_0^{(i+1)}\right)_{i\in\mathbb{N}}$ by adding extra constraints to enforce the other checks performed by functions $\left(G^{(i)}\right)_{i\geq i_0}$. (For $i < i_0$, define $\left(Q^{(i)}, W^{(i+1)}\right) := \left(P^{(i)}, V^{(i+1)}\right)$.) Similarly to the case of sequence $\left(P^{(i)}, V^{(i+1)}\right)_{i\in\mathbb{N}}$, it is can be assumed that the proof systems in sequence $\left(Q^{(i)}, W^{(i+1)}\right)_{i\in\mathbb{N}}$ share the same algorithm to generate parameters which is denoted (Q, W).Setup, and that the proving and verification algorithms are run on the same parameters throughout the sequence. In practice, these algorithms and parameters are those of a universal SNARK system or of a folding scheme, and are the same as those for $\left(P^{(i)}, V^{(i+1)}\right)_{i\in\mathbb{N}}$. It is further assumed that it is the case, and functions $F^{(i)}$ can thus be run on parameters generated by (Q, W).Setup.

Formal Description. Given the building blocks, the following scheme is an IVC scheme with auxiliary proof information for a generator G that runs R and returns its output together with a description of relations

$$R_{aux}^{(1)} = \cdots = R_{aux}^{(i_0)} \leftarrow \{(\varnothing, \varnothing)\},$$

$$R_{aux}^{(i_0+1)} \leftarrow \Big\{((C_0, C_2), (A_2, r_2)) : C_0 = C_2,\ A_2 \in R_{F_1^{(i_0)}} \text{ and } \Gamma.\textsc{Open}\left(C_2, A_2, r_2\right) = 1\Big\},$$

$$R_{aux}^{(i_0+2)} \leftarrow \Big\{ \big((C_0,(C_1,\ell)C_2,\pi_\parallel),(r_0,A_1,r_1,A_2,r_2)\big) : A_1 \in R_{F_1^{(i_0)}} \\ \Gamma.\textsc{Open}\,(C_1,A_1,r_1)=1 \\ A_2 \in R_{F_1^{(i_0+1)}} \\ \Gamma.\textsc{Open}\,(C_2,A_2,r_2)=1 \\ \ell=\mathrm{length}(A_1) \\ \text{and } \mathrm{V}_{\mathrm{F}\parallel}\left(par_\parallel,(C_0,(C_1,\ell),C_2),\pi_\parallel\right)=1 \Big\},$$

$$R_{aux}^{(i_0+3)} \leftarrow \Big\{ \big((C_0,C_1,C_2,\pi_\parallel,T),(r_0,A_1,r_1,A_2 \coloneqq ((\sigma,w_\sigma),(\tau,w_\tau)),r_2)\big) : \\ A_1 \in R_{F_1^{(i_0+1)}} \times R_{F_1^{(i_0)}} \\ \Gamma.\textsc{Open}\,(C_1,A_1,r_1)=1 \\ (\sigma,w_\sigma) \in R_{F_1^{(i-1)}},\ (\tau,w_\tau) \in R_{\mathrm{V}_{\mathrm{F}'\parallel,1}} \\ \Gamma.\textsc{Open}\,(C_2,A_2,r_2)=1 \\ \ell=\mathrm{length}(A_1) \\ \mathrm{V}_{\mathrm{F}\parallel}\left(par_\parallel,(C_0,(C_1,\ell),C_2),\pi_\parallel\right)=1 \\ \text{and } T \in L_{\mathrm{V}_{\mathrm{F}''\parallel,1}} \Big\},$$

and for $i \geq i_0+4$,

$$R_{aux}^{(i)} \leftarrow \Big\{ \big((C_0,C_1,C_2,\pi_\parallel,T),(r_0,A_1,r_1,A_2 \coloneqq ((\sigma,w_\sigma),(\tau,w_\tau),(T',w_{\mathrm{Acc}})),r_2)\big) : \\ A_1 \in R_{A_1^{(i-2)}} \\ \Gamma.\textsc{Open}\,(C_1,A_1,r_1)=1 \\ (\sigma,w_\sigma) \in R_{F_1^{(i-1)}},\ (\tau,w_\tau) \in R_{\mathrm{V}_{\mathrm{F}'\parallel,1}},\ (((T',\bar\tau),T),w_{\mathrm{Acc}}) \in R_{\mathrm{Acc}} \\ \Gamma.\textsc{Open}\,(C_2,A_2,r_2)=1 \\ \ell=\mathrm{length}(A_1) \\ \mathrm{V}_{\mathrm{F}\parallel}\left(par_\parallel,(C_0,(C_1,\ell),C_2),\pi_\parallel\right)=1 \\ \text{and } T \in L_{\mathrm{V}_{\mathrm{F}''\parallel,1}} \Big\},$$

with $\tau=(\tilde\tau,\bar\tau)$,

$$R'_{\mathrm{Acc}} \coloneqq \{(T',w_{\mathrm{Acc}})\colon (((T',\bar\tau),T),w_{\mathrm{Acc}}) \in R_{\mathrm{Acc}}\}$$

(which is parametrised by $\bar\tau$ and T), and

$$R_{A_1^{(i-2)}} = R_{F_1^{(i_0)}} \times R_{F_1^{(i_0+1)}} \times \left(R_{\mathrm{V}_{\mathrm{F}'\parallel,1}} \times R_{F_1^{(i_0+2)}} \times R'_{\mathrm{Acc}}\right) \times \\ \cdots \times \left(R_{\mathrm{V}_{\mathrm{F}'\parallel,1}} \times R_{F_1^{(i-2)}} \times R'_{\mathrm{Acc}}\right).$$

It is assumed that the security parameter 1^λ can be inferred from the output of generator R.

The scheme algorithms are as follows. They are assumed to abort if any of their inputs is not in their specified sets.

$\textbf{Setup}\left(R, \left(R_{aux}^{(i)}\right)_{i\geq 1}, \ell_{\max}\right) \to par:$

$par_\Gamma \leftarrow \Gamma.\textsc{Setup}\left(1^\lambda, \ell_{\max}\right)$

$par_\parallel \leftarrow \Pi_\parallel.\textsc{Setup}\left(1^\lambda, \ell_{\max}\right)$

$par_{(Q,W)} \leftarrow (Q,W).\textsc{Setup}\left(R\right)$

$par \leftarrow \left(par_\Gamma, par_\parallel, par_{(Q,W)}, \left(R_{aux}^{(i)}\right)_{i\geq 1}\right)$

Return par

$\textbf{Prove}(par, (i, x_0, x_{i+1}), x_i, \pi_i, aux_i, st_i, w_i) \to (\pi_{i+1}, aux_{i+1}, st_{i+1}):$

Compute ρ_i from $\left(par_{(Q,W)}, x_0, x_i, \pi_i\right)$

$\sigma_i \leftarrow F_0^{(i)}\left(par_{(Q,W)}, (x_0, x_{i+1}), x_i, \rho_i, w_i\right)$

Compute $w_{\sigma,i}$ such that $(\sigma_i, w_{\sigma,i}) \in R_{F_1^{(i)}}$

$st_i \coloneqq \left(r_0^{(i-1)}, A_1^{(i-1)}, r_1^{(i-1)}, A_2^{(i-1)}, r_2^{(i-1)}\right)$

$aux_i \coloneqq \left(C_0^{(i-1)}, \left(C_1^{(i-1)}, \ell_{i-1}\right), C_2^{(i-1)}, \pi_{\parallel,i}, T_{i-1}\right)$

$x_{\parallel,i} \leftarrow \left(C_0^{(i-1)}, \left(C_1^{(i-1)}, \ell_{i-1}\right), C_2^{(i-1)}\right)$

$\tilde{\tau}_i \leftarrow \mathrm{V}_{\mathrm{F}\parallel,0}\left(par_\parallel, x_{\parallel,i}, \pi_{\parallel,i}\right)$

$\bar{\tau}_i \leftarrow \mathrm{V}_{\mathrm{F}\parallel,1}'\left(\tilde{\tau}_i\right)$

Compute $w_{\tau,i}$ such that $(\tau_i \coloneqq (\tilde{\tau}_i, \bar{\tau}_i), w_{\tau,i}) \in R_{\mathrm{V}_{\mathrm{F}\parallel,1}'}$

$T_i \leftarrow \textsc{Acc}\left(T_{i-1}, \bar{\tau}_i; H\left(T_{i-1}, \bar{\tau}_i\right)\right)$

Compute $w_{\textsc{Acc},i}$ such that $(((T_{i-1}, \bar{\tau}_i), T_i), w_{\textsc{Acc},i}) \in R_{\textsc{Acc}}$

$A_2^{(i)} \leftarrow ((\sigma_i, w_{\sigma,i}), (\tau_i, w_{\tau,i}), (T_{i-1}, w_{\textsc{Acc},i}))$

$C_2^{(i)} \leftarrow \Gamma.\textsc{Com}\left(A_2^{(i)}, r_2^{(i)}\right)$ for a uniformly random $r_2^{(i)}$

$A_1^{(i)}, C_1^{(i)} \leftarrow A_0^{(i-1)} \coloneqq A_1^{(i-1)} \parallel A_2^{(i-1)}, C_0^{(i-1)}$

$C_0^{(i)} \leftarrow \Gamma.\textsc{Com}\left(A_1^{(i)} \parallel A_2^{(i)}, r_0^{(i)}\right)$ for a uniformly random $r_0^{(i)}$

$\ell_i \leftarrow \mathrm{length}\left(A_1^{(i)}\right)$

$\pi_{\|,i+1} \leftarrow \textsc{Prove}_{\|}\left(par_{\|}, \left(C_0^{(i)}, \left(C_1^{(i)}, \ell_i\right), C_2^{(i)}\right), \left(A_1^{(i)} \parallel A_2^{(i)}, A_1^{(i)}, A_2^{(i)}\right)\right)$

$st_{i+1} \leftarrow \left(r_0^{(i)}, A_1^{(i)}, r_0^{(i-1)}, A_2^{(i)}, r_2^{(i)}\right)$

$aux_{i+1} \leftarrow \left(C_0^{(i)}, \left(C_1^{(i)}, \ell_i\right), C_2^{(i)}, \pi_{\|,i+1}, T_i\right)$

$x_{Q,W} \leftarrow (x_0, x_{i+1}, aux_{i+1})$

$w_{Q,W} \leftarrow \left(A_2^{(i)}, r_2^{(i)}, x_i, \rho_i, aux_i, A_2^{(i-1)}, r_2^{(i-1)}, w_i\right)$

$\pi_{i+1} \leftarrow Q^{(i)}\left(par_{Q,W}, x_{Q,W}, w_{Q,W}\right)$

Return $(\pi_{i+1}, aux_{i+1}, st_{i+1})$

$\mathbf{V_F}(par, (i+1, x_0, x_{i+1}), \pi_{i+1}, aux_{i+1}) \rightarrow \{0,1\}$:

Return $W^{(i+1)}\left(par_{Q,W}, (x_0, x_{i+1}, aux_{i+1}), \pi_{i+1}\right)$.

The above definitions of the prover and verifier algorithms are given for the case $i \geq i_0 + 3$. For $i \leq i_0 + 2$, they simply do not perform the operations which are undefined, given the definitions of aux_i and $G^{(i)}$.

Notice that the work of the verifier is essentially the work of $W^{(i+1)}$ which verifiers proofs for a circuit which only takes the input-outputs $A_2^{(i)}$ and $A_2^{(i-1)}$ at the current and previous steps, not a whole accumulator.

Completeness. Suppose that $\left(Q^{(i)}, W^{(i+1)}\right)$ is complete for all $i \in \mathbb{N}$ and that $\Pi_{\|}$ is complete. Assume Γ to be correct. Then, the scheme is complete (as formally defined in Appendix B.1).

For $i = 0$, given any $((x_0, x_1), w_0)$ in R, by definition, function $F^{(i)}$ accepts on $(par, (0, x_0, x_1), x_0, w_0)$. As the prover algorithm simply runs $Q^{(0)}$, which proves that $G^{(0)} = F^{(0)}$ would accept on $(par, (x_0, x_1), x_0, w_0)$, the verifier algorithm, which runs $W^{(1)}$, accepts the proof. Moreover, $(aux_1, st_1) = (\varnothing, \varnothing) \in R_{aux}^{(1)}$.

For $0 < i < i_0$ (in case $i_0 > 1$), let $(i, x_0, x_i, \pi_i, aux_i)$ be such that $\textsc{Vf}$ accepts on $(par, (i, x_0, x_i), \pi_i, aux_i)$. Algorithm $\textsc{Vf}$ accepts if and only if $W^{(i)} = V^{(i)}$ accepts on $(par, (x_0, x_i), \pi_i)$ and $aux_i = \varnothing$. Given x_{i+1} and w_i such that the tuple $((x_i, x_{i+1}), w_i)$ is in R, by definition, $G^{(i)} = F^{(i)}$ accepts on input $(par_{Q,W}, (x_0, x_{i+1}), x_i, \rho_i, w_i)$ if ρ_i is correctly computed from the tuple $(par_{Q,W}, x_0, x_i, \pi_i)$. As the prover does just that before running $Q^{(i)}$ which shows that $G^{(i)}$ accepts, the verifier, which runs $W^{(i+1)}$, accepts the proof computed by $Q^{(i)}$. Besides, the prover sets $aux_{i+1} \leftarrow \varnothing$, so it is in $L_{aux}^{(i+1)}$.

For $i = i_0$, let $(i_0, x_0, x_{i_0}, \pi_{i_0}, aux_{i_0})$ be such that $\textsc{Vf}$ accepts on input $(par, (i_0, x_0, x_{i_0}), \pi_{i_0})$. It means that $W^{(i_0)} = V^{(i_0)}$ accepts on $(par, x_0, x_{i_0}, \pi_{i_0})$ and that $aux_{i_0} = \varnothing$. Let x_{i_0+1} and w_{i_0} be such that $((x_{i_0}, x_{i_0+1}), w_{i_0})$ is in R. The prover algorithm computes $\rho_{i,0}$ from $(par, x_0, x_{i_0}, \pi_{i_0})$ and computes the output σ_{i_0} of $F_0^{(i_0)}$ on input $(par, (x_0, x_{i_0+1}), x_{i_0}, \rho_{i_0}, w_{i_0})$. It then commits to $(\sigma_{i_0}, w_{\sigma,i_0})$ in $C_2^{(i_0)}$ and sets $C_0^{(i_0)} \leftarrow C_2^{(i_0)}$. If Γ is complete, its opening algorithm would accept on $C_2^{(i_0)}$ and $A_2^{(i_0)}$, and $W^{(i_0+1)}$ would therefore accept the proof

computed by $Q^{(i_0)}$. Moreover, as $F^{(i_0)}$ accepts on $(par, (x_0, x_{i_0+1}), x_{i_0}, \rho_{i_0}, w_{i_0})$, the instance-witness pair $A_2^{(i_0)}$ is in $R_{F_1^{(i_0)}}$ and therefore $(aux_{i_0+1}, st_{i_0+1}) \in R_{aux}^{(i_0+1)}$.

For $i \in \{i_0+1, i_0+2\}$, the correctness of Γ and the completeness of $\Pi_\parallel$ in addition to arguments similar to those given in the previous case guarantee that the verifier accepts proofs computed by the prover, and that $(aux_{i+1}, st_{i+1} \in R_{aux}^{(i+1)}$ if $(aux_i, st_i) \in R_{aux}^{(i)}$.

For $i \geq i_0 + 3$, the fact that $\textsc{Acc}(T_{i-1}, \bar{\tau}_i)$ is in $L_{\mathrm{V_F}''_{\parallel,1}}$ if T_{i-1} and $\bar{\tau}_i$ are, in addition to the previous arguments, implies that the verifier accepts proofs computed by the prover, and that $aux_{i+1} \in L_{aux}^{(i+1)}$ if $aux_i \in L_{aux}^{(i)}$. □

Knowledge Soundness. Suppose that $\left(Q^{(i)}, W^{(i+1)}\right)$ is knowledge-sound for all $i \in \mathbb{N}$ and that $\Pi_\parallel$ is knowledge-sound. Suppose that for all $i \in \mathbb{N}$, a proof π_i can be extracted from any PPT algorithm which computes a valid proof π_{i+1} and $aux_{i+1} \in L_{aux}^{(i+1)}$ with probability at least $\kappa_\rho^{(i+1)}$. Assume Γ to be ε_Γ-binding and Acc to have an error probability of $\varepsilon_{\textsc{Acc}}^H$ when its randomness is computed as a hash of its inputs with a function H modelled as a random oracle. Then, the scheme is knowledge-sound (as formally defined in Appendix B.1).

An error from algorithm Acc is here understood as the event in which a probabilistic algorithm with runs in expected polynomial time returns a tuple $(T', \bar{\tau}, T)$ such that T is in $L_{\mathrm{V_F}''_\parallel}$ but $\bar{\tau}$ or T' is not. The error probability may depend on the probability that this algorithm returns such a triple with T in $L_{\mathrm{V_F}''_\parallel}$, and on the number of random-oracle queries it makes. See Section 5.4 for an instantiation of Acc and its error probability.

Let A be an algorithm which returns a tuple $(st, (n+1, x_0, x_n))$ with n non-negative and constant in λ, and let $\textsc{Prove}^*$ be a deterministic algorithm such that, denoting by (π^*, aux^*) its output on input $(st, (n+1, x_0, x_{n+1}))$, the piece of information aux^* is in $L_{aux}^{(n+1)}$ and algorithm $\mathrm{V_F}$ accepts (π^*, aux^*) for the instance $(n+1, x_0, x_{n+1})$ with probability at least ε. Let q_H be the number of queries that $\textsc{Prove}^*$ makes to the random oracle H.

To extract sequences of IVC outputs and witnesses from algorithm $\textsc{Prove}^*$, the idea is to inductively define a sequence of extractors $\mathrm{E}^{(i+1)}$ for all $i \in \mathbb{N}$. Intuitively, extractor $\mathrm{E}^{(i+1)}$ is designed to extract sequences $x_1, \ldots, x_i$ (if $i \geq 1$) and $w_0, \ldots, w_i$ such that $((x_0, x_1), w_1), \ldots, ((x_i, x_{i+1}), w_i)$ are in R, given rewinding access to a prover which computes a proof and an auxiliary piece of information $aux_{i+1} \in L_{aux}^{(i+1)}$ that the verifier accepts. Each extractor calls the previous one on an input and on a prover that is determined by its own input and prover to which it has rewinding access. In addition to that, each extractor is so that if the auxiliary information computed by the prover to which it has access is valid, then the auxiliary information computed by the prover to which it gives the previous extractor access is also valid. Extractor E then simply runs $\mathrm{E}^{(n+1)}$ on input (x_0, x_{n+1}) and with rewinding access to $\textsc{Prove}^*(st, n, \cdot)$.

For all $i \in \mathbb{N}$, let $\mathrm{E}_{Q,W}^{(i+1)}$ denote an extractor for proof system $\left(Q^{(i)}, W^{(i+1)}\right)$. In the case of SNARK bootstrapping, it is an extractor for the universal SNARK,

applied to the circuit for which $W^{(i+1)}$ verifies satisfiability proofs. In the case of the Nova IVC scheme, it is an algorithm which computes (x_i, ρ_i, w_i) from the witness v_{i+1} for the instance u_{i+1} which is part of the IVC proof that $W^{(i+1)}$ verifies. Let $\mathrm{E}_{\|}$ be an extractor for $\Pi_{\|}$. Denote by $\mathrm{E}_{\rho}^{(i+1)}$ the algorithm which extracts π_i from any PPT algorithm which computes a valid proof π_{i+1} and on an instance determined by ρ_i. In the case of SNARK bootstrapping, it is the algorithm which simply returns ρ_i. In the case of the NOVA IVC, it is an extractor for the folding scheme.

Extractor $\mathrm{E}^{(1)}$ is defined below for any probabilistic algorithm $(P^*)^{(0)}$ that runs in expected polynomial time and to which the extractor is given black-box rewinding access (but without access to its state output). To define $\mathrm{E}^{(i+1)}$ for a positive integer i, consider any probabilistic algorithm $(P^*)^{(i)}$ that runs in expected polynomial time, of which the output is parsed as $(\mathit{\Pi}_{i+1}, AUX_{i+1}, ST_{i+1})$, and to which $\mathrm{E}^{(i+1)}$ is given black-box rewinding access (but without access to its ST_{i+1} output). The algorithm is here assumed to run in expected polynomial time rather than strict polynomial time because the algorithms to follow run proof-system extractors, which are generally only guaranteed to run in expected polynomial time, c.f. Appendix A.1.2. Intuitively, it is an adversarial prover the IVC step i. As extractor $\mathrm{E}^{(i+1)}$ will run extractor $\mathrm{E}^{(i)}$ (assuming it to already be defined), a prover $(P^*)^{(i-1)}$ for step $i-1$ must be defined.

The first step is to define an algorithm $(Q^*)^{(i-1)}$ (given rewinding access to $(P^*)^{(i)}$) which computes a witness tuple $\left(A_2^{(i)}, r_2^{(i)}, x_i, \pi_i, aux_i, A_2^{(i-1)}, r_2^{(i-1)}, w_i\right)$. It runs on an instance $(x_0, x_{i+1}, aux_{i+1})$ with a *fixed* value aux_{i+1}. Assuming aux_{i+1} to be in $L_{aux}^{(i+1)}$, and denoting by $\mathit{\Pi}_{i+1|AUX_{i+1}=aux_{i+1}}$ the first output of $(P^*)^{(i)}$ conditioned on the event in which its second output is the fixed valued aux_{i+1}, if the verifier accepts $(\mathit{\Pi}_{i+1|AUX_{i+1}=aux_{i+1}}, aux_{i+1})$ for the instance $(i+1, x_0, x_{i+1})$ with a probability greater than the knowledge-soundness error threshold of $\left(Q^{(i)}, W^{(i+1)}\right)$ and greater than $\kappa_{\rho}^{(i+1)}$, then the tuple that $(Q^*)^{(i-1)}$ returns is such that

$$G^{(i)}\left(par, (x_0, x_{i+1}, aux_{i+1}), A_2^{(i)}, r_2^{(i)}, x_i, \rho_i, aux_i, A_2^{(i-1)}, r_2^{(i-1)}, w_i\right) = 1,$$

for ρ_i computed from (par, x_0, x_i, π_i).

$\left((Q^*)^{(i-1)}\right)^{(P^*)^{(i)}}\left(par_{Q,W}, (x_0, x_{i+1}, aux_{i+1})\right)$:

$\left(A_2^{(i)}, r_2^{(i)}, x_i, \rho_i, aux_i, A_2^{(i-1)}, r_2^{(i-1)}, w_i\right) \leftarrow \left(\mathrm{E}_{Q,W}^{(i+1)}\right)^{(P^*)^{(i)}}$

$\pi_i \leftarrow \left(\mathrm{E}_{\rho}^{(i+1)}\right)^{(P^*)^{(i)}}\left(par_{Q,W}, \rho_i\right)$

Return $\left(A_2^{(i)}, r_2^{(i)}, x_i, \pi_i, aux_i, A_2^{(i-1)}, r_2^{(i-1)}, w_i\right)$.

Extractor $\mathrm{E}_{Q,W}^{(i+1)}$ is here run on input $\left(par_{Q,W}, (x_0, x_{i+1}, aux_{i+1})\right)$. The piece of information aux_i that it returns is in $L_{aux}^{(i)}$ if the fixed value aux_{i+1} is in $L_{aux}^{(i+1)}$.

Given the output of $(Q^*)^{(i-1)}$, the next step is to define a prover algorithm $(P^*)^{(i-1)}$ for the step $i-1$, i.e., an algorithm which computes a state st_i that is a valid witness for the validity of the piece of information aux_i auxiliary to π_i. In other words, this algorithm reconstructs from aux_i a state st_i such that $(aux_i, st_i) \in R_{aux}^{(i)}$. However, $E^{(i)}$ is not given access to this state output st_i.

$\left((P^*)^{(i-1)}\right)^{(P^*)^{(i)}} \left(par, (x_0, x_{i+1}, aux_{i+1}), A_2^{(i)}, r_2^{(i)}, x_i, \pi_i, aux_i, A_2^{(i-1)}, r_2^{(i-1)}\right)$:

If $i \geq i_0 + 2$

$x_{\|} \leftarrow \left(C_0^{(i-1)}, \left(C_1^{(i-1)}, \ell_{i-1}\right), C_2^{(i-1)}\right)$

$(A_k, r_k)_{k=0}^2 \leftarrow \mathrm{E}_{\|}^{(Q^*)^{(i-1)}} \left(par_{\|}, x_{\|}\right)$

Return $\varnothing$ if $A_2 \neq A_2^{(i-1)}$

$r_0^{(i-1)}, A_1^{(i-1)}, r_1^{(i-1)} \leftarrow r_0, A_1, r_1$

If $i \geq i_0 + 3$

$aux_i =: \left(C_0^{(i-1)}, \left(C_1^{(i-1)}, \ell_{i-1}\right), C_2^{(i-1)}, \pi_{\|,i}, T_{i-1}\right)$

If $i = i_0 + 3$

$A_2^{(i)} \coloneqq ((\tau_i \coloneqq (\tilde{\tau}_i, \bar{\tau}_i), w_{\tau,i}), (\sigma_i, w_{\sigma,i}))$

Else

$A_2^{(i)} \coloneqq ((\tau_i \coloneqq (\tilde{\tau}_i, \bar{\tau}_i), w_{\tau,i}), (\sigma_i, w_{\sigma,i}), (T_i', w_{\mathrm{Acc},i}))$

Return $\varnothing$ if T_{i-1} or $\bar{\tau}_i$ is not in $L_{\mathrm{VF}''_{\|,1}}$

$st_i \leftarrow \left(r_0^{(i-1)}, A_1^{(i-1)}, r_1^{(i-1)}, A_2^{(i-1)}, r_2^{(i-1)}\right)$

Return $(x_i, \pi_i, aux_i, st_i)$.

Suppose that aux_{i+1} is in $L_{aux}^{(i+1)}$ and that the verifier accepts on

$$(par, (i+1, x_0, x_{i+1}), \Pi_{i+1|AUX_{i+1}=aux_{i+1}}, aux_{i+1})$$

with a probability greater than the soundness-error threshold of $\left(Q^{(i)}, W^{(i+1)}\right)$ and greater than $\kappa_\rho^{(i+1)}$. As the tuple algorithm $(Q^*)^{(i-1)}$ returns is such that

$$G^{(i)}\left(par, (x_0, x_{i+1}, aux_{i+1}), A_2^{(i)}, r_2^{(i)}, x_i, \rho_i, aux_i, A_2^{(i-1)}, r_2^{(i-1)}, w_i\right) = 1,$$

σ_i is the output of $F_0^{(i)}$ on input $(par, (x_0, x_{i+1}), x_i, \rho_i, w_i)$. Moreover, as aux_{i+1} is in $L_{aux}^{(i+1)}$ and as $A_2 = A_2^{(i-1)}$ if $(P^*)^{(i-1)}$ does not abort, function $F_1^{(i)}$ accepts on input σ_i, and therefore $F^{(i)}$ accepts on input $(par, (x_0, x_{i+1}), x_i, \rho_i, w_i)$.

The piece of information aux_{i+1} being in $L_{aux}^{(i+1)}$ and the fact that $A_2 = A_2^{(i-1)}$ if $(P^*)^{(i-1)}$ does not abort also imply that $\tau_i = (\tilde{\tau}_i, \bar{\tau}_i)$ is a valid input-output of $\mathrm{VF}'_{\|,1}$ and that $\mathrm{VF}''_{\|,1}$ accepts on T_i. If $i \geq i_0 + 3$, the latter is the output of algorithm Acc on input $(T_{i-1}, \bar{\tau}_i)$ because $w_{\mathrm{Acc},i}$ is a valid witness for the membership of $((T'_i, \bar{\tau}_i), T_i)$ to L_{Acc} (the language corresponding to R_{Acc}), and because the definition of $G^{(i)}$ implies that $T_{i-1} = A_2^{(i)}.T'_i$. If $(P^*)^{(i-1)}$ does not abort, T_{i-1} and $\bar{\tau}_i$ are both in the language of inputs that $\mathrm{VF}''_{\|,1}$ accepts. Therefore, by definition of $G^{(i)}$, algorithm $\mathrm{VF}_{\|}$ accepts $\pi_{\|,i}$ for the instance $\left(C_0^{(i-1)}, \left(C_1^{(i-1)}, \ell_{i-1}\right), C_2^{(i-1)}\right)$.

It remains to show that there exists an opening to $C_1^{(i-1)}$ that is in $R_{A_1^{(i-2)}}$ and that has length ℓ_{i-1}. The equality $C_1^{(i)} = C_0^{(i-1)}$ holds because $G^{(i)}$ accepts, and there exists an opening to $C_1^{(i)}$ that is in $R_{A_1^{(i-1)}}$ and that has length ℓ_i because $aux^{(i+1)}$ is in $L_{aux}^{(i+1)}$. Algorithm $(P^*)^{(i-1)}$ runs the extractor $\mathrm{E}_{\|}$ of $\Pi_{\|}$ to extract openings to $C_0^{(i-1)}$, $C_1^{(i-1)}$ and $C_2^{(i-1)}$. If the opening to $C_2^{(i-1)}$ is distinct from $A_2^{(i-1)}$, the algorithm aborts as the binding property of Γ has been contradicted. Otherwise, since the length ℓ_{i-1} (in case $i \geq i_0 + 2$) on which $\mathrm{VF}_{\|}$ is run is equal to $\ell_i - \mathrm{length}\left(A_2^{(i-1)}\right)$ by definition of $G^{(i)}$, that opening to $C_0^{(i-1)}$ is the concatenation of an opening to $C_1^{(i-1)}$ and $A_2^{(i-1)}$. That is also because $G^{(i)}$ checks that $\ell_i \leq \ell_{\max}$. As $R_{A_1^{(i-1)}} = R_{A_1^{(i-2)}} \times \left(R_{\mathrm{VF}'_{\|,1}} \times R_{F_1^{(i-1)}} \times R'_{\mathrm{Acc}}\right)$, that opening to $C_1^{(i-1)}$ is in $R_{A_1^{(i-2)}}$, and aux_i is thus in $L_{aux}^{(i)}$.

Given these algorithms, a sequence of extractors $\left(\mathrm{E}^{(i+1)}\right)_{i\in\mathbb{N}}$ can now be inductively defined as follows. Extractor $\mathrm{E}^{(i+1)}$ mainly runs $(Q^*)^{(i-1)}$ to extract an IVC output x_i and a witness w_i, and runs $\mathrm{E}^{(i)}$ with rewinding access to $(P^*)^{(i-1)}$ to extract the rest of the intermediate IVC outputs and witnesses.

$\left(\mathbf{E}^{(1)}\right)^{(P^*)^{(0)}}(par, (x_0, x_1))$:

$w_0 \leftarrow \left(\mathrm{E}_{Q,W}^{(1)}\right)^{(P^*)^{(0)}}\left(par_{Q,W}, (x_0, x_1)\right)$

Return w_0

and for $i > 0$,

$\left(\mathbf{E}^{(i+1)}\right)^{(P^*)^{(i)}}(par, (x_0, x_{i+1}))$:

$(\pi_{i+1}, aux_{i+1}, st_{i+1}) \leftarrow (P^*)^{(i)}(x_0, x_{i+1})$

Return $\varnothing$ if $\mathrm{VF}(par, (i+1, x_0, x_{i+1}), \pi_{i+1}, aux_{i+1}) = 0$

$$\left(A_2^{(i)}, r_2^{(i)}, x_i, \pi_i, aux_i, A_2^{(i-1)}, r_2^{(i-1)}, w_i\right) \leftarrow \left((Q^*)^{(i-1)}\right)^{(P^*)^{(i)}}$$

$$(w_0, x_1, w_1, \ldots, x_{i-1}, w_{i-1}) \leftarrow \left(\mathrm{E}^{(i)}\right)^{(P^*)^{(i-1)}}(par, (x_0, x_i))$$

Return $(w_0, x_1, w_1, \ldots, x_i, w_i)$.

The extractors abort if any of the algorithms they run as subroutines does.

With the extractors defined, the last step to analyse their success probabilities and run-time. To that end, first let $\kappa_{Q,W}^{(i+1)}$ denote the soundness error of proof system $\left(Q^{(i)}, W^{(i+1)}\right)$ and $p_{Q,W}^{(i+1)}$ the polynomial factor in the extraction probability. Functions $\kappa_\rho^{(i+1)}$ and $p_\rho^{(i+1)}$ are defined similarly for $\mathrm{E}_\rho^{(i+1)}$, and $\kappa_\parallel$ and $p_\parallel$ for $\mathrm{E}_\parallel$.

Set $(P^*)^{(n)}(\cdot) \coloneqq \textsc{Prove}^*(st, n, \cdot)$, and in case $n \geq 1$, algorithms $(Q^*)^{(i-1)}$ and $(P^*)^{(i-1)}$, for $i \in \{1, \ldots, n\}$, are as defined above. Recall that $aux_{n+1} \coloneqq aux^*$ is in $L_{aux}^{(n+1)}$ by assumption, and that by the previous analysis, aux_i is consequently in $L_{aux}^{(n)}$ for all $i \in \{1, \ldots, n\}$ if $\mathrm{E}^{(n+1)}$ does not abort.

Let Acc denote the event in which for some $i \in \{i_0 + 4, \ldots, n\}$ (in case $n \geq i_0 + 4$), T_i is $L_{\mathrm{V_F}''_\parallel}$ but T_{i-1} or $\bar{\tau}_i$ is not. The success probability of the extractor is first conducted under the condition that Acc does not occur, and the probability that it does is accounted for afterwards.

For all $i \in \{0, \ldots, n-1\}$, let $q\left((P^*)^{(i)}\right)$ denote the probability that algorithm $\left((P^*)^{(i)}\right)^{(P^*)^{(i+1)}}$ computes a triple $(\pi_{i+1}, aux_{i+1}, st_{i+1})$ that $\mathrm{V_F}$ accepts (π_{i+1}, aux_{i+1}) for the instance $(i+1, x_0, x_{i+1})$, conditioned on event $\overline{Acc}$.

For $i \in \{1, \ldots, n\}$, let $q\left((Q^*)^{(i-1)}\right)$ denote the probability that algorithm $\left((Q^*)^{(i-1)}\right)^{(P^*)^{(i)}}$ returns an output distinct from $\varnothing$, still conditioned on event $\overline{Acc}$. Note that $(Q^*)^{(i-1)}$ runs $\mathrm{E}_{Q,W}^{(i+1)}$ on a fixed value aux_{i+1}. By an averaging argument, conditioned on the event that $\mathrm{V_F}$ accepts (π_{i+1}, aux_{i+1}), for any real value $0 < \gamma < 1$, aux_{i+1} is with probability at least $1-\gamma$ such that $(P^*)^{(i)}$ returns a pair that has aux_{i+1} as second component and that $\mathrm{V_F}$ accepts with probability at least $\gamma q\left((P^*)^{(i)}\right)$. For $\gamma \leftarrow 1/2$, it means that with probability at least $1/2$, algorithm $(P^*)^{(i)}$ computes a pair that has aux_{i+1} as second component and that $\mathrm{V_F}$ accepts with probability at least $q\left((P^*)^{(i)}\right)/2$. Therefore, assuming that $q\left((P^*)^{(i)}\right) > 2\kappa_{Q,W}^{(i+1)}$, algorithm $\mathrm{E}_{Q,W}^{(i+1)}$ succeeds with probability at least

$$q\left(\mathrm{E}_{Q,W}^{(i+1)}\right) \coloneqq \left(q\left((P^*)^{(i)}\right)/2 - \kappa_{Q,W}^{(i+1)}\right) / \left(2p_{Q,W}^{(i+1)}\right).$$

Assuming also that $q\left(\left(P^*\right)^{(i)}\right) > \kappa_\rho^{(i+1)}$, algorithm $\left(\mathrm{E}_\rho^{(i+1)}\right)$ succeeds with probability at least

$$q\left(\mathrm{E}_\rho^{(i+1)}\right) \coloneqq \left(q\left(\left(P^*\right)^{(i)}\right) - \kappa_\rho^{(i+1)}\right)/p_\rho^{(i+1)}.$$

On this account, if $q\left(\left(P^*\right)^{(i)}\right) > \max\left(2\kappa_{Q,W}^{(i+1)}, \kappa_\rho^{(i+1)}\right)$, algorithm $\left(Q^*\right)^{(i-1)}$ returns an output distinct from $\varnothing$ with probability

$$q\left(\left(Q^*\right)^{(i-1)}\right) \geq 1 - \left(1 - q\left(\mathrm{E}_{Q,W}^{(i+1)}\right) + 1 - q\left(\mathrm{E}_\rho^{(i+1)}\right)\right).$$

If this lower bound is strictly greater than $\kappa_\parallel$, then

$$q\left(\left(P^*\right)^{(i-1)}\right) \geq 1 - \left(1 - \underbrace{\left(q\left(\left(Q^*\right)^{(i-1)}\right) - \kappa_\parallel\right)/p_\parallel}_{q\left(\mathrm{E}_\parallel^{\left(Q^*\right)^{(i-1)}}\right)} + \varepsilon_\Gamma\right).$$

Notice that the sequence of lower bounds on $\left(q\left(\left(P^*\right)^{(i)}\right)\right)_i$ is increasing, i.e., the closer to the start of the computation the extraction procedure gets, the lower the success probability of the prover is.

Now define a sequence of knowledge-soundness errors as follows:

$$\kappa^{(1)} \coloneqq \kappa_{Q,W}^{(1)}$$

$$\kappa^{(i+1)} \coloneqq \inf\Big\{q \in \left(\max\left(2\kappa_{Q,W}^{(i+1)}, \kappa_\rho^{(i+1)}\right), 1\right] : \text{ if } q\left(\mathrm{B}\right) = q \text{ then } q\left(\left(\left(Q^*\right)^{(i-1)}\right)^{\mathrm{B}}\right) > \kappa_\parallel \text{ and } q\left(\left(\left(P^*\right)^{(i-1)}\right)^{\mathrm{B}}\right) > \kappa^{(i)}\Big\}$$

for $i \in \{1, \ldots, n\}$ (if $n \geq 1$).

As for the extraction polynomial factor, let

$$p^{(1)} \coloneqq p_{Q,W}^{(1)}.$$

Suppose that $q\left(\left(P^*\right)^{(i-1)}\right) > \kappa^{(i)}$. Let $q\left(\mathrm{E}^{(i)}\right)$ denote the success probability of $\left(\mathrm{E}^{(i)}\right)^{\left(P^*\right)^{(i-1)}}$ conditioned on event $\overline{Acc}$. It is at least

$$\left(q\left(\left(P^*\right)^{(i-1)}\right) - \kappa^{(i)}\right)/p^{(i)},$$

assuming $p^{(i)}$ to have already been determined.

The success probability $q\left(\mathrm{E}^{(i+1)}\right)$ of $\left(\mathrm{E}^{(i+1)}\right)^{\left(P^*\right)^{(i)}}$ conditioned on event $\overline{Acc}$ is at least

$$1 - \left(1 - q\left(\left(P^*\right)^{(i)}\right) + 1 - q\left(\left(Q^*\right)^{(i-1)}\right) + 1 - q\left(\mathrm{E}^{(i)}\right)\right).$$

Define $p^{(i+1)}$ as a polynomial of smallest possible degree that is at least as large as the ratio between $\left(1-\kappa^{(i+1)}\right)$ and the previous lower bound on $q\left(\mathrm{E}^{(i+1)}\right)$ with $q\left(\left(P^*\right)^{(i)}\right)$ replaced with $\kappa^{(i+1)}$.

Let $\varepsilon_{\mathrm{Acc}}$ be the probability of event *Acc*. If ε is such that $\varepsilon - \varepsilon_{\mathrm{Acc}} > \kappa^{(n+1)}$, then $\mathrm{E}^{(n+1)}$ can extract from $\left(P^*\right)^{(n)}$ a sequence of valid intermediate IVC outputs and witnesses.

Error

$$\varepsilon_{\mathrm{Acc}} = \varepsilon_{\mathrm{Acc}}\left(\left(\mathrm{E}^{(n+1)}\right)^{\left(P^*\right)^{(n)}}, q_H\left(\left(\mathrm{E}^{(n+1)}\right)^{\left(P^*\right)^{(n)}}\right)\right),$$

is a function of the success probability of $\mathrm{E}^{(n+1)}$ with access to $\left(P^*\right)^{(n)}$, if the check that T_{i-1} and $\bar{\tau}_i$ are is $L_{\mathrm{VF}\|,1''}$ were not present in the definition of $\left(P^*\right)^{(i-1)}$, and of the number of queries it makes to oracle H. The former is precisely $q\left(\mathrm{E}^{(n+1)}\right)$ (removing the check is tantamount being in an event in which it always passes) and the latter depends on q_H (the number of queries $\textsc{Prove}^*$ makes) and on the expected run-time of $\mathrm{E}^{(n+1)}$. See Section 5.4 for an instantiation of algorithm Acc and an analysis of probability $\varepsilon_{\mathrm{Acc}}$.

In what follows, given an algorithm B, let $T(\mathrm{B})$ stand for its run-time. For $i \geq 1$,

$$T\left(\left(\mathrm{E}^{(i+1)}\right)^{\left(P^*\right)^{(i)}}\right) = T\left(\left(P^*\right)^{(i)}\right) + T\left(W^{(i+1)}\right) + T\left(\left(\left(Q^*\right)^{(i-1)}\right)^{\left(P^*\right)^{(i)}}\right) + T\left(\left(\mathrm{E}^{(i)}\right)^{\left(P^*\right)^{(i-1)}}\right)$$

by definition of $\mathrm{E}^{(i+1)}$. Furthermore,

$$T\left(\left(\left(Q^*\right)^{(i-1)}\right)^{\left(P^*\right)^{(i)}}\right) = T\left(\left(\left(E_{Q,W}\right)^{(i+1)}\right)^{\left(P^*\right)^{(i)}}\right) + T\left(\left(\left(E_\rho\right)^{(i+1)}\right)^{\left(P^*\right)^{(i)}}\right)$$

Let $n_{\mathrm{E}_{Q,W}}^{(i+1)}$ and $n_{\mathrm{E}_\rho}^{(i+1)}$ respectively denote the number of times that $\mathrm{E}_{Q,W}^{(i+1)}$ and $\mathrm{E}_\rho^{(i+1)}$ run $\left(P^*\right)^{(i)}$. It is in expectation at most polynomial in the length of their inputs. These extractors may also do a number of operations that is in expectation polynomial in the size of their inputs. These inputs are computed from outputs that $W^{(i+1)}$ accepts, so they are of a length that is polynomial in λ. This implies that

$$T\left(\left(\left(Q^*\right)^{(i-1)}\right)^{\left(P^*\right)^{(i)}}\right) = \left(n_{\mathrm{E}_{Q,W}}^{(i+1)} + n_{\mathrm{E}_\rho}^{(i+1)}\right) T\left(\left(P^*\right)^{(i)}\right) + \mathrm{poly}(\lambda).$$

Besides, $T\left(W^{(i+1)}\right) = \mathrm{poly}(\lambda)$. Consequently,

$$T\left(\left(\mathrm{E}^{(i+1)}\right)^{(P^*)^{(i)}}\right) \leq \left(n_{\mathrm{E}_{Q,W}}^{(i+1)} + n_{\mathrm{E}_\rho}^{(i+1)} + 1\right) T\left((P^*)^{(i)}\right) + \mathrm{poly}(\lambda) + T\left(\left(\mathrm{E}^{(i)}\right)^{(P^*)^{(i-1)}}\right).$$

In addition to that, if $i \geq i_0 + 2$, denoting by $n_\parallel$ the number of times that $\mathrm{E}_\parallel$ runs algorithm $\left((Q^*)^{(i-1)}\right)^{(P^*)^{(i)}}$, which is in expectation polynomial in the size of its inputs,

$$\begin{aligned} T\left(\left((P^*)^{(i-1)}\right)^{(P^*)^{(i)}}\right) &\leq T\left(\left((Q^*)^{(i-1)}\right)^{(P^*)^{(i)}}\right) + \mathrm{poly}(\lambda) \\ &\leq \left(n_{\mathrm{E}_{Q,W}}^{(i+1)} + n_{\mathrm{E}_\rho}^{(i+1)}\right) T\left((P^*)^{(i)}\right) + \mathrm{poly}(\lambda). \end{aligned}$$

These two inequalities show that the run-time of $\mathrm{E}^{(n+1)}$ is polynomial in expectation as long as n is constant in λ.

To define the knowledge-soundness error of the scheme, the number of oracle queries $\mathrm{E}^{(n+1)}$ must be bounded by above, which means that the run-time of E should strictly be polynomial instead of just polynomial in expectation. For this reason, instead of running exactly $\mathrm{E}^{(n+1)}$ on $(P^*)^{(n)}$, extractor E repeats the following procedure J times, for a positive integer J to be determined later. E runs $\mathrm{E}^{(n+1)}$ for a number of steps that is at most twice the expected run-time of $\mathrm{E}^{(n+1)}$. If $\mathrm{E}^{(n+1)}$ returns within that time an output distinct from $\varnothing$, algorithm E exits the loop and returns that output, and otherwise continues.

To analyse the success probability of E, remark that by Markov's inequality, the probability that $\mathrm{E}^{(n+1)}$ does not return an output within twice its expected runtime is at most $1/2$. Now let $(X_\nu)_{\nu \geq 1}$ be a family of independent random variables with the same binomial distribution of parameter β. In essence, X_ν indicates whether the ν-th iteration of the loop is successful, and β is at least half the success probability of $\mathrm{E}^{(n+1)}$ (*not* conditioned on event Acc). Let $T \coloneqq \min\{\nu \geq 1 \colon X_1 + \cdots + X_\nu = 1\}$. It is a stopping time which indicates the number of trials to obtain a success. The random variable $T - 1$, which indicates the number of trials before a success, has a negative binomial distribution with parameters 1 and β. Its expectation is then $(1-\beta)/\beta$. Therefore, by $\mathbb{E}[T] = 1/\beta$, which is at most twice the inverse of the success probability of $\mathrm{E}^{(n+1)}$.

Moreover, the Chernoff bound (Appendix A.3) implies that the probability that the number of trials is higher that this expectation decreases exponentially fast. Formally, consider a real number $\gamma > 0$, and let $J \leftarrow \lceil (1+\gamma)/\beta \rceil$. Note that $T > J$ if and only if $X_1 + \cdots + X_J < 1$, which is equivalent to

$$X_1 + \cdots + X_J - J\beta < 1 - J\beta \leq -\gamma.$$

Besides, $\mathbb{E}[X_1 + \cdots + X_J] = J\beta$, and $0 < \gamma/J\beta \leq 1$. The Chernoff bound implies that

$$\begin{aligned} P[T > J] &\leq P[X_1 + \cdots + X_J - J\beta < -\gamma] \\ &= P\left[X_1 + \cdots + X_J - J\beta < -(\gamma/J\beta)J\beta\right] \\ &\leq \exp(-(\gamma/J\beta)^2 J\beta/2) \\ &\sim \exp(-\gamma/2) \text{ as } \gamma \to \infty. \end{aligned}$$

In other words, the probability that none of the J executions of $\mathrm{E}^{(n+1)}$ leads to a success decreases exponentially with J. Now set $\gamma = \gamma(\delta)$ and J so that the success probability of E is arbitrarily close to that of $\mathrm{E}^{(n+1)}$. Yet, the run-time of E is strictly polynomial time, which means that the number of random oracle queries it makes is bounded. Now Let Acc' denote the event in which, during the execution of E (rather than $\mathrm{E}^{(n+1)}$) for some $i \in \{i_0 + 4, \ldots, n\}$ (in case $n \geq i_0 + 4$), T_i is $L_{\mathrm{V}\mathrm{F}''_{\|}}$ but T_{i-1} or $\bar{\tau}_i$ is not. Let $\varepsilon'_{\mathrm{Acc}}$ be the probability of event Acc'. If ε is such that $\varepsilon - \varepsilon'_{\mathrm{Acc}} > \kappa^{(n+1)}$, then E can extract from $(P^*)^{(n)}(\cdot) = \textsc{Prove}^*(st, n, \cdot)$ a sequence of valid intermediate IVC outputs and witnesses. Define then the knowledge-soundness error of the scheme as the infimum of the set of values ζ such that if the success probability of a prover is ζ, is larger than $\max\left(2\kappa_{Q,W}^{(n+1)}, \kappa_{\rho}^{(n+1)}\right)$ and is such that

$$q\left(\left((Q^*)^{(n-1)}\right)^{\mathrm{B}}\right) > \kappa_{\|}$$

and

$$q\left(\left((P^*)^{(n-1)}\right)^{\mathrm{B}}\right) > \kappa^{(n)},$$

then $\zeta - \varepsilon'_{\mathrm{Acc}} > \kappa^{(n+1)}$. Define the extraction factor as a polynomial of smallest possible degree that is at least as large as the ration between $(1 - \kappa)$ and the difference between the lower bound on $q\left(\mathrm{E}^{(n+1)}\right)$, with $q\left((P^*)^{(n)}\right)$ replaced with $\kappa^{(n+1)}$, and the additive factor (parametrised γ) between $q\left(\mathrm{E}^{(n+1)}\right)$ and the success probability of E.

It should be noted that the preceding analysis abuses the knowledge-soundness definition of proof systems because algorithms

$$(Q^*)^{(n-1)}, (P^*)^{(n-1)}, \ldots, (Q^*)^{(0)}, (P^*)^{(0)}$$

are only guaranteed to run in expected polynomial time, although the definition requires extraction only from provers that run in strict polynomial time. To resolve this issue, one could instead define from $\left((Q^*)^{(n-1)}\right)^{(P^*)^{(n)}}$ an algorithm $\left(\left(\tilde{Q}^*\right)^{(n-1)}\right)^{(P^*)^{(n)}}$ that runs in strict polynomial, in the exact same way extractor E was defined from $\mathrm{E}^{(n+1)}$. Then, define from $\left((P^*)^{(n-1)}\right)^{(P^*)^{(n)}}$ (that

still runs $(Q^*)^{(n-1)}$ as sub-routine, not $\left(\tilde{Q^*}\right)^{(n-1)}$ – to preserve the previous success probability conditioned on event Acc) an algorithm $\left(\left(\tilde{P^*}\right)^{(n-1)}\right)^{(P^*)^{(n)}}$ that runs in strict polynomial time overall. For all $i \in \{1, \ldots, n-1\}$ (if $i \geq 2$), define $\left(\left(\tilde{Q^*}\right)^{(i-1)}\right)^{(P^*)^{(i)}}$ from $\left((Q^*)^{(i-1)}\right)^{(P^*)^{(i)}}$, and $\left(\left(\tilde{P^*}\right)^{(i-1)}\right)^{(P^*)^{(i)}}$ (which still runs $(Q^*)^{(i-1)}$ as sub-routine) from $\left((P^*)^{(i-1)}\right)^{(P^*)^{(i)}}$. A common parameter γ can be chosen for all the algorithms, and so that the success probabilities (conditioned on event Acc) are negligibly (in λ) close to those of the original algorithms. As long as n is constant in λ, the loss in success probability from $(P^*)^{(0)}$ to $\left(\tilde{P^*}\right)^{(0)}$ remains negligible. □

5 Instantiation

The first part of this section shows how to represent ECC operations that arise in IVC constructions as a finite state machine. This gives a precise structure in that case to what is denoted A_0, A_1 and A_2 in Section 4. The section continues with instantiations of the building blocks of the construction given in Section 4.

5.1 EllIptiC-Curve State Machine.

In IVC constructions based on primitives that are secure under classical assumptions, the "expensive" operations usually are multi-scalar multiplications (ECC operations) that the verifier algorithm performs. To apply the construction in Section 4 to this case, the input-outputs to these constructions and the witnesses for their correct executions must be represented in the arithmetisation chosen for the IVC. What follows is an abstraction of these ECC operations. Then comes an instantiation of this abstraction in the Plonk arithmetisation.

5.1.1 Abstract State Machine. Let $E(\mathbb{F}_q)$ be an elliptic curve defined over a prime-order field F_q. Let $(\mathbb{G}, +)$ be a subgroup of the group of points of E that is of prime order r. Computations in $\mathbb{G}$ are captured by the following deterministic finite-state machine.

The state set is the union of

$$\{(0,0), (0,1), (\infty,0), (\infty,1)\}$$

and

$$\{(G,0), (G,1) \colon G \in \mathbb{G}\}.$$

The initial state is $(0,0)$ and the set of final states is $\{(\infty,0), (\infty,1)\}$. The boolean second state component is a flag indicating whether an invalid ECC

operation has occurred in the machine execution thus far. State $(\infty, 0)$ is hence interpreted as an accepting final state and state $(\infty, 1)$ as a rejecting final state.

The input alphabet is the set that consists of 0_E, $(+, P)$ for all $P \in \mathbb{G}$, $(\cdot, a, P)$ for all $a \in \mathbb{F}_r$ and $P \in \mathbb{G}$, $(=, P)$ for all $P \in \mathbb{G}$, and ∞. These represent "reset," "addition," "scalar-multiplication" and "equality" operations.

The state-transition function is now defined as follows. From a state $(0, b)$ with $b \in \{0, 1\}$, if the input is 0_E, then the next state is set to $(0_E, b)$. If the input is ∞, the next (and final) state is set to (∞, b). From a state (G, b) with $G \in \mathbb{G}$ and $b \in \{0, 1\}$, if the input is $(+, P)$, then the next state is set to $(G + P, b)$. If the input is $(\cdot, a, P)$, then the next state is set to $(G + aP, b)$. If the input is $(=, P)$, then the next state is set to $(0, b)$ if $G = P$ and otherwise to $(0, 1)$. Note that once the second state component is set to 1, it can never revert to 0.

To illustrate an execution of the state machine, consider two field elements a and b, as well as four group elements $P_0, \ldots, P_3$. The computation of two separate group elements $Q \coloneqq P_0 + P_1 + aP_2 = ((0_E + P_0) + P_1) + aP_2$ and $R \coloneqq bP_3$ is captured via the following sequence of inputs

$$\begin{aligned} &0_E \to (+, P_0) \to (+, P_1) \to (\cdot, a, P_2) \to (=, Q) \\ \to\ &0_E \to (\cdot, b, P_3) \to (=, R) \to \infty. \end{aligned}$$

5.1.2 State-Machine Execution in the Plonk Arithmetisation. To represent an execution of the ECC state machine in the Plonk arithmetisation, it is necessary and sufficient to "encode" its state and input at each step within an execution trace (in $\mathbb{F}_r$). Encoding the state and input at a given execution step here means defining a one-to-one correspondence between that state and input and the entries of an execution-trace sub-matrix (together with a bit indicating whether all gate constraints are so far satisfied). The correspondence naturally depends on the relative sizes of $\mathbb{F}_r$ and $\mathbb{F}_q$.

First suppose that $r < q < 2r$. It is for instance the case for the BN254 curve, for which $2^{253} < r < q < 2^{254}$, and which is still one of the most prevalent choice for on-chain SNARKs because of existing Ethereum pre-compiles. Let ℓ denote the binary length of r, i.e., $2^{\ell-1} \leq r < 2^\ell$. The condition $q < 2r$ implies that $q < 2^{\ell+1}$. Now let k be a integer such that $\ell < 2k \leq 2(\ell-1)$, which exists as soon as $\ell > 2$. Under these conditions, for any integer $x \in \{0, \ldots, q-1\}$, there exists a unique pair (x_0, x_1) of k-bit integers, i.e., integers $\{0, \ldots, 2^k - 1\} \subseteq \{0, \ldots, r-1\}$ such that $x = x_0 + 2^k x_1$. (The inclusion is because $k \leq \ell - 1$, which implies that $2^k \leq 2^{\ell-1} \leq r$.) Unicity of the pair is guaranteed via the Euclidian division of x by 2^k. As for its existence, it follows from the fact that

$$2^k - 1 + 2^k \left(2^k - 1\right) = 2^{2k} - 1 \geq 2^{\ell+1} - 1 \geq q.$$

In other words, an element in $\mathbb{F}_q$ can be represented by two elements of F_r.

For inputs of the state machine that contain a group element, that input is a pair (x, y) of $\mathbb{F}_q$ elements in Weierstrass affine coordinates. What precedes shows that x and y can uniquely be written as pairs (x_0, x_1) and (y_0, y_1) of $\mathbb{F}_r$

elements. On this account, the state (except for the boolean second state component) and input (distinct from ∞) at each execution step of the machine can be written in two consecutive rows of a four-column execution trace as follows. Define an arbitrary one-to-one correspondence between each $\mathrm{op} \in \{0_E, +, \cdot, =\}$ and a subset of $\mathbb{F}_r$, e.g., $\{0, \ldots, 3\}$ (and each operation is further conflated with its image).

$$\begin{array}{cccc} \mathrm{op} & x_0 & x_1 & y_0 \\ 0 & y_1 & a & 0 \end{array}$$

If $\mathrm{op} = 0_E$, set $x_0 \leftarrow x_1 \leftarrow y_0 \leftarrow y_1 \leftarrow a \leftarrow 0$. If $\mathrm{op} \neq \cdot$, set $a \leftarrow 0$. Input ∞ is not represented in the execution trace as it is, by construction, necessarily the final input to the machine, and that final input is the only occurrence of ∞.

Consecutive ECC operations represented in this way can be accumulated in an execution trace from top to bottom.

In fact, if the elliptic curve has an efficiently computable endomorphism ψ, a scalar multiplication $a \cdot P$ can be reduced to [11,12] computing $a_0 \cdot P + a_1 \cdot \psi(P)$, for a_0 and a_1 of around half the binary size of a. This reduces by half the amount of computation to be proved in Section 6.1.2 if the most significant half of the bits of a only comprises zeroes ($a_1 = 0$). In the execution trace, the second row can thus be replaced with

$$\begin{array}{cccc} 0 & y_1 & a_0 & a_1 \end{array} .$$

5.2 Commitment Scheme

The scheme used to commit to ECC-operation tables, denoted Γ in Section 4, is the hiding KZG commitment scheme presented in Section 2.4.2. Let $N_{\max}$ the maximum length of the vectors that can be committed using parameters that have been generated for hiding KZG commitments. Note that a commitment to a vector $A \in \mathbb{F}^N$ for some $N < N_{\max}$ is also a commitment to $(A \parallel \mathbf{0}^j)$ for any $j \in \{1, \ldots, N_{\max} - N\}$. On this account, a KZG commitment to a vector A is actually a commitment to the set of vectors $(A \| \mathbf{0}^j)$ for all $j \in \{1, \ldots, N_{\max} - N\}$. The exact length (as referred to in Section 4) of a vector $A \in \mathbb{F}^{N_{\max}}$ is defined as the sum of 1 and the maximum index $i \in \{0, \ldots, N_{\max} - 1\}$ such that $A_i \neq 0$ if such an index exists, and 0 otherwise.

5.3 Concatenation Proofs

Consider three positive integers N_0, N_1 and N_2 such that N_0 and $N_1 + N_2$ are at most $N_{\max}$. Let $A_0 \in \mathbb{F}^{N_0}$, $A_1 \in \mathbb{F}^{N_1}$ and $A_2 \in \mathbb{F}^{N_2}$. Define $A_0(X)$, $A_1(X)$ and $A_2(X)$ as the univariate polynomials of which vectors A_0, A_1 and A_2 are the coefficients in the monomial basis. The main observation is that

$$A_0 = A_1 \parallel A_2 = \left(A_1, \mathbf{0}^{N_2}\right) + \left(\mathbf{0}^{N_1}, A_2\right)$$

if and only if

$$A_0(X) = A_1(X) + X^{N_1} A_2(X).$$

By the polynomial-identity lemma, the latter equation holds, except with probability at most $N_{\max}/|\mathbb{F}|$, if

$$Z_x \coloneqq A_0(X) - A_1(X) - x^{N_1} A_2(X)$$

evaluates to 0 at x, for a uniformly random $x \leftarrow_{\$} \mathbb{F}$.

Given commitments C_0, C_1 and C_2 to vectors A_0, A_1 and A_2, and the length N_1 of A_1, the verifier sends to the prover a uniformly random field element x, computes a commitment $C_0 - C_1 - x^{N_1} C_2$ to Z_x and uses it to verify a proof that $Z_x(x) = 0$.

To apply this idea to the case where A_0, A_1 and A_2 are not column vectors but rather matrices with M columns, the prover must now show that for each $j \in \{0, \ldots, M-1\}$, polynomial

$$A_{0,j}(X) - A_{1,j}(X) - X^{N_1} A_{2,j}(X)$$

is the zero polynomial. This is equivalent to proving that the bivariate polynomial

$$\sum_{j=0}^{M-1} Y^j \left(A_{0,j}(X) - A_{1,j}(X) - X^{N_1} A_{2,j}(X) \right)$$

is the zero polynomial. Except with probability at most $(M + N_{\max})/|\mathbb{F}|$ (again via the polynomial-identity lemma), it is the case if

$$Z_{x,y} \coloneqq \sum_{j=0}^{M-1} y^j \left(A_{0,j}(X) - A_{1,j}(X) - x^{N_1} A_{2,j}(X) \right)$$

evaluates to 0 at x. The verifier algorithm can thus proceed similarly to the case of single-column vectors (i.e., $M = 1$), except that it now also sends a uniformly random field element y (independent from x) in the first protocol round. Note that instead of being independent of X, variable Y could be replaced with $X^{N_{\max}}$ (which would incur a larger number of field operations).

Formally, the following protocol, denoted $\Pi_{\|}$, is for the relation

$$\left\{ \left((C_0, (C_1, \ell), C_2), \left(A_k \in \mathbb{F}^{N_{\max} \times M}, r_k \right)_{k=0}^{2} \right) : \Gamma.\textsc{Open}\left(C_k, A_k, r_k\right) = 1 \right.$$
$$\left. A_0(X) = A_1(X) + X^{\ell} A_2(X) \right\},$$

$\textbf{Prove} \leftarrow \textbf{V}_{\textbf{F}}:\ x, y \leftarrow_{\$} \mathbb{F}$

$\textbf{V}_{\textbf{F}}:\ C_Z \leftarrow \sum_{k=0}^{M-1} y^k \cdot \left(C_{j,0} - C_{j,1} - x^{\ell} C_{j,2} \right)$

$\textbf{Prove} \rightarrow \textbf{V}_{\textbf{F}}:$

$q \leftarrow Z_{x,y} : (X - x)$

$s \leftarrow_{\$} \mathbb{F}$

$\pi \leftarrow [q(\tau)]_1 + s \cdot [\xi]_1$

$\delta \leftarrow r \cdot [1]_1 - s \cdot [\tau]_1 + (s \cdot x) \cdot [1]_1 = [r - s(\tau - x)]_1$

Output (π, δ)

$\mathbf{V_F}: \; e\left(C_Z, [1]_2\right) \stackrel{?}{=} e\left(\pi, [\tau]_2 - x \cdot [1]_2\right) + e\left(\delta, [\xi]_2\right).$

The non-interactive proof system denoted $\Pi_{\|}$ in Section 4 can be instantiated with the Fiat–Shamir transformation [9] of this interactive proof system. The IVC prover in Section 4 proves a statement about $\mathrm{V}_{\mathrm{F}_{\|}}$, so about a concrete hash function instantiating the random oracle. Therefore, the security of the instantiation of the construction in Section 4 with the Fiat–Shamir transformation of this proof system is only heuristic. However, that is already the case for existing IVC proof systems which apply the Fiat–Shamir transformation to either a interactive universal SNARK or to an interactive folding scheme.

5.4 Accumulation Algorithm

Given the above protocol, define $\mathrm{V}''_{\mathrm{F}_{\|},1}$ as the algorithm which takes a tuple $(C_Z, x, (\pi, \delta))$, and returns 1 if

$$e\left(C_Z, [1]_2\right) = e\left(\pi, [\tau]_2\right) - e\left(x \cdot \pi, [1]_2\right) + e\left(\delta, [\xi]_2\right)$$

and 0 otherwise. (The function is parametrised by $vk_{\|}$.) Following the notation of Section 4, Given $T' \coloneqq (C'_Z, x', (\pi', \delta'))$ and $\bar{\tau} \coloneqq (C_Z, x, (\pi, \delta))$ as inputs, algorithm Acc generates $\alpha \leftarrow \mathbb{F}$ and returns $T \leftarrow \bar{\tau} + \alpha \cdot T'$. Note that

$$\begin{aligned}
& e\left(C_Z + \alpha C'_Z, [1]_2\right) - e\left(\pi + \alpha\pi', [\tau]_2\right) + e\left(x \cdot \pi + \alpha\left(x' \cdot \pi'\right), [1]_2\right) \\
& - e\left(\delta + \alpha\delta', [\xi]_2\right) \\
& = e\left(C_Z, [1]_2\right) - e\left(\pi, [\tau]_2\right) + e\left(x \cdot \pi, [1]_2\right) - e\left(\delta, [\xi]_2\right) \\
& + \alpha \cdot \left(e\left(C'_Z, [1]_2\right) - e\left(\pi', [\tau]_2\right) + e\left(x' \cdot \pi', [1]_2\right) - e\left(\delta', [\xi]_2\right)\right) \\
& = \left(\mathrm{dlog}_{[1]_T}(\cdot) + \alpha \cdot \mathrm{dlog}_{[1]_T}(\cdot)\right) \cdot [1]_T.
\end{aligned}$$

$\left(\mathrm{dlog}_{[1]_T}(\cdot) + \alpha \cdot \mathrm{dlog}_{[1]_T}(\cdot)\right)$ is a degree-1 polynomial in $\mathbb{F}[\alpha]$. If T' and $\bar{\tau}$ satisfy the pairing equation, it is the zero polynomial and thus T satisfies the pairing equation as well. Conversely, if T' or $\bar{\tau}$ is non-zero, the polynomial is non-zero and by the polynomial-identity lemma, T satisfies the pairing equation with probability at most $1/|\mathbb{F}|$.

The construction in Section 4 de-randomises this algorithm by using a hash function modelled as a random oracle. (Its domain should in practice be separated from that of the random oracle used in the Fiat–Shamir transformation

of the proof system in Section 5.3.) The inputs to the algorithm are hashed to obtain the randomness on which the algorithm runs.

Denote by φ the linear map

$$(C_Z, x, (\pi, \delta)) \mapsto e\left(C_Z, [1]_2\right) - e\left(\pi, [\tau]_2\right) + e\left(x \cdot \pi, [1]_2\right) - e\left(\delta, [\xi]_2\right).$$

In the random-oracle model, consider an algorithm which makes at most q_H oracle queries and which, with probability at least ε_φ, returns $m \geq 1$ tuple

$$\left(T'_0, \bar{\tau}_0, T_0\right), \ldots, \left(T'_{m-1}, \bar{\tau}_{m-1}, T_{m-1}\right)$$

such that for all $i \in \{0, \ldots, m-1\}$, with $\alpha_i \coloneqq H\left(T'_i, \bar{\tau}_i\right)$, $T_i = \bar{\tau}_i + \alpha_i T'_i$ and $\varphi\left(\bar{\tau}_i + \alpha_i T'_i\right) = 0$.

Bagherzandi, Cheon and Jarecki [1] designed a forking algorithm which, when applied to the present algorithm, returns tuples

$$\left(T'_0, \bar{\tau}_0, T_{0,0}\right), \left(T'_0, \bar{\tau}_0, T_{0,1}\right), \ldots, \left(T'_{m-1}, \bar{\tau}_{m-1}, T_{m-1,0}\right), \left(T'_0, \bar{\tau}_0, T_{m-1,1}\right)$$

such that for all i in $\{0, \ldots, m-1\}$ and all $b \in \{0,1\}$, $T_{i,b} = \bar{\tau}_i + \alpha_{i,b} T'_i$, $\alpha_{i,0} \neq \alpha_{i,1}$, and $\varphi(T_{i,0}) = \varphi(T_{i,1}) = 0$. It does so with probability at least $\varepsilon/8$, and in at most $8m^2 q_H/\varepsilon \cdot \ln(8m/\varepsilon)$ times the runtime of the forked algorithm.

The forking algorithm does so by generating $m+1$ executions of the algorithm with the same random string; only the random-oracle answers change. In each execution $j \in \{0, \ldots m\}$, for all $i \in \{0, \ldots, m-1\}$, the algorithm queries the oracle (among other queries) at a pair $\left(\left(T'_i\right)^{(j)}, \bar{\tau}_i^{(j)}\right)$, receives an answer $\alpha_i^{(j)}$, and returns a value $T_i^{(j)}$ such that $T_i^{(j)} = \bar{\tau}_i^{(j)} + \alpha_i^{(j)} \left(T'_i\right)^{(j)}$ and $\varphi\left(T_i^{(j)}\right) = 0$. Importantly, for each i, the corresponding query is made at the same computation step. The executions are also such that all random-oracle answers in executions i and $i+1$ (for $i \in \{0, \ldots, m-1\}$) are the same up to the step at which the algorithm queries the oracle at $\left(\left(T'_i\right)^{(i)}, \bar{\tau}_i^{(i)}\right)$. The answers in the two executions are generated uniformly and independently, and the forking algorithm eventually returns a non-empty value only if the answers are distinct.

The fact that the answers are the same up to the forking point implies that $\left(T'_i\right)^{(i)} = \left(T'_i\right)^{(i+1)}$ and $\bar{\tau}_i^{(i)} = \bar{\tau}_i^{(i+1)}$. Set then $T'_i \leftarrow \left(T'_i\right)^{(i)}$, $\bar{\tau}_i \leftarrow \bar{\tau}_i^{(i)}$, $\alpha_{i,0} \leftarrow H\left(\left(T'_i\right)^{(i)}, \bar{\tau}_i^{(i)}\right)$, $\alpha_{i,1} \leftarrow H\left(\left(T'_i\right)^{(i+1)}, \bar{\tau}_i^{(i+1)}\right)$ $T_{i,0} \leftarrow T_i^{(i)}$ and $T_{i,1} \leftarrow T_i^{(i+1)}$. As $T'_i = (T_{i,0} - T_{i,1})(\alpha_{i,0} - \alpha_{i,1})^{-1}$ and $\bar{\tau}_i = T_{i,0} - \alpha_{i,0} T'_i$, the linearity of φ implies that $\varphi\left(\bar{\tau}_i\right) = \varphi\left(T'_i\right) = 0$.

In the knowledge-soundness proof of the construction in Section 4, T_i is computed as $\bar{\tau}_i + \alpha_i T_{i-1}$ from step i_0+3 included. Integer m is therefore $n - i_0 - 2$ in that case. The forked algorithm is extractor $\mathrm{E}^{(n+1)}$, and as shown in the proof of knowledge soundness, the run-time of $\mathrm{E}^{(n+1)}$ is polynomial in expectation as long as n is polynomial in the security parameter λ, so the run-time of the forking algorithm applied to $\mathrm{E}^{(n+1)}$ is polynomial in expectation, and the number of queries it makes also is. The forking lemma of Bagherzandi, Cheon and Jarecki implies that the probability $\varepsilon_{\mathrm{Acc}}$ of event *Acc* is at most $1 - q\left(\mathrm{E}^{(n+1)}\right)/8$.

In spite of this analysis, as in the case of the concatenation proof, the IVC prover in Section 4 proves a statement about Acc, so about a concrete hash function instantiating the random oracle. The security of the construction is therefore heuristic.

6 Proving Validity of Auxiliary Proof Information

This section first shows how to prove the validity of ECC-operation tables as defined in Section 5.1.2. It then gives a simple method to check the validity of auxiliary proof information.

6.1 Validity of ECC-Operation Tables

As in Section 5.1.2, consider an elliptic curve E over a prime-order field $\mathbb{F}_q$ and a group of points in $E(\mathbb{F}_q)$ that is of prime order r such that $r < q < 2r$. As shown in that section, operations in that group can be represented in a four-column Plonk execution trace over $\mathbb{F}_r$ in two consecutive rows:

$$\begin{array}{cccc} \text{op} & x_0 & x_1 & y_0 \\ 0 & y_1 & a & 0 \end{array}$$

for $\text{op} \in \{0_E, +, \cdot, =\}$, and x_0, x_1, y_0 and y_1 in $\{0, \ldots, 2^k - 1\}$ for some integer k such that $\ell < 2k \leq 2(\ell - 1)$. Such a matrix $A_r \in \mathbb{F}_r^{2N \times 4}$ is considered valid if the state machine (Section 5.1) it represents ends in an accepting state.

6.1.1 From $\mathbb{F}_r$ to $\mathbb{F}_q$.

To prove that the validity of A_r, let

$$\iota \colon \mathbb{F}_r \cong \mathbb{Z}/r\mathbb{Z} \hookrightarrow \mathbb{Z} \to \mathbb{Z}/q\mathbb{Z} \cong \mathbb{F}_q$$

denote the map which maps an $\mathbb{F}_r$ element with representative in $\{0, \ldots, r-1\}$ to the class in $\mathbb{F}_q$ of that representative. Now define a matrix A_q in $\mathbb{F}_q^{N \times 4}$ such that

$$\begin{aligned} A_{q,i,0} &= \iota(A_{r,2i,0}) \\ A_{q,i,1} &= \iota(A_{r,2i,1}) + 2^k \iota(A_{r,2i,2}) \\ A_{q,i,2} &= \iota(A_{r,2i,3}) + 2^k \iota(A_{r,2i+1,1}) \\ A_{q,i,3} &= \iota(A_{r,2i+1,2}) \end{aligned}$$

for all $i \in \{0, \ldots, N-1\}$. The rows of A_q are thus of the form

$$\begin{array}{cccc} \text{op} & x & y & \iota(a) \end{array} .$$

In case the curve has an efficiently computable endomorphism, assuming that A_q is defined with 5 columns instead of 4, its rows are of the form

$$\text{op} \qquad x \qquad y \qquad \iota(a_0) \qquad \iota(a_1) \quad .$$

It is possible to prove the validity of A_q with a universal SNARK for circuits defined over $\mathbb{F}_q$, but one must first prove that it is correctly computed from A_r. To that end, it suffices to show that

$$\begin{aligned}
\sum_{0\leq i\leq N-1} A_{q,i,j}X^i &= \sum_{0\leq i\leq N-1} \iota(A_{r,2i,0})X^i \\
\sum_{0\leq i\leq N-1} A_{q,i,1}X^i &= \sum_{0\leq i\leq N-1} \left(\iota(A_{r,2i,1}) + 2^k\iota(A_{r,2i,2})\right)X^i \\
\sum_{0\leq i\leq N-1} A_{q,i,2}X^i &= \sum_{0\leq i\leq N-1} \left(\iota(A_{r,2i,3}) + 2^k\iota(A_{r,2i+1,1})\right)X^i \\
\sum_{0\leq i\leq N-1} A_{q,i,3}X^i &= \sum_{0\leq i\leq N-1} \iota(A_{r,2i+1,2})X^i
\end{aligned}$$

in $\mathbb{F}_q[X]$. (There is an extra equation to be proved if scalar a is split in two.) These four polynomial equations hold if and only if an equality holds between the two bivariate polynomials in X and Y of which the left-hand and right-hand sides of the equalities respectively are their coefficients of degree $0, \ldots, 3$ in Y. This observation allows to prove all equalities at once. However, to explain in a simple manner how to prove such an equality, first consider the case in which there is only the first equation to prove. The proof for four equations at once is a straightforward generalisation of that case.

Suppose that the prover is given a univariate commitment to

$$\sum_{0\leq i\leq N-1} A_{q,i,j}X^i.$$

The verifier can choose a uniformly random value $x \leftarrow_\$ \mathbb{F}_q$ and the prover can reveal the evaluation of that polynomial at x, denote it v_x together with a proof that the evaluation is correct. The next step is to prove that

$$\sum_{0\leq i\leq N-1} \iota(A_{r,2i,0})x^i = v_x \bmod q.$$

It is sufficient for that to prove the correct execution of an algorithm which takes $(\iota(A_{r,i,0}))_{i=0}^{2N-1}$, x and v_x as input, computes $\sum_{0\leq i\leq N-1} \iota(A_{r,2i,0})x^i$ and accepts if and only if it is equal to v_x modulo q.

$w_x \leftarrow \iota(A_{r,2(N-1),0})$

For $i = 2(N-2)$ **down to** 0 **by 2**

$\quad w_x \leftarrow \iota(A_{r,i,0}) + x \cdot w_x \bmod q$

$w_x \stackrel{?}{=} v_x \bmod q$

In practice, the prover is given a commitment to $(A_{r,i,0})_{i=0}^{2N-1}$, so the execution of this algorithm is proved via a circuit defined over $\mathbb{F}_r$. Since the algorithm performs computations modulo q, proving the correct execution of this algorithm requires to prove arithmetic operations in a foreign field.

This method for proving one of the polynomial equations can be generalised to the case in which all four equations are proved at once, provided that the prover has commitments to the columns of matrix A_r. Namely, the prover uses a univariate commitment scheme to prove an evaluation $v_{x,y}$ at (x, y) of the bivariate polynomial of which the coefficients at $Y^0, \ldots, Y^3$ are given by the univariate polynomials on the right sides of the previous equalities. The previous algorithm now initialises a value

$$\begin{aligned} w_{x,y} \leftarrow \iota(A_{r,2(N-1),0}) &+ y \cdot \left(\iota(A_{r,2(N-1),1}) + 2^k \iota(A_{r,2N-1,2})\right) \\ &+ y^2 \cdot \left(\iota(A_{r,2(N-1),3}) + 2^k \iota(A_{r,2N-1,1})\right) \\ &+ y^3 \cdot \iota(A_{r,2N-1,2}) \bmod q \end{aligned}$$

and for $i = 2(N-2)$ down to 0 by 2, updates it as

$$\begin{aligned} w_{x,y} \leftarrow \iota(A_{r,i,0}) &+ y \cdot \left(\iota(A_{r,i,1}) + 2^k \iota(A_{r,i,2})\right) \\ &+ y^2 \cdot \left(\iota(A_{r,i,3}) + 2^k \iota(A_{r,i+1,1})\right) \\ &+ y^3 \cdot \iota(A_{r,i+1,2}) \\ &+ x \cdot w_{x,y} \bmod q. \end{aligned}$$

Assuming that representing $\mathbb{F}_q$ elements as two k-bit integers allows to compute the above without having to constrain $2^k \cdot \iota(A_{r,i,2}) \bmod q$ in an $\mathbb{F}_r$ circuit, that is four $\mathbb{F}_q$ multiplications in total per ECC operation to be enforced in an $\mathbb{F}_r$ circuit.

Note that no operation on the curve defined over $\mathbb{F}_r$ must be constrained in an $\mathbb{F}_r$ circuit. It means that the scheme does not actually need a curve cycle, it only requires the existence of another cryptographic group that has order q.

6.1.2 Validity of Operation Tables in $\mathbb{F}_q$. Proving the validity of operations in $\{0_E, +, =\}$ is straightforward. The one which requires care is the $\cdot$ scalar-multiplication operation.

As in section 5.1.2, integer ℓ denotes the binary length of r, i.e., $2^{\ell-1} \leq r < 2^\ell$. Consider a positive integer h less than $\ell - 1$ and the Euclidian division $\ell - 1 = hm + s$ of $\ell - 1$ by h.

Note that

$$\left(2^h\right)^m \leq \left(2^h\right)^m \cdot 2^s = 2^{\ell-1} \leq r < 2^\ell = \left(2^h\right)^m \cdot 2^{s+1} \leq \left(2^h\right)^{m+1}.$$

The decomposition of $a < r$ in base 2^h thus has m digits; denote them $a_0, \ldots, a_{m-1}$. Now observe that for any group element H, of which the distribution is to be

determined later,

$$\begin{aligned}
a \cdot G &= \sum_{0 \leq i < m} \left(2^h\right)^i \cdot a_i \cdot G \\
&= \sum_i \left(2^h\right)^i \cdot (a_i \cdot G + H - H) \\
&= \sum_i \left(2^h\right)^i \cdot (a_i \cdot G + H) - \sum_i \left(2^h\right)^i H \\
&= (a_0 G + H) + 2^h \cdot \left((a_1 G + H) + 2^h \cdot \left(\cdots 2^h \cdot (a_{m-1} G + H) \cdots\right)\right) \\
&\quad - \sum_i \left(2^h\right)^i H.
\end{aligned}$$

This suggests an algorithm akin to the double-and-add algorithm in which instead of doubling at each round, the algorithm doubles h times, and adds a term $(a_i G + H)$. To verify the correct execution of this algorithm, constraints enforcing correct point adding and doubling are necessary.

Given three points $P = (x_P, y_P), Q = (x_Q, y_Q)$ and $R = (x_R, y_R)$ in $E(\mathbb{F}_q)$, assuming that $Q \neq \pm P$ and that P and Q are both not the neutral element of E (the point 0_E at infinity), the equality $P \pm Q = R$ is satisfied if and only if [23]

$$\begin{aligned}
(x_P + x_Q + x_R)(x_Q - x_P)^2 - (y_Q \pm y_P)^2 &= 0 \\
(y_P + y_R)(x_Q - x_P) - (y_Q \pm y_P)(x_P - x_R) &= 0.
\end{aligned}$$

Moreover, in case $Q = P$, the equality $2P = R$ holds if and only if

$$\begin{aligned}
3x_P^2(x_P - x_R) - 2y_P(y_P + y_R) &= 0 \\
4y_P^2(x_R + 2x_P) - 9x_P(y_P^2 - b) &= 0
\end{aligned}$$

Elliptic-curve-point coordinates can be arranged in a Plonk execution trace as follows.

x_P	y_P	x_Q	y_Q
0	x_R	y_R	0

(In case $P = Q$, x_Q and y_Q can be replaced with 0.) The above equations thus yield an algebraic equation that wire values must satisfy.

Distribution of H. The previous equations hold in case neither P nor Q is 0_E, and if $Q \neq -P$. If H were set to 0_E, then at the $i+1$-th step of the algorithm, when $a_{m-1-i} \cdot G$ is to be added to the mutable variable which eventually holds $a \cdot G$, it is likely that $a_{m-1-i} = 0$. More precisely, it suffices for a to have 2^h consecutive bits in its binary decomposition to be 0, which is likely if h is small. In that case, the above equations which constrain the execution trace of the algorithm to be correct would not be applicable. It is of course possible to introduce conditionals to check whether the point to be added is 0_E and simply set the result to match the first argument, but this would introduce several extra

constraints. Alternatively, one could simply hash into E to obtain a uniformly random value in the group. The probability that $a_{m-1-i}G + H$ is zero for any i is negligible. Besides, the probability (over the distribution of H) that, at any step of the algorithm execution, the point to be added is the inverse of the value currently stored is also negligible. Hashing into E to generate H thus guarantees that with high probability, the equations given for the case $P, Q \neq 0_E$ and $P \neq -Q$ are sufficient to verify the correct execution of the scalar-multiplication algorithm.

Multi Scalar Multiplications. Consecutive scalar-multiplication operations can be considered as a single Multi-Scalar Multiplication (MSM). In that case, instead of writing the trace of successive executions of an algorithm for single-scalar multiplication, write instead the trace of an MSM algorithm best suited for the size of the MSMs that arise in practice. This can be Straus's algorithm [25], Pippenger's algrorithm [21] or sliding-window methods [27]. In any case, algorithms to compute MSMs often make use of variant of the double-and-add algorithm, and potentially leverage results from a pre-computation phase.

Choice of h. The larger h is, the less iterations there are in the scalar-multiplication algorithm, and the shorter the execution trace gets. However, the number of group operations in a pre-computation phase grows with h in some MSM algorithms. For instance, in Straus's algorithm applied to an MSM of size $\nu \geq 2$, the pre-computation phase requires $2^{\nu h} - 1 - \nu$ group operations. The online phase requires $mh = \ell - 1 - s$ doublings and approximately ℓ/h group additions. For $\ell = 254$, the optimal value of h seems to be around 4.

6.2 Validity of Auxiliary Proof Information

To now show the validity of an auxiliary piece of proof information is after $n \geq i_0 + 3$ steps construction, the verifier simply checks that $C_0^{(n)}$ is a commitment to a valid table of ECC operations as per Section 6.1, that $\mathrm{VF}_{\|}$ accepts on input

$$\left(par_{\|}, \left(C_0^{(n)}, \left(C_1^{(n)}, \ell_n\right), C_2^{(n)}\right), \pi_{\|,n+1}\right),$$

and that T is in $L_{\mathrm{VF}''_{\|,1}}$, i.e., that $\varphi(T) = 0$, for φ defined in Section 5.4.

7 Performance

Throughout the incremental computation with auxiliary proof information, only the proof at the end that table A_q is correctly computed from A_r (Section 6.1.1) requires $\mathbb{F}_q$ operations to be emulated in $\mathbb{F}_r$. That is four or five $\mathbb{F}_q$ multiplications, depending on whether the curve has an efficiently endomorphism. In comparison, standard IVC proofs requires $\mathbb{F}_q$ operations to be simulated in $\mathbb{F}_r$ for each scalar multiplication that the SNARK or folding verifier does.

In the algorithm given in Section 6.1.2, each of the $m \approx \ell/h$ algorithm steps requires h point doubling. That is at least $5(h + 1)$ multiplications in $\mathbb{F}_q$ (the 7 multiplications for point doubling are replaced with the number of

multiplications for addition, which is 5, to get a lower bound). In total, it means that the number of non-native field multiplications in standard IVC is $m*5(h+1)/5 = m(h+1)$ times the amount in the construction. If $\ell = 254$ and $h = 4$, then $m = 63$, it means 315 times more $\mathbb{F}_q$ multiplications to be emulated in $\mathbb{F}_r$ compared to standard IVC. In the case of successive scalar multiplications that are then treated as a single MSM, $m(h+1)$ should be replaced with the amortised number of $\mathbb{F}_q$ multiplications per scalar multiplication.

A more precise estimation of the performance gain is as follows. Consider an MSM of size $\nu \geq 1$. Denote by

$N_{\mathbb{F}_r}(\mathbb{F}_q)$ the number of gates in the Plonk four-column execution trace of an $\mathbb{F}_r$ circuit to emulate a multiplication in $\mathbb{F}_q$

$N_{\mathbb{F}_r}(\mathbb{G})$ the number of gates in the execution trace of an $\mathbb{F}_r$ circuit to emulate an operation (addition or point doubling) in $\mathbb{G}$

$m(\nu, \ell, h)$ denotes the number of steps in a given algorithm to compute an MSM of size ν with ℓ-bit scalars represented in base 2^h

$N_{\mathbb{G}}(\nu, \ell, h)$: the number of group operations per step of the same algorithm to compute an MSM of size ν with ℓ-bit scalars represented in base 2^h

$N_{\mathbb{F}_q}(\nu * (\mathbb{F}_r \cdot \mathbb{G}), \ell, h)$ the number of gates in the execution trace of an $\mathbb{F}_q$ circuit to emulate an MSM of size ν, with ℓ-bit scalars represented in base 2^h, for the same MSM algorithm

$N_{\mathbb{F}_q}(\mathbb{G})$ the number of gates in the execution trace of an $\mathbb{F}_q$ to emulate an operation in $\mathbb{G}$.

Note that the number of field and group operations that the prover performs at the end of an IVC scales at least linearly with the number of gates in the execution trace.

In the construction, the total number of execution-trace gates in $\mathbb{F}_r$ that an MSM of size ν incurs is at most $\nu * 5 * N_{\mathbb{F}_r}(\mathbb{F}_q)$ for the proof that A_q is correctly computed from A_r, and $N_{\mathbb{F}_q}(\nu * (\mathbb{F}_r \cdot G))$ for the proof of the MSM in the $\mathbb{F}_q$ circuit. That is at most

$$\nu * 5 * N_{\mathbb{F}_r}(\mathbb{F}_q) + N_{\mathbb{F}_q}(\nu * (\mathbb{F}_r \cdot \mathbb{G}), \ell, h)$$

gates in total.

In standard IVC, for the same MSM algorithm, the total number of gates in an $\mathbb{F}_r$ circuit is

$$m(\nu, \ell, h) * N_{\mathbb{G}}(\nu, \ell, h) * N_{\mathbb{F}_q}(\mathbb{G}) * N_{\mathbb{F}_r}(\mathbb{F}_q).$$

As an example to get a numerical estimation, let the MSM algorithm simply consist of ν executions of the algorithm in Section 6.1.2. In that algorithm, $m(\nu, \ell, h) = \nu\lfloor \ell - 1/h \rfloor \approx \nu\ell/h$, $N_{\mathbb{G}}(\nu, \ell, h) = (h+1)$ (there are h point doublings and an addition at each step) and $N_{\mathbb{F}_q}(\nu * (\mathbb{F}_r \cdot \mathbb{G}), \ell, h) = \nu N_{\mathbb{F}_q}(1 * (\mathbb{F}_r \cdot \mathbb{G}), \ell, h)$. Standard IVC requires at least

$$\frac{\ell/h * (h+1) * N_{\mathbb{F}_q}(\mathbb{G}) * \mathbb{N}_{\mathbb{F}_r}(\mathbb{F}_q)}{5 * N_{\mathbb{F}_r}(\mathbb{F}_q) + N_{\mathbb{F}_q}(\mathbb{F}_r \cdot \mathbb{G}, \ell, h)}$$

more gates than the construction the IVC with auxiliary proof information.

In practice, $N_{\mathbb{F}_q}(\mathbb{G})$ is around 2 (one row for the inputs to the operation and one row for the output), $\mathbb{N}_{\mathbb{F}_r}(\mathbb{F}_q)$ is around 10, and $N_{\mathbb{F}_q}(\mathbb{F}_r \cdot \mathbb{G}, \ell, h)$ is approximately $m(\ell, h) \approx \ell/h$. For $\ell \leftarrow 254 \approx 256$ and $h \leftarrow 4$, the previous ratio is at least 56. For $\nu = 64$, that is 7296 gates versus 409600. For an optimised MSM algorithm, although the ratio would definitely be lower, the number of constraints in standard IVC would still be at least an order of magnitude higher than in the construction with auxiliary proof information.

Acknowledgements. Many thanks to Liam Eagen, Luke Edwards and Ariel Gabizon for helpful discussions and corrections. The first author contributed while still at Aztec Labs.

References

1. Bagherzandi, A., Cheon, J.H., Jarecki, S.: Multisignatures secure under the discrete logarithm assumption and a generalized forking lemma. In: Ning, P., Syverson, P.F., Jha, S. (eds.) ACM CCS 2008. pp. 449–458. ACM Press (Oct 2008). https://doi.org/10.1145/1455770.1455827
2. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Scalable zero knowledge with no trusted setup. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019, Part III. LNCS, vol. 11694, pp. 701–732. Springer, Cham (Aug 2019). https://doi.org/10.1007/978-3-030-26954-8_23
3. Bitansky, N., Canetti, R., Chiesa, A., Tromer, E.: Recursive composition and bootstrapping for SNARKS and proof-carrying data. In: Boneh, D., Roughgarden, T., Feigenbaum, J. (eds.) 45th ACM STOC. pp. 111–120. ACM Press (Jun 2013). https://doi.org/10.1145/2488608.2488623
4. Bünz, B., Chen, B.: Protostar: Generic efficient accumulation/folding for special-sound protocols. In: Guo, J., Steinfeld, R. (eds.) ASIACRYPT 2023, Part II. LNCS, vol. 14439, pp. 77–110. Springer, Singapore (Dec 2023). https://doi.org/10.1007/978-981-99-8724-5_3
5. Bünz, B., Chiesa, A., Lin, W., Mishra, P., Spooner, N.: Proof-carrying data without succinct arguments. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021, Part I. LNCS, vol. 12825, pp. 681–710. Springer, Cham, Virtual Event (Aug 2021). https://doi.org/10.1007/978-3-030-84242-0_24
6. Bünz, B., Chiesa, A., Mishra, P., Spooner, N.: Recursive proof composition from accumulation schemes. In: Pass, R., Pietrzak, K. (eds.) TCC 2020, Part II. LNCS, vol. 12551, pp. 1–18. Springer, Cham (Nov 2020). https://doi.org/10.1007/978-3-030-64378-2_1
7. Eagen, L., Gabizon, A.: ProtoGalaxy: Efficient ProtoStar-style folding of multiple instances. Cryptology ePrint Archive, Report 2023/1106 (2023), https://eprint.iacr.org/2023/1106
8. Eagen, L., Gabizon, A., Sefranek, M., Towa, P., Williamson, Z.J.: Stackproofs: Private proofs of stack and contract execution using protogalaxy. Cryptology ePrint Archive, Paper 2024/1281 (2024), https://eprint.iacr.org/2024/1281
9. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) CRYPTO'86. LNCS, vol. 263, pp. 186–194. Springer, Berlin, Heidelberg (Aug 1987). https://doi.org/10.1007/3-540-47721-7_12
10. Gabizon, A., Williamson, Z.J., Ciobotaru, O.: PLONK: Permutations over Lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953 (2019), https://eprint.iacr.org/2019/953
11. Galbraith, S.D., Lin, X., Scott, M.: Endomorphisms for faster elliptic curve cryptography on a large class of curves. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 518–535. Springer, Berlin, Heidelberg (Apr 2009). https://doi.org/10.1007/978-3-642-01001-9_30
12. Gallant, R.P., Lambert, R.J., Vanstone, S.A.: Faster point multiplication on elliptic curves with efficient endomorphisms. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 190–200. Springer, Berlin, Heidelberg (Aug 2001). https://doi.org/10.1007/3-540-44647-8_11
13. Kate, A., Zaverucha, G.M., Goldberg, I.: Constant-size commitments to polynomials and their applications. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 177–194. Springer, Berlin, Heidelberg (Dec 2010). https://doi.org/10.1007/978-3-642-17373-8_11

14. Kohrita, T., Towa, P.: Zeromorph: Zero-knowledge multilinear-evaluation proofs from homomorphic univariate commitments. Cryptology ePrint Archive, Report 2023/917 (2023), https://eprint.iacr.org/2023/917
15. Kothapalli, A., Setty, S.: SuperNova: Proving universal machine executions without universal circuits. Cryptology ePrint Archive, Report 2022/1758 (2022), https://eprint.iacr.org/2022/1758
16. Kothapalli, A., Setty, S.: CycleFold: Folding-scheme-based recursive arguments over a cycle of elliptic curves. Cryptology ePrint Archive, Report 2023/1192 (2023), https://eprint.iacr.org/2023/1192
17. Kothapalli, A., Setty, S., Tzialla, I.: Nova: Recursive zero-knowledge arguments from folding schemes. In: Dodis, Y., Shrimpton, T. (eds.) CRYPTO 2022, Part IV. LNCS, vol. 13510, pp. 359–388. Springer, Cham (Aug 2022). https://doi.org/10.1007/978-3-031-15985-5_13
18. Kothapalli, A., Setty, S.T.V.: HyperNova: Recursive arguments for customizable constraint systems. In: Reyzin, L., Stebila, D. (eds.) CRYPTO 2024, Part X. LNCS, vol. 14929, pp. 345–379. Springer, Cham (Aug 2024). https://doi.org/10.1007/978-3-031-68403-6_11
19. Motwani, R., Raghavan, P.: Randomized Algorithms. Cambridge International Series on Parallel Computation, Cambridge University Press (1995)
20. Nguyen, W.D., Datta, T., Chen, B., Tyagi, N., Boneh, D.: Mangrove: A scalable framework for folding-based SNARKs. In: Reyzin, L., Stebila, D. (eds.) CRYPTO 2024, Part X. LNCS, vol. 14929, pp. 308–344. Springer, Cham (Aug 2024). https://doi.org/10.1007/978-3-031-68403-6_10
21. Pippenger, N.: On the evaluation of powers and related problems. In: 17th Annual Symposium on Foundations of Computer Science (sfcs 1976). pp. 258–263 (1976). https://doi.org/10.1109/SFCS.1976.21
22. Setty, S., Thaler, J., Wahby, R.: Customizable constraint systems for succinct arguments. Cryptology ePrint Archive, Paper 2023/552 (2023), https://eprint.iacr.org/2023/552, https://eprint.iacr.org/2023/552
23. Silverman, J.: The Arithmetic of Elliptic Curves. Graduate Texts in Mathematics, Springer New York (2009), https://books.google.ch/books?id=Z9OCA_EUCCkC
24. Soukhanov, L.: WARPfold : Wrongfield ARithmetic for protostar folding. Cryptology ePrint Archive, Report 2024/354 (2024), https://eprint.iacr.org/2024/354
25. Straus, E.G.: Addition chains of vectors (problem 5125). American Mathematical Monthly **70**(806-808), 16 (1964)
26. Valiant, P.: Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In: Canetti, R. (ed.) TCC 2008. LNCS, vol. 4948, pp. 1–18. Springer, Berlin, Heidelberg (Mar 2008). https://doi.org/10.1007/978-3-540-78524-8_1
27. Yen, S.M., Laih, C.S., Lenstra, A.K.: Multi-exponentiation. IEE proceedings: computers and digital techniques **141**(6), 325–326 (1994)
28. Zhang, Y., Genkin, D., Katz, J., Papadopoulos, D., Papamanthou, C.: A zero-knowledge version of vSQL. Cryptology ePrint Archive, Report 2017/1146 (2017), https://eprint.iacr.org/2017/1146

A Preliminaries

A.1 Proof Systems

The properties that proof systems should satisfy are formally defined as follows.

A.1.1 Completeness. A proof system for a relation-generator R is *complete* if for all $\lambda \in \mathbb{N}_{\geq 1}$, for all R in the range of $\mathrm{R}\left(1^{\lambda}\right)$, for all (par, τ) in the range of $\textsc{Setup}(R)$, for all (x, w) in R,

$$P\left[\textsc{Vf}\left(par, x, \textsc{Prove}(par, x, w)\right) = 1\right] = 1.$$

A.1.2 Knowledge Soundness. A proof system is knowledge sound with error κ, function with values in $(0, 1)$, if there exist a real polynomial p and an extractor E, which runs in expected polynomial time (on the size of its inputs), such that for any PPT algorithm $\textsc{Prove}^*$ to which E is given black-box rewinding access, for all $\lambda \in \mathbb{N}_{\geq 1}$, for any R in the range of $\mathrm{R}\left(1^{\lambda}\right)$ and any (par, τ) in the range of $\textsc{Setup}(R)$, for any (potentially unbounded) algorithm A, for any $(st \in \{0,1\}^*, x) \leftarrow \mathrm{A}(par)$ such that

$$P\left[\textsc{Vf}_{(\textsc{Prove}^*(st),\cdot)}(par, x) = 1\right] > \kappa(\lambda, |x|),$$

the inequality

$$\begin{aligned} &P\left[(x, w) \in R \colon w \leftarrow \mathrm{E}^{(\textsc{Prove}^*(st),\cdot)}(par, \tau, x)\right] \\ &\geq \left(P\left[\textsc{Vf}_{(\textsc{Prove}^*(st),\cdot)}(par, x) = 1\right] - \kappa(\lambda, |x|)\right) / p(\lambda, |x|) \end{aligned}$$

holds. The notation $\mathrm{E}^{(\textsc{Prove}^*(st),\cdot)}$ means that E has rewinding black-box access to $\textsc{Prove}^*$, which runs on some state st to which E does *not* have access. It is further assumed that E returns $w \neq \varnothing$ only if it is such that $(x, w) \in R$. The reason is that from any extractor E that satisfies the condition, one can always another extractor which tests whether the witness E computed is valid and returns $\varnothing$ if that is not the case. This new extractor has the same extraction probability as E does.

It is in fact sufficient to show that the condition on the extraction probability holds for any deterministic polynomial-time algorithm $\textsc{Prove}^*$. That is because for any PPT algorithm $\textsc{Prove}^*$, its success probability is the expectation, over the choice of its random string, of the success probability of $\textsc{Prove}^*$ running on a fixed random string. If the condition holds for any deterministic algorithm, it holds for $\textsc{Prove}^*$ running on any fixed random string, and therefore for the probabilistic algorithm $\textsc{Prove}^*$.

A.2 Folding Schemes

This section gives formal definitions for the properties that folding schemes should satisfy.

A.2.1 Completeness. A folding scheme for a relation-generator R is *complete* if for all $\lambda \in \mathbb{N}_{\geq 1}$, for all R in the range of $\mathrm{R}\left(1^{\lambda}\right)$, for all *par* in the range of $\textsc{Setup}(R)$, for all $(x_0, w_0), (x_1, w_1)$ in R, with (W, X) denoting

$$(\textsc{Prove}(par, (x_0, x_1), (w_0, w_1)), \textsc{Vf}(par, (x_0, x_1)),$$

the random outputs of Prove and Vf at the end of the interaction,

$$P\left[(X, W) \in R = 1\right].$$

A.2.2 Knowledge Soundness. A folding scheme is knowledge sound with error κ, function with values in $(0, 1)$, if there exist a real polynomial p and an extractor E, which runs in expected polynomial time (on the size of its inputs), such that for any PPT algorithm $\textsc{Prove}^*$ to which E is given black-box rewinding access, for all $\lambda \in \mathbb{N}_{\geq 1}$, for any R in the range of $\mathrm{R}\left(1^{\lambda}\right)$ and any *par* in the range of $\textsc{Setup}(R)$, for any (potentially unbounded) algorithm A, for any $((st \in \{0,1\}^*, (x_0, x_1)) \leftarrow \mathrm{A}(par)$ such that $P\left[(x, w) \in R\right] > \kappa(\lambda, |(x_0, x_1)|)$, the inequality

$$\begin{aligned} &P\left[(x_0, w_0), (x_1, w_1) \in R \colon (w_0, w_1) \leftarrow \mathrm{E}^{(\textsc{Prove}^*(st), \cdot)}(par, (x_0, x_1))\right] \\ &\geq \left(P\left[(X, W) \in R\right] - \kappa(\lambda, |(x_0, x_1)|)\right) / p(\lambda, |(x_0, x_1)|) \end{aligned}$$

holds. The pair (W, X) denotes

$$(\textsc{Prove}(par, (x_0, x_1), (w_0, w_1)), \textsc{Vf}(par, (x_0, x_1)),$$

the random outputs of Prove and Vf at the end of the interaction. The notation

$$\mathrm{E}^{(\textsc{Prove}^*(st), \cdot)}$$

means that E has rewinding black-box access to $\textsc{Prove}^*$, which runs on some state *st* to which E does *not* have access. It is further assumed that E returns $(w_0, w_1) \neq \varnothing$ only if it is such that (x_0, w_0) and (x_1, w_1) are in R. The reason is similar to that for the case of proof systems.

A.3 Chernoff Bound

The Chernoff bound gives bound on the tail distribution of sums of independent Bernoulli random variables. Let $X_1, X_2, \ldots, X_n$ be independent Bernoulli random variables such that, for $1 \leq i \leq n$, $P[X_i = 1] = p_i$, with $0 < p_i < 1$. Then [19, Theorem 4.2], for $X = \sum_{i=1}^{n} X_i$, $\mathbb{E}[X] = \sum_{i=1}^{n} p_i$, and any $0 < \delta \leq 1$,

$$P\left[X < (1 - \delta)\mathbb{E}[X]\right] < \exp\left(-\mathbb{E}[X]\delta^2/2\right).$$

B Definitions

This section gives formal definitions for the schemes defined in Section 3.

B.1 Incrementally Verifiable Computation with Auxiliary Proof Information

B.1.1 Completeness. An IVC scheme with auxiliary proof information for a generator G is *complete* if for all $\lambda \in \mathbb{N}_{\geq 1}$, for all $\left(R, \left(R_{aux}^{(i)}\right)_{i\geq 1}\right)$ in the range of $\mathrm{G}\left(1^{\lambda}\right)$ and all *par* in the range of $\textsc{Setup}\left(R, \left(R_{aux}^{(i)}\right)_{i\geq 1}\right)$, for all $((x_0, x_1), w_0)$ in R, with (Π_1, AUX_1, ST_1) denoting the random variable

$$\textsc{Prove}\left(par, (0, x_0, x_1), w_0\right),$$

$$P\left[\textsc{Vf}\left(par, (1, x_0, x_1), (\Pi_1, AUX_1)\right) = 1\right] = 1$$

and $(AUX_1, ST_1) \in R_{aux}^{(1)}$; and if for all $(i, x_0, x_i, \pi_i, aux_i)$ with i positive such that

$$\textsc{Vf}(par, (i, x_0, x_i), \pi_i, aux_i) = 1,$$

for all x_{i+1}, w_i such that $((x_i, x_{i+1}), w_i)$ is in R, with $(\Pi_{i+1}, AUX_{i+1}, ST_{i+1})$ denoting $\textsc{Prove}\left(par, (i, x_0, x_{i+1}), x_i, \pi_i, aux_i, st_i, w_i\right)$,

$$P\left[\textsc{Vf}\left(par, (i+1, x_0, x_{i+1}), (\Pi_{i+1}, AUX_{i+1})\right) = 1\right] = 1,$$

and $(AUX_{i+1}, ST_{i+1}) \in R_{aux}^{(i+1)}$ if $(aux_i, st_i) \in R_{aux}^{(i)}$.

B.1.2 Knowledge Soundness. An IVC scheme with auxiliary proof information for a generator G is *knowledge-sound* with error κ, function with values in $(0, 1)$, if there exist a real polynomial p and an extractor E, which runs in expected polynomial time (on the size of its inputs), such that for any PPT algorithm $\textsc{Prove}^*$ to which E is given rewinding black-box access, for all $\lambda \in \mathbb{N}_{\geq 1}$, for all $\left(R, \left(R_{aux}^{(i)}\right)_{i\geq 1}\right)$ in the range of $\mathrm{G}\left(1^{\lambda}\right)$ and all *par* in the range of $\textsc{Setup}\left(R, \left(R_{aux}^{(i)}\right)_{i\geq 1}\right)$, for any (potentially unbounded) algorithm A, for any

$$(st \in \{0,1\}^*, (n+1, x_0, x_{n+1})) \leftarrow \mathrm{A}(par)$$

with n positive such that,

$$\begin{aligned} &P\left[\textsc{Vf}\left(par, (n+1, x_0, x_{n+1}), (\Pi^*, AUX^*)\right) = 1 \mid AUX^* \in L_{aux}^{(n+1)}\right] \\ &> \kappa(\lambda, |(n+1, x_0, x_{n+1})|), \end{aligned}$$

the inequality

$$\begin{aligned} &P\left[((x_0, x_1), w_0), \ldots, ((x_n, x_{n+1}), w_n) \in R : \right. \\ &\left.(w_0, x_1, w_1, \ldots, x_n, w_n) \leftarrow \mathrm{E}^{\textsc{Prove}^*(st,\cdot)}(par, (n+1, x_0, x_{n+1})) \mid AUX^* \in L_{aux}^{(n+1)}\right] \\ &\geq \Big(P\left[\textsc{Vf}\left(par, (n+1, x_0, x_{n+1}), (\Pi^*, AUX^*)\right) = 1 \mid AUX^* \in L_{aux}^{(n+1)}\right] \\ &\quad - \kappa(\lambda, |(n+1, x_0, x_{n+1})|)\Big) / p(\lambda, |(n+1, x_0, x_{n+1})|) \end{aligned}$$

holds. The triple (Π^*, AUX^*, ST^*) denotes the random variable

$$\textsc{Prove}^*(st, (n+1, x_0, x_{n+1})).$$

The notation $\mathrm{E}^{(\textsc{Prove}^*(st),\cdot)}$ means that E has rewinding black-box access to $\textsc{Prove}^*$, which runs on some state st to which E does *not* have access. Extractor E does *not* have access to the ST^* output of $\textsc{Prove}^*$ either. That is because the only additional capabilities that E should have compared to $\textsc{Vf}$ is to have access to the random tape of $\textsc{Prove}^*$. As $\textsc{Vf}$ is not given access to the ST^* output of $\textsc{Prove}^*$, the extractor should not have access to it either.

For E to run in time polynomial in the size of its inputs, $n+1$ should be encoded in unary as E requires at least that many elementary steps to write its output. It is further assumed that E returns an output distinct from $\varnothing$ only if the intermediate IVC outputs and witnesses are valid. Indeed, from any extractor E that satisfies the condition, one can always define another extractor which tests whether the outputs of E are valid intermediate IVC outputs and witnesses and returns $\varnothing$ if that is not the case. This new extractor has the same extraction probability as E does.

The definition fundamentally states that for any tuple $(n+1, x_0, x_{n+1})$ that algorithm A returns and for which algorithm $\textsc{Prove}^*$ can compute a valid proof with a probability above the soundness-error threshold, the extractor can compute a sequence of intermediate computation results together with valid witnesses if the auxiliary proof information is valid. It does so with a probability that is close, up to a polynomial factor, to the probability that $\textsc{Prove}^*$ computes a proof with auxiliary information that passes verification. Note that the definition requires successful extraction only if the auxiliary proof information is valid: a relaxation compared to standard IVC schemes.

Similarly to the case of proof systems (see Appendix A.1), it is sufficient to prove a lower bound on the probability to extract intermediate IVC outputs and witnesses from deterministic algorithms that return a piece of information aux^* in $L_{aux}^{(n+1)}$, instead of proving a lower bound on probabilistic algorithms. Indeed, for any PPT algorithm $\textsc{Prove}^*$, its success probability conditioned on the event $aux^* \in L_{aux}^{(n+1)}$ is the expectation, over the choice of its random strings that lead to $aux \in L_{aux}^{(n+1)}$, of the success probability of $\textsc{Prove}^*$ running on a fixed such random string.