Privacy and Scalability for Decentralized Cryptographic Systems

Pratyush Mishra

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2021-214
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2021/EECS-2021-214.html

September 7, 2021

Privacy and Scalability for Decentralized Cryptographic Systems

by

Pratyush Mishra

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Associate Professor Alessandro Chiesa, Co-chair
Associate Professor Raluca Ada Popa, Co-chair
Associate Professor Sanjam Garg
Associate Professor Matthew Green

Fall 2021

Privacy and Scalability for Decentralized Cryptographic Systems

Abstract

Privacy and Scalability for Decentralized Cryptographic Systems

by

Pratyush Mishra

Doctor of Philosophy in Computer Science

University of California, Berkeley

Associate Professor Alessandro Chiesa, Co-chair

Associate Professor Raluca Ada Popa, Co-chair

The past few years have seen growing interest in decentralized systems owing to their improved censorship-resistance, fault tolerance, and auditability compared to their centralized counterparts. For example, ideas popularized in decentralized protocols like Bitcoin and Ethereum have seen widespread adoption and publicity. However, the benefits of these systems often come at the expense of privacy and scalability: to ensure the correctness of computations, decentralized systems like Ethereum require that parties publish their entire computational state, which is then checked by re-executing the computation. From the privacy perspective, this reveals which computation was performed, the data that was input to the computation, and the identity of the involved users. From the scalability perspective, re-execution means that the cost of expensive computations is borne by every party in the system, as opposed to just the party invoking the computation.

In this dissertation, we show how to overcome these shortcomings and obtain decentralized systems that achieve strong privacy and scalability properties. We do so by providing new constructions and applications of a powerful cryptographic primitive: zero-knowledge succinct non-interactive argument systems, or zkSNARKs. We design new methodologies for constructing zkSNARKs that have lower deployment overhead and improved efficiency compared to the prior state-of-the-art. Finally, we go on to construct a system for *decentralized private computation* that takes advantage of these advances to make all transactions indistinguishable, thus ensuring privacy (transactions reveal no information about the computation) and scalability (transactions can be verified in time independent of the computation).

To my family.

Contents

List of Figures

List of Tables

Acknowledgments

I owe the existence of this thesis, and any success I have had as a researcher, to my wonderful advisors, Alessandro Chiesa and Raluca Ada Popa. They have invested an immense amount of faith and effort in me, and have guided me every step of the way in my journey from an undergraduate student without any experience in cryptographic research to a PhD candidate with a 260-page(!) thesis. Their efforts have greatly improved every aspect of my research skills and philosophy, ranging from picking good research problems, to thinking about how to solve them, to writing clear and rigorous papers, and finally to presenting our research. Ale, I particularly appreciate our frequent meetings (including our many spontaneous late-night calls), and your detailed feedback on even the most minute aspects of research. Raluca, I am very grateful for how you always pushed me to maintain a high standard of research and to take the next step forward, whether that was pushing out a quality implementation or fearlessly applying for fellowships. I would be more than happy if I could pass on to my students even a fraction of the things that I have learnt from the two of you.

I am very grateful to David Wagner, who started me on my research journey as an oblivious sophomore, when I knew very little about computer science, let alone computer security research. His kind words and fantastic teaching ensured that I stuck with research in those early days. I would also like to thank Yan X. Zhang; it was his wonderful teaching in Math 116 that highlighted to me, for the first time, the wonders of the inner workings of cryptography. Math 116 remains to this day the best cryptography course I have taken.

None of my research would be possible without my amazing co-authors: Sean Bowe, Benedikt Bünz, Jerry Chen, Alessandro Chiesa, Matthew Green, Yuncong Hu, Ryan Lehmkuhl, William Lin, Jingcheng Liu, Mary Maller, Peihan Miao, Ian Miers, Rishabh Poddar, Raluca Ada Popa, Nick Spooner, Akshayaram Srinivasan, Nirvan Tyagi, Nicholas Ward, Howard Wu, Psi Vesely, Yinuo Zhang, and Wenting Zheng. I have learnt so much from all of you, including, perhaps most importantly, how to make friends in academia. Special thanks to Matt, who very kindly hosted me for a research visit to JHU. I would also like to thank everybody that has contributed to `arkworks`; it has been exhilarating to see it grow from its cobbled-together beginnings as a Zexe implementation to something that people enjoy using. I would especially like to thank Weikeng Chen, Alessandro Chiesa, Emma Dauterman Kobi Gurkan, Yuncong Hu, Ryan Lehmkuhl, William Lin, Dev Ojha, Tom Shen, Nicholas Ward, and Howard Wu.

My PhD experience has been immensely enriched by the kind and caring friends I made in the Berkeley Security group and the RISE Lab: Angie, Austin, Frank, Grant, Nathan, Rishabh, and Wenting. To the inhabitants of 719/721 Soda Hall, I wonder to this day how we got any work done in the office. Also, the Japan trip was one of the most wonderful, fun, and memorable experiences of my life. A special shout-out to Grant, Rishabh, and Wenting for being amazing gymming, cooking, eating, board gaming, and politics-and-books-and-movies-and-music-discussing buddies.

I felt quite lonely leaving my family behind to move across the globe to come to Berkeley, but Adithya, Aparna, Dhruv, Kevin, Krishna, Ori, and Radhika helped make Berkeley my home away from home. A very special thanks to Aparna, who lovingly and patiently tolerated every aspect of dating a PhD student. My transition to the Bay Area was also made much easier by my aunt, uncle, Reyansh, and Hridhaan, who have always welcomed me into their home with open arms.

Finally, I dedicate this dissertation to my family. They have supported me every step of the way, and left no stone unturned to make sure that I could pursue my heart's desires. Their unwavering love, support, and faith have made all of this possible and worthwhile.

Chapter 1

Introduction

Today, the vast majority of digital services are provided via *centralized systems* which concentrate decision-making power and ability in the hands of a few parties. This centralization brings potential benefits such as lower implementation complexity and better efficiency. However, centralization also brings drawbacks, such as poor censorship-resistance, poor fault tolerance, a lack of transparency and auditability. Due to these shortcomings, the last few years have seen growing interest in developing decentralized applications atop *decentralized ledger systems* that allow a set of heterogeneous parties to come to consensus on the validity of general computations over arbitrary data. Such systems include Bitcoin [Nak09], Ethereum [Woo17], Polkadot [Woo16], Solana [Yak18], and more. These systems collectively validate thousands of program executions each day without relying on a central party.

However, these benefits often come at the expense of two key properties: **privacy** and **scalability**. In more detail, to ensure the correctness of computations, existing decentralized ledger systems require that parties publish their entire computational state, which is then checked by re-executing the computation. From the privacy perspective, this reveals (a) the computation that was performed, (b) the data that was input to the computation, and (c) the identity of the users involved in the computation. From the scalability perspective, this means that the cost of expensive computations is borne by every party in the system, as opposed to just the party invoking the computation. Counteracting this requires metering mechanisms, but these can themselves lead to inconsistencies and attacks [Bit15; Eth16].

In this dissertation, we show how to resolve this dilemma and obtain permissionless, censorship-resistant, and auditable decentralized systems that do not compromise on privacy and scalability. We do this by providing new constructions and applications of a powerful cryptographic primitive: succinct zero-knowledge non-interactive argument systems, or zkSNARKs.

Primer on zkSNARKs. Informally, a zkSNARK for an NP relation $\mathcal{R}$ is a type of zero knowledge proof system [GMR89] that allows a prover to quickly convince a verifier, by means of a short proof string π, that it knows a witness $\mathbb{w}$ corresponding to an instance $\mathbb{x}$, so that $(\mathbb{x}, \mathbb{w}) \in \mathcal{R}$. The proof π achieves *zero-knowledge*, which means it hides all information about $\mathbb{w}$, and *succinctness*, which means that it is much shorter than $\mathbb{w}$. (Optionally, the verifier's run time can additionally be much shorter than the time required to check membership in $\mathcal{R}$.)

zkSNARKs in decentralized ledgers. The zero-knowledge and succinctness properties of zkSNARKs have made them an attractive tool for developers of decentralized ledger systems. Indeed, zkSNARKs have already been used to improve the privacy [Ben+14; KMSWP16] as well as scalability [Eth21; Whi18; Mina; KB20; BMRS20; CCDW20] of these systems. However, limitations of existing zkSNARKs, in terms of both efficiency and operational concerns, have constrained their deployment of these applications.

In the rest of this thesis, we first construct novel protocols (Chapters 2 and 3) that avoid these limitations, and then construct a system (Chapter 4) that can leverage these protocols to provide better privacy and scalability guarantees than existing decentralized ledger systems. Below we provide a high-level overview of each of these.

1.1 Marlin: zkSNARKs with universal and updatable SRS

Among the various constructions of zkSNARKs, *preprocessing SNARKs* [Gro10; Lip12; GGPR13; BCIOP13] have attracted the attention of researchers and practitioners due to their ability to achieve succinct verification for arbitrary (non-uniform) computations. However, most efficient preprocessing SNARKs rely on sampling a structured reference string (SRS) that is circuit-dependent: it changes with the circuit. This means that each new circuit requires a separate SRS. In applications to decentralized systems, there is no single party that can be trusted with sampling the SRS. (Indeed, the point of such systems is to *eliminate* such parties). Hence, real-world deployments have had to rely on cryptographic "ceremonies" [ZcashMPC; BCGTV15; BGG17; BGM17; ABLSZ19; KMSV21] to generate the SRS in a distributed manner. These ceremonies require a non-trivial implementation and deployment effort which must be repeated for each distinct circuit.

To solve this problem, in Chapter 2, we develop a novel methodology to construct SNARKs that have a *universal*, or circuit-*independent* SRS. This latter can be specialized into a circuit-specific SRS in a publicly auditable manner by anyone, thus reducing the trust requirements on the SRS generation process. We then develop novel ingredients that we plug into our methodology to obtain Marlin, a universal SRS SNARK that is competitive with the state-of-the-art circuit-specific zkSNARK [Gro16] in all relevant metrics.

1.2 Proof-carrying data without succinct arguments

zkSNARKs enable a single party to prove the correctness of a single step of computation. However, many usecases in decentralized systems require mutually distrustful parties to each perform different steps of a distributed computation. Proving the correctness of these computations requires *proof-carrying data* (PCD) [CT10], a more powerful primitive that enables these parties to perform the distributed computation in a manner that ensures the correctness of each step. While prior work [CT10; BCCT13; BCTV17; COS20; BCMS20] has provided a number of PCD constructions, all of these impose significant overhead due to their reliance on relatively heavy algebraic and cryptographic objects.

In Chapter 3, we propose a new methodology of constructing PCD that relies only on relatively lightweight cryptographic objects: *non-succinct* arguments. We then construct efficient ingredients for our methodology from simple cryptographic assumptions, and plug these in to obtain a concretely efficient PCD scheme that achieves the lowest PCD overhead compared to all prior schemes.

1.3 Zexe: Enabling decentralized private computation

In Chapter 4, we design, implement, and evaluate Zexe (*Zero knowledge EXEcution*), a ledger-based system that enables users to execute offline computations and subsequently produce publicly-verifiable transactions that attest to the correctness of these offline executions. Zexe simultaneously provides privacy and succinctness for these executions: a transaction reveals *no information* about the offline computation whose correctness it attests to, and, furthermore, can be validated in time that is *independent* of the cost of the computation.

Zexe achieves these strong privacy and scalability properties *without* sacrificing rich functionality. We illustrate this by using Zexe to construct privacy-preserving analogues of popular applications: private user-defined assets, private decentralized or non-custodial exchanges (DEXs), and private stablecoins.

To achieve an efficient implementation of Zexe, we draw upon advances in zkSNARKs and in recursive proof composition of these. Overall, transactions in Zexe with two input records and two output records are 968 bytes and can be verified in tens of milliseconds, *regardless of the offline computation*; generating these transactions takes less than a minute plus a time that grows with the offline computation (inevitably so). This implementation is achieved in a modular fashion via a collection of Rust libraries; see Section 1.4 for more details.

1.4 Impact and adoption

All the works in this thesis are seeing promising industrial adoption. Here we provide a short summary, and defer a complete discussion to Chapter 5. First, we developed our initial implementation of Zexe into `arkworks` [con], a state-of-the-art open source Rust ecosystem for zkSNARK development. `arkworks` has seen adoption across a number of industrial projects, and has attracted over fifty contributors that have collectively written over $120,000$ lines of Rust code. Next, Marlin is seeing deployment in existing projects [Aleo; HGD21], and has spurred novel academic research on universal SNARKs [COS20; SZ20; BCMS20; CFFQR20; HGD21; BCLMS21; RZ21; ABCGOT21; ZZWG21] as well as their applications [CCDW20; KZGM21]. Finally, the ideas behind Zexe are in the process of being adopted by a number of decentralized ledger projects [Aleo; Aztec; Mir; Anoma].

Chapter 2

MARLIN: zkSNARKs with Universal and Updatable SRS

In this chapter, we present a methodology to construct preprocessing zkSNARKs where the structured reference string (SRS) is universal and updatable. This exploits a novel use of *holography* [BFLS91], where fast verification is achieved provided the statement being checked is given in encoded form.

We use our methodology to obtain MARLIN, a preprocessing zkSNARK where the SRS has linear size and arguments have constant size. Our construction improves on Sonic [MBKM19], the prior state of the art in this setting, in all efficiency parameters: proving is an order of magnitude faster and verification is thrice as fast, even with smaller SRS size and argument size. Our construction is most efficient when instantiated in the algebraic group model (also used by Sonic), but we also demonstrate how to realize it under concrete knowledge assumptions. We implement and evaluate our construction.

The core of our preprocessing zkSNARK is an efficient *algebraic holographic proof* (AHP) for rank-1 constraint satisfiability (R1CS) that achieves linear proof length and constant query complexity.

This work was previously published in [CHMMVW20].

2.1 Introduction

Succinct non-interactive arguments (SNARGs) are efficient certificates of membership in non-deterministic languages. Recent years have seen a surge of interest in zero-knowledge SNARGs of knowledge (zkSNARKs), with researchers studying constructions under different cryptographic assumptions, improvements in asymptotic efficiency, concrete performance of implementations, and numerous applications. The focus of this work is *SNARGs in the preprocessing setting*, a notion that we motivate next.

When is fast verification possible? The size of a SNARG must be, as a minimum condition, sublinear in the size of the non-deterministic witness, and often is required to be even smaller (e.g., logarithmic in the size of the non-deterministic computation). The time to verify a SNARG would be, ideally, as fast as reading the SNARG. *This is in general too much to hope for, however.* The verification procedure must also read the *description* of the computation, in order know what statement is being verified. While there are natural computations that have succinct descriptions (e.g., machine computations), in general the description of a computation could be as large as the computation itself, which means that the time to verify the SNARG could be asymptotically comparable to the size of the computation. This is unfortunate because there is a very useful class of computations for which we cannot expect fast verification: general circuit computations.

The preprocessing setting. An approach to avoid the above limitation is to design a verification procedure that has two phases: an offline phase that produces a short summary for a given circuit; and an online phase that uses this short summary to verify SNARGs that attest to the satisfiability of the circuit with different partial assignments to its input wires. Crucially, now the online phase could in principle be as fast as reading the SNARG (and the partial assignment), and thus sublinear in the circuit size. This goal was captured by *preprocessing SNARGs* [Gro10; Lip12; GGPR13; BCIOP13], which have been studied in an influential line of works that has led to highly-efficient constructions that fulfill this goal (e.g., [Gro16]) and large-scale deployments in the real world that benefit from the online fast verification (e.g., [Zcash]).

The problem: circuit-specific SRS. The offline phase in efficient constructions of preprocessing SNARGs consists of sampling a structured reference string (SRS) that depends on the circuit that is being preprocessed. This implies that producing/validating proofs with respect to different circuits requires different SRSs. In many applications of interest, there is no single party that can be entrusted with sampling the SRS, and so real-world deployments have had to rely on cryptographic "ceremonies" [ZcashMPC] that use secure multi-party sampling protocols [BCGTV15; BGG17; BGM17; ABLSZ19; KMSV21]. However, any modification in the circuit used in an application requires another cryptographic ceremony, which is unsustainable for many applications.

A solution: universal SRS. The above motivates preprocessing SNARGs where the SRS is *universal*, which means that the SRS supports any circuit up to a given size bound by enabling anyone, in an offline phase *after* the SRS is sampled, to publicly derive a circuit-specific SRS.[1] Known techniques to obtain a universal SRS from circuit-specific SRS introduce expensive overheads due

[1] Even better than a universal SRS would be a URS (uniform reference string). However, achieving preprocessing SNARGs in the URS model with small argument size remains an open problem; see Section 2.1.2.

to universal simulation [BCTV14; BCTV17]. Also, these techniques lead to universal SRSs that are not *updatable*, a property introduced in [GKMMM18] that significantly simplifies cryptographic ceremonies. The recent work of Maller et al. [MBKM19] overcomes these shortcomings, obtaining the first efficient construction of a preprocessing SNARG with universal (and updatable) SRS. Even so, the construction in [MBKM19] is considerably more expensive than the state of the art for circuit-specific SRS [Gro16]. In this work we ask: *can the efficiency gap between universal SRS and circuit-specific SRS be closed, or at least significantly reduced?*

Concurrent work. A concurrent work [GWC19] studies the same question as this work. See Section 2.1.2 for a brief discussion that compares the two works.

construction	argument size over BN-256 (bytes)	argument size over BLS12-381 (bytes)
Sonic [MBKM19]	1152	1472
MARLIN [this work]	704	880
Groth16 [Gro16]	128	192

zkSNARK construction		sizes			time complexity			
		$\lvert\mathsf{ipk}\rvert$	$\lvert\mathsf{ivk}\rvert$	$\lvert\pi\rvert$	generator	indexer	prover	verifier
Sonic [MBKM19]	$\mathbb{G}_1$	$8m$	—	20	8 f-MSM(M)	4 v-MSM($3m$)	273 v-MSM(m)	7 pairings
	$\mathbb{G}_2$	$8m$	3	—	8 f-MSM(M)	—	—	
	$\mathbb{F}_q$	—	—	16	—	$O(m \log m)$	$O(m \log m)$	$O(\lvert\mathbb{x}\rvert + \log m)$
MARLIN [this work]	$\mathbb{G}_1$	$4m$	2	13	1 f-MSM($3M$)	12 v-MSM(m)	22 v-MSM(m)	2 pairings
	$\mathbb{G}_2$	—	2	—	—	—	—	
	$\mathbb{F}_q$	—	—	8	—	$O(m \log m)$	$O(m \log m)$	$O(\lvert\mathbb{x}\rvert + \log m)$
Groth16 [Gro16]	$\mathbb{G}_1$	$4n$	$O(\lvert\mathbb{x}\rvert)$	2	4 f-MSM(n)		4 v-MSM(n)	1 v-MSM($\lvert\mathbb{x}\rvert$)
	$\mathbb{G}_2$	n	$O(1)$	1	1 f-MSM(n)	N/A	1 v-MSM(n)	3 pairings
	$\mathbb{F}_q$	—	—	—	$O(m + n \log n)$		$O(m + n \log n)$	—

n: number of multiplication gates in the circuit
m: total number of (addition or multiplication) gates in the circuit
M: maximum supported circuit size (maximum number of addition and multiplication gates)

Figure 2.1: Comparison of two preprocessing zkSNARKs with universal (and updatable) SRS: the prior state of the art and our construction. We include the current state of the art for circuit-specific SRS (in gray), for reference. Here $\mathbb{G}_1$/$\mathbb{G}_2$/$\mathbb{F}_q$ denote the number of elements or operations over the respective group/field; also, f-MSM(m) and v-MSM(m) denote fixed-base and variable-base multi-scalar multiplications (MSM) each of size m, respectively. The number of pairings that we report for Sonic's verifier is lower than that reported in [MBKM19] because we account for standard batching techniques for pairing equations.

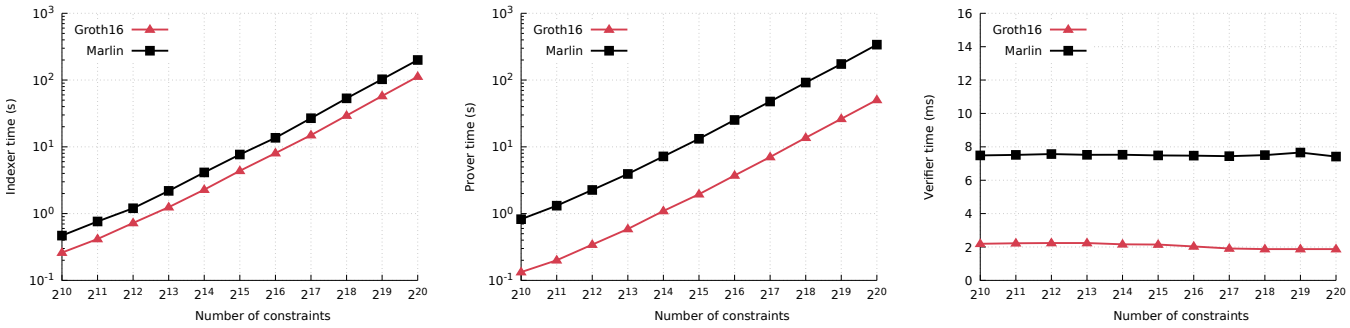

Figure 2.2: Measured performance of MARLIN and [Gro16] over the BLS12-381 curve. We could not include measurements for [MBKM19, Sonic] because at the time of writing there is no working implementation of its unhelped variant.

2.1.1 Our results

In this work we present MARLIN, a new preprocessing zkSNARK with universal (and updatable) SRS that improves on the prior state of the art [MBKM19, Sonic] in essentially all relevant efficiency parameters.[2] In addition to reducing argument size by several group and field elements and reducing time complexity of the verifier by over $3\times$, our construction overcomes the main efficiency drawback of [MBKM19, Sonic]: the cost of producing proofs. Indeed, our construction improves time complexity of the prover by over $10\times$, achieving prover efficiency comparable to the case of preprocessing zkSNARKs with *circuit-specific* SRS. In Fig. 2.1 we provide a comparison of our construction and [MBKM19, Sonic], including argument sizes for two popular elliptic curves; the table also includes the state of the art for circuit-specific SRS. We have implemented MARLIN in a Rust library,[3] and report evaluation results in Fig. 2.2.

Our zkSNARK is the result of several contributions that we deem of independent interest, summarized below.

(1) A new methodology. We present a general methodology to construct preprocessing SNARGs (and also zkSNARKs) where the SRS is universal (and updatable). The methodology in fact produces succinct *interactive* arguments that can be made non-interactive via the Fiat–Shamir transformation [FS86]. Hence below we focus on *preprocessing arguments with universal and updatable SRS* (see Section 2.7 for the definition).

Our key observation is that the ability to preprocess a circuit in an offline phase is closely related to constructing "holographic proofs" [BFLS91], which means that the verifier does not receive the circuit description as an input but, rather, makes a small number of queries to an encoding of it. These queries are in addition to queries that the verifier makes to proofs sent by the prover. Moreover, in this work we focus on the setting where the encoding of the circuit description consists of low-degree polynomials and also where proofs are themselves low-degree polynomials — this

[2] Maller et al. [MBKM19] discuss two variants of their protocol, a cheaper one for the "helped setting" and a costlier one for the "unhelped setting". The variant that is relevant to this work is the latter one, because it is a preprocessing zkSNARK. (The former variant does not achieve succinct verification, and instead achieves a weaker guarantee that applies to proof batches.)

[3] `https://github.com/arkworks-rs/marlin`

can be viewed as a requirement that honest and malicious provers are "algebraic". We call these *algebraic holographic proofs* (AHPs); see Section 2.4 for definitions.

We present a transformation that "compiles" any public-coin AHP into a corresponding preprocessing argument with universal (and updatable) SRS by using suitable polynomial commitments.

Theorem 1 (informal version of Theorem 2.8.1)**.** *There is an efficient transformation that combines any public-coin AHP for a relation $\mathcal{R}$ and an extractable polynomial commitment scheme to obtain a public-coin preprocessing argument with universal SRS for the relation $\mathcal{R}$. The transformation preserves zero knowledge and proof of knowledge of the underlying AHP. The SRS is updatable provided the SRS of the polynomial commitment scheme is.*

The above transformation provides us with a *methodology* to construct preprocessing zkSNARKs with universal SRS (see Fig. 2.3). Namely, to improve the efficiency of preprocessing zkSNARKs with universal SRS it suffices to improve the efficiency of *simpler building blocks*: AHPs (an information-theoretic primitive) and polynomial commitments (a cryptographic primitive).[4]

The improvements achieved by our preprocessing zkSNARK (see Fig. 2.1) were obtained by following this methodology: we designed efficient constructions for each of these two building blocks (which we discuss shortly), combined them via Theorem 1, and then applied the Fiat–Shamir transformation [FS86].

Methodologies that combine information-theoretic probabilistic proofs and cryptographic tools have played a fundamental role in the construction of efficient argument systems. In the particular setting of preprocessing SNARGs, for example, the compiler introduced in [BCIOP13] for *circuit-specific* SRS has paved the way towards current state-of-the-art constructions [Gro16], and also led to constructions that are plausibly post-quantum [BISW17; BISW18]. We believe that our methodology for universal SRS will also be useful in future work, and may lead to further efficiency improvements.

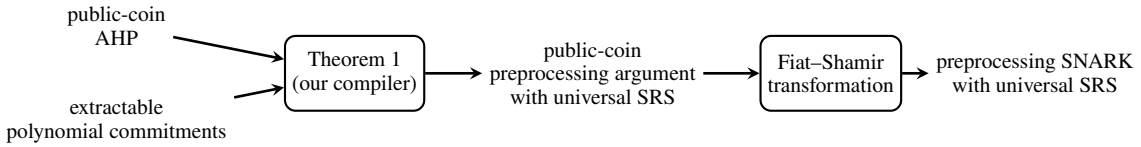

Figure 2.3: Diagram of our methodology to construct preprocessing SNARGs with universal SRS.

(2) An efficient AHP for R1CS. We design an algebraic holographic proof (AHP) that achieves linear proof length and constant query complexity, among other useful efficiency features. The protocol is for rank-1 constraint satisfiability (R1CS), a well-known generalization of arithmetic circuits where the "circuit description" is given by coefficient matrices (see definition below). Note that the relations that we consider consist of *triples* rather than *pairs*, because we need to split the

[4]The methodology also captures as a special case various folklore approaches used in prior works to construct *non*-preprocessing zkSNARKs via polynomial commitment schemes (see Section 2.1.2), thereby providing the first formal statement that clarifies what properties of algebraic proofs and polynomial commitment schemes are essential for these folklore approaches.

verifier's input into a part for the offline phase and a part for the online phase. The offline input is called the *index*, and it consists of the coefficient matrices; the online input is called the *instance*, and it consists of a partial assignment to the variables. The algorithm that encodes the index (coefficient matrices) in the offline phase is called the *indexer*.

Definition 1 (informal)**.** *The* **indexed relation** $\mathcal{R}_{\mathrm{R1CS}}$ *is the set of triples* $(\mathbb{i}, \mathbb{x}, \mathbb{w}) = \big((\mathbb{F}, n, m, A, B, C), x, w\big)$ *where* $\mathbb{F}$ *is a finite field,* A, B, C *are* $n \times n$ *matrices over* $\mathbb{F}$*, each containing at most* m *non-zero entries, and* $z := (x, w)$ *is a vector in* $\mathbb{F}^n$ *such that* $Az \circ Bz = Cz$*. (Here "*$\circ$*" denotes the entry-wise product.)*

Theorem 2 (informal)**.** *There exists a constant-round AHP for the indexed relation* $\mathcal{R}_{\mathrm{R1CS}}$ *with linear proof length and constant query complexity. The soundness error is* $O(m/|\mathbb{F}|)$*, and the construction is a zero knowledge proof of knowledge. The arithmetic complexity of the indexer is* $O(m \log m)$*, of the prover is* $O(m \log m)$*, and of the verifier is* $O(|x| + \log m)$*.*

The literature on probabilistic proofs contains algebraic protocols that are holographic (e.g., [BFLS91] and [GKR15]) but *none achieve constant query complexity*, and so applying our methodology (Theorem 1) to these would lead to large argument sizes (many tens of kilobytes). These prior algebraic protocols rely on the multivariate sumcheck protocol applied to certain multivariate polynomials, which means that they incur sizable communication costs due to (a) the many rounds of the sumcheck protocol, and (b) the fact that applying the methodology would involve using multivariate polynomial commitment schemes that (for known constructions) lead to communication costs that are linear in the number of variables.

In contrast, our algebraic protocol relies on univariate polynomials and achieves constant query complexity, incurring small communication costs. Our algebraic protocol can be viewed as a "holographic variant" of the algebraic protocol for R1CS used in Aurora [BCRSVW19], because it achieves an *exponential* improvement in verification time when the verifier is given a suitable encoding of the coefficient matrices; see Table 2.1.

construction	holographic?	indexer	prover	verifier	messages	proof length	queries
[BCRSVW19]	NO	N/A	$O(m + n \log n)$	$O(\lvert\mathbb{x}\rvert + n)$	3	$O(n)$	$O(1)$
this work	YES	$O(m \log m)$	$O(m \log m)$	$O(\lvert\mathbb{x}\rvert + \log m)$	7	$O(m)$	$O(1)$

Table 2.1: Comparison of the *non*-holographic protocol for R1CS in [BCRSVW19], and the AHP for R1CS that we construct. Here n denotes the number of variables and m the number of non-zero coefficients in the matrices.

(3) Extractable polynomial commitments. Polynomial commitment schemes, introduced in [KZG10], are commitment schemes specialized to work with univariate polynomials. The security properties in [KZG10], while sufficient for the applications therein, do not appear sufficient for standalone use, or even just for the transformation in Theorem 1. We propose a definition for polynomial commitment schemes that incorporates the functionality and security that we believe to suffice for standalone use (and in particular suffices for Theorem 1). Moreover, we show how to extend the construction of [KZG10] to fulfill this definition in the plain model under non-falsifiable

knowledge assumptions, or via a more efficient construction in the algebraic group model [FKL18] under falsifiable assumptions. These constructions are of independent interest, and when combined with our transformation, lead to the first efficient preprocessing arguments with universal SRS under concrete knowledge assumptions, and also to the efficiency reported in Fig. 2.1.

We have implemented in a Rust library[5] the polynomial commitment schemes, and our implementation of MARLIN relies on this library. We deem this library of independent interest for other projects.

2.1.2 Related work

In this work we study the goal of constructing preprocessing SNARGs with universal SRS, which achieve succinct verification regardless of the structure of the non-deterministic computation being checked. The most relevant prior work is Sonic [MBKM19], on which we improve as already discussed (see Fig. 2.1). The notion of updatable SRS was defined and achieved in [GKMMM18], but with a less efficient construction.

Concurrent work. A concurrent work [GWC19] studies the same question as this work, and also obtains efficiency improvements over Sonic [MBKM19]. Below is a brief comparison.

- Similarly to our work, [GWC19] extends the polynomial commitment in [KZG10] to support batching, and proves the extension secure in the algebraic group model. We additionally show how to prove security in the plain model under non-falsifiable knowledge assumptions, and consider the problem of enforcing different degrees for different polynomials (a feature that is not needed in [GWC19]).

- We show how to compile any algebraic holographic proof into a preprocessing argument with universal SRS, while [GWC19] focus on compiling a more restricted notion that they call "polynomial protocols".

- Our protocol natively supports R1CS, and can be viewed as a holographic variant of the algebraic protocol in [BCRSVW19]. The protocol in [GWC19] natively supports a different constraint system, and involves a protocol that, similar to [Gro10], uses a permutation argument to attest that all variables in the same cycle of a permutation are equal (e.g., $(1)(2,3)(4)$ would require that the second and third entries are equal).

Preprocessing SNARGs with a URS. Concurrent with this work, Chiesa, Ojha, and Spooner [COS20] proposed FRACTAL, a preprocessing SNARK for R1CS that relies only on a URS. Compared to this work, FRACTAL achieves larger proof sizes, asymptotically similar but concretely worse prover time, and similar verifier time, but relies only on simple symmetric cryptography.

Subsequent to this work, Setty [Set20] also proposed a preprocessing SNARG for R1CS that uses only a URS (uniform reference string). For an R1CS instance containing m non-zero entries, Setty's protocol achieves proving time $O_\lambda(m)$, argument size $O_\lambda(m^{1/c})$, and verification time

[5] `https://github.com/scipr-lab/poly-commit`

$O_\lambda(m^{1-1/c})$, for a chosen constant c. The protocol in [Set20] offers a tradeoff compared to our work: preprocessing with a URS instead of an SRS, at the cost of asymptotically larger argument size and verification time. The question of achieving processing with a URS while also achieving asymptotically small argument size and verification time remains open.

Non-preprocessing SNARGs for arbitrary computations. Checking arbitrary circuits without preprocessing them requires the verifier to read the circuit, so the main goal is to obtain small argument size. In this setting of non-preprocessing SNARGs for arbitrary circuits, constructions with a URS (uniform reference string) are based on discrete logarithms [BCCGP16; BBBPWM18] or hash functions [AHIV17; BCRSVW19], while constructions with a universal SRS (structured reference string) combine polynomial commitments and non-holographic algebraic proofs [Gab19]; all use random oracles to obtain non-interactive arguments.[6]

We find it interesting to remark that our methodology from Theorem 1 generalizes protocols such as [Gab19] in two ways. First, it formalizes the folklore approach of combining polynomial commitments and algebraic proofs to obtain arguments, identifying the security properties required to make this approach work. Second, it demonstrates how for algebraic *holographic* proofs the resulting argument enables preprocessing.

Non-preprocessing SNARGs for structured computations. Several works study SNARGs for structured computations. This structure enables fast verification *without* preprocessing. A line of works [Ben+17; BBHR19; BCGGRS19] combines hash functions and various interactive oracle proofs. Another line of works [ZGKPP17b; ZGKPP18; ZGKPP17a; WTSTW18; XZZPS19] combines multivariate polynomial commitments [PST13] and doubly-efficient interactive proofs [GKR15].

While in this work we study a different setting (*preprocessing* SNARGs for *arbitrary* computations), there are similarities, and notable differences, in the polynomial commitments used in our work and prior works. We begin by noting that the notion of "multivariate polynomial commitments" varies considerably across prior works, despite the fact that most of those commitments are based on the protocol introduced in [PST13].

- The commitments used in [ZGKPP17b; ZGKPP18] are required to satisfy extractability (a stronger notion than binding) because the security proof of the argument system involves extracting a polynomial encoding a witness. The commitment is a modification of [PST13] that uses knowledge commitments, a standard ingredient to achieve extractability under non-falsifiable assumptions in the plain model. Neither of these works consider hiding commitments as zero knowledge is not a goal for them.

- The commitments used in [ZGKPP17a; WTSTW18] must be compatible with the Cramer–Damgård transform [CD98] used in constructing the argument system. They consider a *modified setting* where the sender does not reveal the value of the commitment polynomial at a desired point but, instead, reveals a commitment to this value, along with a proof attesting that the

[6] The linear verification time in most of the cited constructions can typically be partially mitigated via techniques that enable an untrusted party to help the verifier to check a batch of proofs for the same circuit faster than checking each proof individually (the linear cost in the circuit is paid only once per batch rather than once for each proof in the batch).

committed value is correct. For this modified setting, they consider commitments that satisfy natural notions of extractability *and hiding* (achieving zero knowledge arguments is a goal in both papers). The commitments constructed in the two papers offer different tradeoffs. The commitment in [ZGKPP17a] is based on [PST13]: it relies on an SRS (structured reference string); it uses pairings; and for ℓ-variate polynomials achieves $O_\lambda(\ell)$-size arguments that can be checked in $O_\lambda(\ell)$ time. The commitment in [WTSTW18] is inspired from [BG12] and [BBBPWM18]: it relies on a URS (uniform reference string); it does not use pairings; and for ℓ-variate *multilinear* polynomials and a given constant $c \geq 2$ achieves $O_\lambda(2^{\ell/c})$-size arguments that can be checked in $O_\lambda(2^{\ell-\ell/c})$ time.

- The commitments used in [XZZPS19] are intended for the regular (unmodified) setting of commitment schemes where the sender reveals the value of the polynomial, because zero knowledge is later achieved by building on the algebraic techniques described in [CFS17]. The commitment definition in [XZZPS19] considers binding and hiding, but not extractability. However, the given security analysis for the argument system does not seem to go through for this definition (there is no explanation of where the witness encoded in the committed polynomial comes from). Also, no commitment construction is provided in [XZZPS19], and instead the reader is referred to [ZGKPP17a], which considers the modified setting described above.

In sum there are multiple notions of commitment and one must be precise about the functionality and security needed to construct an argument system. We now compare prior notions of commitments to the one that we use.

First, since in this work we do not use the Cramer–Damgård transform for zero knowledge, commitments in the modified setting are not relevant. Instead, we achieve zero knowledge via *bounded independence* [BCGV16], and in particular we consider the familiar setting where the sender reveals evaluations to the committed polynomial. Second, prior works consider protocols where the sender commits to a polynomial in a single round, while we consider protocols where the sender commits to multiple polynomials of different degrees in each of several rounds. This multi-polynomial multi-round setting requires suitable extensions in terms of functionality (to enable batching techniques to save on argument size) and security (extractability and hiding need to be strengthened), which means that prior definitions do not suffice for us.

The above discrepancies have led us to formulate new definitions of functionality and security for polynomial commitments (as summarized in Section 2.2.2). We conclude by noting that, since in this work we construct arguments that use *univariate* polynomials, our definitions are specialized to commitments for univariate polynomials. Corresponding definitions for multivariate polynomials can be obtained with straightforward modifications, and would strengthen definitions appearing in some prior works. Similarly, we fulfill the required definitions via natural adaptations of the univariate scheme of [KZG10], and analogous adaptations of the multivariate scheme of [PST13] would fulfill the multivariate analogues of these definitions.

2.2 Techniques

We discuss the main ideas behind our results. First we describe the two building blocks used in Theorem 1: AHPs and polynomial commitment schemes (described in Sections 2.2.1 and 2.2.2 respectively). We describe how to combine these to obtain preprocessing arguments with universal SRS in Section 2.2.3. Next, we discuss constructions for these building blocks: in Section 2.2.4 we describe our AHP (underlying Theorem 2), and in Section 2.2.5 we describe our construction of polynomial commitments.

Throughout, instead of considering the usual notion of relations that consist of instance-witness pairs, we consider *indexed relations*, which consist of triples $(\mathtt{i}, \mathtt{x}, \mathtt{w})$ where $\mathtt{i}$ is the index, $\mathtt{x}$ is the instance, and $\mathtt{w}$ is the witness. This is because $\mathtt{i}$ represents the part of the verifier input that is preprocessed in the offline phase (e.g., the circuit description) and $\mathtt{x}$ represents the part of the verifier input that comes in the online phase (e.g., a partial assignment to the circuit's input wires). The *indexed language* corresponding to an indexed relation $\mathcal{R}$, denoted $\mathcal{L}(\mathcal{R})$, is the set of pairs $(\mathtt{i}, \mathtt{x})$ for which there exists a witness $\mathtt{w}$ such that $(\mathtt{i}, \mathtt{x}, \mathtt{w}) \in \mathcal{R}$.

2.2.1 Building block: algebraic holographic proofs

Interactive oracle proofs (IOPs) [BCS16; RRR16] are multi-round protocols where in each round the verifier sends a challenge and the prover sends an oracle (which the verifier can query). IOPs combine features of interactive proofs [Bab85; GMR89]and probabilistically checkable proofs [BFLS91; AS98; ALMSS98]. *Algebraic holographic proofs* (AHPs) modify the notion of an IOP in two ways.

- *Holographic:* the verifier does not receive its input explicitly but, rather, has oracle access to a prescribed *encoding* of it. This potentially enables the verifier to run in time that is much faster than the time to read its input in full. (Our constructions will achieve this fast verification.)
- *Algebraic:* the honest prover must produce oracles that are low-degree polynomials (this restricts the completeness property), and all malicious provers must produce oracles that are low-degree polynomials (this relaxes the soundness property). The encoded input to the verifier must also be a low-degree polynomial.

Since in this work we only work with *univariate* polynomials, our definitions focus on this case, but they can be modified in a straightforward way to be more general.

Informally, a (public-coin) AHP over a field $\mathbb{F}$ for an indexed relation $\mathcal{R}$ is specified by an indexer $\mathbf{I}$, prover $\mathbf{P}$, and verifier $\mathbf{V}$ that work as follows.

- *Offline phase.* The indexer $\mathbf{I}$ receives as input the index $\mathtt{i}$ to be preprocessed, and outputs one or more univariate polynomials over $\mathbb{F}$ encoding $\mathtt{i}$.
- *Online phase.* For some instance $\mathtt{x}$ and witness $\mathtt{w}$, the prover $\mathbf{P}$ receives $(\mathtt{i}, \mathtt{x}, \mathtt{w})$ and the verifier $\mathbf{V}$ receives $\mathtt{x}$; $\mathbf{P}$ and $\mathbf{V}$ interact over some (in this work, constant) number of rounds, where in each round $\mathbf{V}$ sends a challenge and $\mathbf{P}$ sends one or more polynomials; after the interaction, $\mathbf{V}(\mathtt{x})$

probabilistically queries the polynomials output by the indexer and the polynomials output by the prover, and then accepts or rejects. Crucially, $\mathbf{V}$ does *not* receive $\mathtt{i}$ as input, but instead queries the polynomials output by $\mathbf{I}$ that encode $\mathtt{i}$. This enables the construction of verifiers $\mathbf{V}$ that run in time that is sublinear in $|\mathtt{i}|$.

The completeness property states that for every $(\mathtt{i}, \mathtt{x}, \mathtt{w}) \in \mathcal{R}$ the probability that $\mathbf{P}(\mathtt{i}, \mathtt{x}, \mathtt{w})$ convinces $\mathbf{V}^{\mathbf{I}(\mathtt{i})}(\mathtt{x})$ to accept is 1. The soundness property states that for every $(\mathtt{i}, \mathtt{x}) \notin \mathcal{L}(\mathcal{R})$ and *admissible* prover $\tilde{\mathbf{P}}$ the probability that $\tilde{\mathbf{P}}$ convinces $\mathbf{V}^{\mathbf{I}(\mathtt{i})}(\mathtt{x})$ to accept is at most a given soundness error ϵ. A prover is "admissible" if the degrees of the polynomials it outputs fit within prescribed degree bounds of the protocol. See Section 2.4 for details on AHPs, including definitions of proof of knowledge and zero knowledge.

Remark 2.2.1 (prior holographic proofs)**.** Various definitions of "holographic proofs" have been studied in the literature on probabilistic proofs, starting with the seminal work of Babai, Fortnow, Levin, and Szegedy [BFLS91]. Recent examples include the IPs in [GKR15], whose verifier runs in sublinear time when given (multivariate low-degree) encodings of the circuit's wiring predicates and of the circuit's input; and also the IOPs in [RRR16], where encoded provers and encoded inputs play a role in amortizing interactive proofs.

2.2.2 Building block: polynomial commitments

Informally, a *polynomial commitment scheme* [KZG10] allows a prover to produce a commitment c to a univariate polynomial $p \in \mathbb{F}[X]$, and later "open" $p(X)$ at any point $z \in \mathbb{F}$, producing an *evaluation proof* π showing that the opened value is consistent with the polynomial "inside" c at z. Turning this informal goal into a useful definition requires some care, however, as we explain below. In this work we propose a set of definitions for polynomial commitment schemes that we believe are useful for standalone use, and in particular suffice as a building block for our compiler described in Sections 2.2.3 and 2.8.

First, we consider constructions with strong efficiency requirements: the commitment c is much smaller than the polynomial p (e.g., c consists of a constant number of group elements), and the proof π can be validated very fast (e.g., in a constant number of cryptographic operations). These requirements not only rule out natural constructions, [7]but also imply that the usual binding property, which states that an efficient adversary cannot open the same commitment to two different values, does not capture the desired security. Indeed, even if the adversary were to be bound to opening values of some function $f : \mathbb{F} \to \mathbb{F}$, it may be that the function f is consistent with a polynomial

[7] A natural construction would be to use a standard commitment scheme to commit to each coefficient of p, and then open to a value by revealing the committed coefficients. However, this construction is inefficient, because the commitment c and evaluation proof π are "long" (linear in the degree of p). An alternative construction would be to use a Merkle tree on the coefficients of p. While c now becomes short, the evaluation proof π remains long because the receiver would need to see all coefficients to validate a claimed evaluation. Crucially, both constructions enable the receiver to check the degree of the committed polynomial.

whose degree is *higher* than what was claimed. This means that a security definition needs to incorporate guarantees about the degree of the committed function.[8]

Second, in many applications of polynomial commitments, an adversary produces multiple commitments to polynomials within a round of interaction and across rounds of interaction. After this interaction, the adversary reveals values of all of these polynomials at one or more locations. This setting motivates a number of considerations. First, it is desirable to rely on a single set of public parameters for committing to multiple polynomials, even if the polynomials differ in degree. A construction such as that of [KZG10] can be modified in a natural way to achieve this is by committing both to the polynomial and its shift to the maximum degree, similarly to techniques used to bundle multiple low-degree tests into a single one [BCRSVW19]. This modification needs to be addressed in any proof of security. Second, it would be desirable to batch evaluation proofs across different polynomials for the same location. Again the construction in [KZG10] can support this, but one must argue that security still holds in this more general case.

The preceding considerations require an extension of previous definitions and motivate our re-formulation of the primitive. Informally, a polynomial commitment scheme PC is a tuple of algorithms $\mathsf{PC} = (\mathsf{Setup}, \mathsf{Trim}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Check})$. The setup algorithm $\mathsf{PC.Setup}$ takes as input a security parameter and maximum supported degree bound D, and outputs public parameters pp that contain the description of a finite field $\mathbb{F}$. The "trimming" algorithm $\mathsf{PC.Trim}$ then deterministically specializes these parameters for a given set of degree bounds and outputs a committer key ck and a receiver key rk. The sender can then invoke $\mathsf{PC.Commit}$ with input ck and a list of polynomials $\boldsymbol{p}$ with respective degree bounds $\boldsymbol{d}$ to generate a set of commitments $\boldsymbol{c}$. Subsequently, the sender can use $\mathsf{PC.Open}$ to produce a proof π that convinces the receiver that the polynomials "inside" $\boldsymbol{c}$ respect the degree bounds $\boldsymbol{d}$ and, moreover, evaluate to the claimed set of values $\boldsymbol{v}$ at a given query set Q that specifies any number of evaluation points for each polynomial. The receiver can invoke $\mathsf{PC.Check}$ to check this proof.

The scheme PC is required to satisfy *extractability* and *efficiency* properties, and also, optionally, a *hiding* property. We outline these properties below (see Section 2.6.1 for the details).

Extractability. Consider an efficient sender adversary $\mathcal{A}$ that can produce a commitment c and degree bound $d \leq D$ such that, when asked for an evaluation at some point $z \in \mathbb{F}$, can produce a supposed evaluation v and proof π such that $\mathsf{PC.Check}$ accepts. Then PC is *extractable* if for every maximum degree bound D and every sender adversary $\mathcal{A}$ who can produce such commitments, there exists a corresponding efficient extractor $\mathcal{E}_{\mathcal{A}}$ that outputs a polynomial p of degree at most d that "explains" c so that $p(z) = v$. While for simplicity we have described the most basic case here, our definition considers adversaries and extractors who interact over multiple rounds, wherein the adversary may produce multiple commitments in each round and the extractor is required to output corresponding polynomials on a per-round basis (before seeing the query set, proof, or supposed evaluations).

[8] This consideration motivates the *strong correctness* property in [KZG10], which states that *if* the adversary knows a polynomial that leads to the claimed commitment c then this polynomial has bounded degree. This notion, while sufficient for the application in [KZG10], does not seem to suffice for standalone use because there is no a priori guarantee that an adversary that can open values to a commitment knows a polynomial inside the commitment. In some sense, a knowledge assumption is hidden in this hypothesis.

In this work we rely on extractability to prove the security of our compiler (see Section 2.2.3); we do not know if weaker security notions studied in prior works, such as evaluation binding, suffice. More generally, we believe that extractability is a useful property that may be required across a range of other applications.

Efficiency. We require two notions of efficiency for PC. First, the time required to commit to a polynomial p and then to create an evaluation proof must be proportional to the degree of p, and not to the maximum degree D. (This ensures that the argument prover runs in time proportional to the size of the index.)

On the receiver's side, the commitment size, proof size, and time to verify an opening must be independent of the claimed degrees for the polynomials. (This ensures that the argument produced by our compiler is succinct.)

Hiding. The hiding property of PC states that commitments and proofs of evaluation reveal no information about the committed polynomial beyond the publicly stated degree bound and the evaluation itself. Namely, PC is *hiding* if there exists an efficient simulator that outputs simulated commitments and simulated evaluation proofs that cannot be distinguished from their real counterparts by any malicious distinguisher that only knows the degree bound and the evaluation.

Analogously to the case of extractability, we actually consider a more general definition that considers commitments to multiple polynomials within and across multiple rounds; moreover, the definition considers the case where some polynomials are designated as not hidden (and thus given to the simulator) because in our application we sometimes prefer to commit to a polynomial in a non-hiding way (for efficiency reasons).

2.2.3 Compiler: from AHPs to preprocessing arguments with universal SRS

We describe the main ideas behind Theorem 1, which uses polynomial commitment schemes to compile any (public-coin) AHP into a corresponding (public-coin) preprocessing argument with universal SRS. In a subsequent step, the argument can be made non-interactive via the Fiat–Shamir transformation, and thereby obtain a preprocessing SNARG with universal SRS.

The basic intuition of the compiler follows the well-known framework of "commit to oracles and then open query answers" pioneered by Kilian [Kil92]. However, the commitment scheme used in our compiler leverages and enforces the algebraic structure of these oracles. While several works in the literature already take advantage of algebraic commitment schemes applied to algebraic oracles, our contribution is to observe that if we apply this framework to a holographic proof then we obtain a preprocessing argument.

Informally, first the argument indexer invokes the AHP indexer to generate polynomials, and then deterministically commits to these using the polynomial commitment scheme. Subsequently, the argument prover and argument verifier interact, each respectively simulating the AHP prover and AHP verifier. In each round, the argument prover sends succinct commitments to the polynomials output by the AHP prover in that round. After the interaction, the argument verifier declares its queries to the polynomials (of the prover and of the indexer). The argument prover replies with

the desired evaluations along with an evaluation proof attesting to their correctness relative to the commitments.

This approach, while intuitive, must be proven secure. In particular, in the proof of soundness, we need to show that if the argument prover convinces the argument verifier with a certain probability, then we can find an AHP prover that convinces the AHP verifier with similar probability. This step is non-trivial: the AHP prover outputs polynomials, while the argument prover merely outputs succinct commitments and a few evaluations, which is much less information. In order to deduce the former from the latter requires *extraction*. This motivates considering polynomial commitment schemes that are extractable, in the sense described in Section 2.2.2. We do not know whether weaker security properties, such as the evaluation binding property studied in some prior works, suffice for proving the compiler secure.

The compiler outlined above is compatible with the properties of argument of knowledge and zero knowledge. Specifically, we prove that if the AHP is a proof of knowledge, then the compiler produces an argument of knowledge; also, if the AHP is (bounded-query) zero knowledge and the polynomial commitment scheme is hiding, then the compiler produces a zero knowledge argument.

See Section 2.8 for more details on the compiler.

2.2.4 Construction: an AHP for constraint systems

In prior sections we have described how we can use polynomial commitment schemes to compile AHPs into corresponding preprocessing SNARGs. In this section we discuss the main ideas behind Theorem 2, which provides an efficient AHP for the indexed relation corresponding to R1CS (see Definition 1). The preprocessing zkSNARK that we achieve in this work (see Fig. 2.1) is based on this AHP.

Our protocol can be viewed as a "holographic variant" of the *non*-holographic algebraic proof for R1CS constructed in [BCRSVW19]. Achieving holography involves designing a new sub-protocol that enables the verifier to evaluate low-degree extensions of the coefficient matrices at a random location. While in [BCRSVW19] the verifier performed this computation in time $\mathrm{poly}(|\mathbb{i}|)$ on its own, in our protocol the verifier performs it *exponentially faster*, in time $O(\log |\mathbb{i}|)$, by receiving help from the prover and having oracle access to the polynomials produced by the indexer. We introduce notation and then discuss the protocol.

Some notation. Consider an index $\mathbb{i} = (\mathbb{F}, n, m, A, B, C)$ specifying coefficient matrices, an instance $\mathbb{x} = x \in \mathbb{F}^*$ specifying a partial assignment to the variables, and a witness $\mathbb{w} = w \in \mathbb{F}^*$ specifying an assignment to the other variables such that the R1CS equation holds. The R1CS equation holds if and only if $Az \circ Bz = Cz$ for $z := (x, w) \in \mathbb{F}^m$. Below, we let H and K be prescribed subsets of $\mathbb{F}$ of sizes n and m respectively; we also let $v_H(X)$ and $v_K(X)$ be the vanishing polynomials of these two sets. (The vanishing polynomial of a set S is the monic polynomial of degree $|S|$ that vanishes on S, i.e., $\prod_{\gamma \in S}(X - \gamma)$.) We assume that both H and K are smooth multiplicative subgroups. This allows interpolation/evaluation over H in $O(n \log n)$ operations and also makes $v_H(X)$ computable in $O(\log n)$ operations (and similarly for K). Given an $n \times n$ matrix M with rows/columns indexed by elements of H, we denote by $\hat{M}(X, Y)$ the low-degree extension

of M, i.e., the polynomial of individual degree less than n such that $\hat{M}(\kappa, \iota)$ is the (κ, ι)-th entry of M for every $\kappa, \iota \in H$.

A non-holographic starting point. We sketch a *non*-holographic protocol for R1CS with linear proof length and constant query complexity, inspired from [BCRSVW19], that forms the starting point of our work. In this case the prover receives as input $(\mathbb{i}, \mathbb{x}, \mathbb{w})$ and the verifier receives as input $(\mathbb{i}, \mathbb{x})$. (The verifier reads the non-encoded index $\mathbb{i}$ because we are describing a non-holographic protocol.)

In the first message the prover $\mathbf{P}$ sends the univariate polynomial $\hat{z}(X)$ of degree less than n that agrees with the variable assignment z on H, and also sends the univariate polynomials $\hat{z}_A(X), \hat{z}_B(X), \hat{z}_C(X)$ of degree less than n that agree with the linear combinations $z_A := Az$, $z_B := Bz$, and $z_C := Cz$ on H. The prover is left to convince the verifier that the following two conditions hold:

$$\text{(1) Entry-wise product:} \quad \forall\, \kappa \in H\ ,\ \hat{z}_A(\kappa)\hat{z}_B(\kappa) - \hat{z}_C(\kappa) = 0\ .$$

$$\text{(2) Linear relation:} \quad \forall\, M \in \{A, B, C\}\ ,\ \forall\, \kappa \in H\ ,\ \hat{z}_M(\kappa) = \sum_{\iota \in H} M[\kappa, \iota]\hat{z}(\iota)\ .$$

(The prover also needs to convince the verifier that $\hat{z}(X)$ encodes a full assignment z that is consistent with the partial assignment x, but we for simplicity we ignore this in this informal discussion.)

In order to convince the verifier of the first (entry-wise product) condition, the prover sends the polynomial $h_0(X)$ such that $\hat{z}_A(X)\hat{z}_B(X) - \hat{z}_C(X) = h_0(X)v_H(X)$. This polynomial equation is equivalent to the first condition (the left-hand side equals zero everywhere on H if and only if it is a multiple of H's vanishing polynomial). The verifier will check the equation at a random point $\beta \in \mathbb{F}$: it queries $\hat{z}_A(X), \hat{z}_B(X), \hat{z}_C(X), h_0(X)$ at β, evaluates $v_H(X)$ at β on its own, and checks that $\hat{z}_A(\beta)\hat{z}_B(\beta) - \hat{z}_C(\beta) = h_0(\beta)v_H(\beta)$. The soundness error is the maximum degree over the field size, which is at most $2n/|\mathbb{F}|$.

In order to convince the verifier of the second (linear relation) condition, the prover expects a random challenge $\alpha \in \mathbb{F}$ from the verifier, and then replies in a second message. For each $M \in \{A, B, C\}$, the prover sends polynomials $h_M(X)$ and $g_M(X)$ such that

$$r(\alpha, X)\hat{z}_M(X) - r_M(\alpha, X)\hat{z}(X) = h_M(X)v_H(X) + Xg_M(X) \quad \text{for} \quad r_M(Z, X) := \sum_{\kappa \in H} r(Z, \kappa)\hat{M}(\kappa, X)$$

where $r(Z, X)$ is a prescribed polynomial of individual degree less than n such that $(r(Z, \kappa))_{\kappa \in H}$ are n linearly independent polynomials. Prior work [BCRSVW19] on checking linear relations via univariate sumchecks shows that this polynomial equation is equivalent, up to a soundness error of $n/|\mathbb{F}|$ over α, to the second condition.[9] The verifier will check this polynomial equation at the random point $\beta \in \mathbb{F}$: it queries $\hat{z}(X), \hat{z}_A(X), \hat{z}_B(X), \hat{z}_C(X), h_M(X), g_M(X)$ at β, evaluates $v_H(X)$ at β on its own, evaluates $r(Z, X)$ and $r_M(Z, X)$ at (α, β) on its own, and checks that

[9] In particular, we are using the fact from [BCRSVW19] that, given a multiplicative subgroup S of $\mathbb{F}$, a polynomial $f(X)$ sums to σ over S if and only if $f(X)$ can be written as $h(X)v_S(X) + Xg(X) + \sigma/|S|$ for some $h(X)$ and $g(X)$ with $\deg(g) < |S| - 1$.

$r(\alpha,\beta)\hat{z}_M(\beta) - r_M(\alpha,\beta)\hat{z}(\beta) = h_M(\beta)v_H(\beta) + \beta g_M(\beta)$. The additional soundness error is $2n/|\mathbb{F}|$.

The above is a simple 3-message protocol for R1CS with soundness error $\max\{2n/|\mathbb{F}|, 3n/|\mathbb{F}|\} = 3n/|\mathbb{F}|$ in the setting where the honest prover and malicious provers send polynomials of prescribed degrees, which the verifier can query at any location. The proof length (sum of all degrees) is linear in n and the query complexity is constant.

Barrier to holography. The verifier in the above protocol runs in time that is $\Omega(|\mathbb{i}|) = \Omega(n+m)$. While this is inherent in the non-holographic setting (because the verifier must read $\mathbb{i}$), we now discuss how exactly the verifier's computation depends on $\mathbb{i}$. We shall later use this understanding to achieve an exponential improvement in the verifier's time when given a suitable encoding of $\mathbb{i}$.

The verifier's check for the entry-wise product is $\hat{z}_A(\beta)\hat{z}_B(\beta) - \hat{z}_C(\beta) = h_0(\beta)v_H(\beta)$, and can be carried out in $O(\log n)$ operations *regardless* of the coefficient matrices contained in the index $\mathbb{i}$. In other words, this check is efficient even in the non-holographic setting. However, the verifier's check for the linear relation is $r(\alpha,\beta)\hat{z}_M(\beta) - r_M(\alpha,\beta)\hat{z}(\beta) = h_M(\beta)v_H(\beta) + \beta g_M(\beta)$, which has a linear cost. Concretely, evaluating the polynomial $r_M(Z,X)$ at (α,β) requires $\Omega(n+m)$ operations.

In the holographic setting, a natural idea to reduce this cost would be to grant the verifier oracle access to the low-degree extension $\hat{M}$ for $M \in \{A,B,C\}$. This idea has two problems: the verifier *still* needs $\Omega(n)$ operations to evaluate $r_M(Z,X)$ at (α,β) and, moreover, the size of $\hat{M}$ is *quadratic* in n, which means that the encoding of the index $\mathbb{i}$ is $\Omega(n^2)$. We cannot afford such an expensive encoding in the offline preprocessing phase. We now describe how we overcome both of these problems, and obtain a holographic protocol.

Achieving holography. To overcome the above problems and obtain a holographic protocol, we rely yet again on the univariate sumcheck protocol. We introduce two additional rounds of interaction, and in each round the verifier learns that their verification equation holds provided the sumcheck from the next round holds. The last sumcheck will rely on polynomials output by the indexer, which the verifier knows are correct.

We address the first problem by letting the prover and verifier interact in an additional round, where we rely on an additional univariate sumcheck to reduce the problem of evaluating $r_M(Z,X)$ at (α,β) to the problem of evaluating $\hat{M}$ at (β_2,β) for a random $\beta_2 \in \mathbb{F}$. Namely, the verifier sends β to the prover, who computes

$$\sigma_2 := r_M(\alpha,\beta) = \sum_{\kappa \in H} r(\alpha,\kappa)\hat{M}(\kappa,\beta).$$

Then the prover replies with σ_2 and the polynomials $h_2(X)$ and $g_2(X)$ such that

$$r(\alpha,X)\hat{M}(X,\beta) = h_2(X)v_H(X) + Xg_2(X) + \sigma_2/n \ .$$

Prior techniques on univariate sumcheck [BCRSVW19] tell us that this equation is equivalent to the polynomial $r(\alpha,X)\hat{M}(X,\beta)$ summing to σ_2 on H. Thus the verifier needs to check this equation at a random $\beta_2 \in \mathbb{F}$: $r(\alpha,\beta_2)\hat{M}(\beta_2,\beta) = h_2(\beta_2)v_H(\beta_2) + \beta_2 g_2(\beta_2) + \sigma_2/n$. The only expensive part

of this equation for the verifier is computing the value $\hat{M}(\beta_2, \beta)$, which is problematic. Indeed, we have already noted that we cannot afford to simply let the verifier have oracle access to $\hat{M}$, because this polynomial has quadratic size (it contains a quadratic number of terms).

We address this second problem as follows. Let $u_H(X, Y) := \frac{v_H(X) - v_H(Y)}{X - Y}$ be the formal derivative of the vanishing polynomial $v_H(X)$, and note that $u_H(X, Y)$ vanishes on the square $H \times H$ except for on the diagonal, where it takes on the (non-zero) values $(u_H(a, a))_{a \in H}$. Moreover, $u_H(X, Y)$ can be evaluated at any point in $\mathbb{F} \times \mathbb{F}$ in $O(\log n)$ operations. Using this polynomial, we can write $\hat{M}$ as a sum of $m = |K|$ terms instead of $n^2 = |H|^2$ terms:

$$\hat{M}(X, Y) := \sum_{\kappa \in K} u_H(X, \hat{\mathsf{row}}_M(\kappa)) \cdot u_H(Y, \hat{\mathsf{col}}_M(\kappa)) \cdot \hat{\mathsf{val}}_M(\kappa) \ ,$$

where $\hat{\mathsf{row}}_M, \hat{\mathsf{col}}_M, \hat{\mathsf{val}}_M$ are the low-degree extensions of the row, column, and value of the non-zero entries in M according to some canonical order over K.[10]

This method of representing the low-degree extension of M suggests an idea: let the verifier have oracle access to the polynomials $\hat{\mathsf{row}}_M, \hat{\mathsf{col}}_M, \hat{\mathsf{val}}_M$ and do *yet another* univariate sumcheck, but this time over the set K. The verifier sends β_2 to the prover, who computes

$$\sigma_3 := \hat{M}(\beta_2, \beta) = \sum_{\kappa \in K} u_H(\beta_2, \hat{\mathsf{row}}_M(\kappa)) \cdot u_H(\beta, \hat{\mathsf{col}}_M(\kappa)) \cdot \hat{\mathsf{val}}_M(\kappa) \ .$$

Then the prover replies with σ_3 and the polynomials $h_3(X)$ and $g_3(X)$ such that

$$u_H(\beta_2, \hat{\mathsf{row}}_M(X)) u_H(\beta, \hat{\mathsf{col}}_M(X)) \hat{\mathsf{val}}_M(X) = h_3(X) v_K(X) + X g_3(X) + \sigma_3 / m \ .$$

The verifier can then check this equation at a random $\beta_3 \in \mathbb{F}$, which only requires $O(\log m)$ operations.

The above idea *almost* works; the one remaining problem is that $h_3(X)$ has degree $\Omega(nm)$ (because the left-hand size of the equation has quadratic degree), which is too expensive for our target of a quasilinear-time prover. We overcome this problem by letting the prover run the univariate sumcheck protocol on the unique low-degree extension $\hat{f}(X)$ of the function $f \colon K \to \mathbb{F}$ defined as $f(\kappa) := u_H(\beta_2, \hat{\mathsf{row}}_M(\kappa)) u_H(\beta, \hat{\mathsf{col}}_M(\kappa)) \hat{\mathsf{val}}_M(\kappa)$. Observe that $\hat{f}(X)$ has degree less than m. The verifier checks that $\hat{f}(X)$ and $u_H(\beta_2, \hat{\mathsf{row}}_M(X)) u_H(\beta, \hat{\mathsf{col}}_M(X)) \hat{\mathsf{val}}_M(X)$ agree on K.

From sketch to protocol. In the above discussion we have ignored a number of technical aspects, such as proof of knowledge and zero knowledge (which are ultimately needed in the compiler if we want to construct a preprocessing zkSNARK). We have also not discussed time complexities of many algebraic steps, and we omitted discussion of how to batch multiple sumchecks into fewer ones, which brings important savings in argument size. For details, see our detailed construction in Section 2.5.

[10] Technicality: $\hat{\mathsf{val}}(\kappa)$ actually equals the value divided by $u_H(\hat{\mathsf{row}}_M(\kappa), \hat{\mathsf{row}}_M(\kappa)) u_H(\hat{\mathsf{col}}_M(\kappa), \hat{\mathsf{col}}_M(\kappa))$.

2.2.5 Construction: extractable polynomial commitments

We now sketch how to construct a polynomial commitment scheme that achieves the strong functionality and security requirements of our definition in Section 2.2.2. Our starting point is the $\mathsf{PolyCommit}_{\mathrm{DL}}$ construction of Kate et al. [KZG10], and then describe a sequence of natural and generic transformations that extend this construction to enable extractability, commitments to multiple polynomials, and the enforcement of per-polynomial degree bounds. In fact, once we arrive at a scheme that supports extractability for committed polynomials at a single point (Section 2.11), our transformations build on this construction in a black box way to first support per-polynomial degree bounds (Section 2.12), and then query sets that may request multiple evaluation points per polynomial (Section 2.13). Indeed, it is sufficient to produce a polynomial commitment scheme that satisfies the much more simple interface and definitions in Section 2.11.1, and apply these black box transformations to obtain a polynomial commitment scheme that satisfies the interface of and provides the properties described in Section 2.6.1 ultimately needed by our compiler.

Starting point: $\mathsf{PolyCommit}_{\mathbf{DL}}$. The setup phase samples a cryptographically secure bilinear group $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, G, H, e)$ and then samples a committer key ck and receiver key rk for a given degree bound D. The committer key consists of group elements encoding powers of a random field element β, namely, $\mathsf{ck} := \{G, \beta G, \ldots, \beta^D G\} \in \mathbb{G}_1^{D+1}$. The receiver key consists of the group elements $\mathsf{rk} := (G, H, \beta H) \in \mathbb{G}_1 \times \mathbb{G}_2^2$. Note that the SRS, which consists of the keys ck and rk, is updatable because the coefficients of group elements in the SRS are all monomials (see Remark 2.7.1).

To commit to a polynomial $p \in \mathbb{F}_q[X]$, the sender computes $c := p(\beta)G$. To subsequently prove that the committed polynomial evaluates to v at a point z, the sender computes a witness polynomial $w(X) := (p(X) - p(z))/(X - z)$, and provides as proof a commitment to w: $\pi := w(\beta)G$. The idea is that the witness function w is a polynomial *if and only if* $p(z) = v$; otherwise, it is a rational function, and cannot be committed to using ck.

Finally, to verify a proof of evaluation, the receiver checks that the commitment and proof of evaluation are consistent. That is, it checks that the proof commits to a polynomial of the form $(p(X) - p(z))/(X - z)$ by checking the equality $e(c - vG, H) = e(\pi, \beta H - zH)$.

Achieving extractability. While the foregoing construction guarantees correctness of evaluations, it does not by itself guarantee that a commitment actually "contains" a suitable polynomial of degree at most D. We study two methods to address this issue, and thereby achieve extractability. One method is to modify the construction to use knowledge commitments [Gro10], and rely on a concrete knowledge assumption. The main disadvantage of this approach is that each commitment doubles in size. The other method is to move away from the plain model, and instead conduct the security analysis in the algebraic group model (AGM) [FKL18]. This latter method is more efficient because each commitment remains a single group element.

Committing to multiple polynomials at once. We enable the sender to simultaneously open multiple polynomials $[p_i]_{i=1}^n$ at the same point z as follows. Before generating a proof of evaluation for $[p_i]_{i=1}^n$, the sender requests from the receiver a random field element ξ, which he uses to take a random linear combination of the polynomials: $p := \sum_{i=1}^n \xi^i p_i$, and generates a proof of evaluation π for this polynomial p.

The receiver verifies π by using the fact that the commitments are additively homomorphic. The receiver takes a linear combination of the commitments and claimed evaluations, obtaining the combined commitment $c = \sum_{i=1}^{n} \xi^i c_i$ and evaluation $v = \sum_{i=1}^{n} \xi^i v_i$. Finally, it checks the pairing equations for c, π, and v.

Completeness of this check is straightforward, while soundness follows from the fact that if any polynomial does not match its evaluation, then the combined polynomial will not match its evaluation with high probability.

Enforcing multiple degree bounds. The construction so far enforces a single bound D on the degrees of all the polynomials p_i. To enforce a different degree bound d_i for each p_i, we require the sender to commit not only to each p_i, but also to "shifted polynomials" $p'_i(X) := X^{D-d_i} p_i(X)$. The proof of evaluation proves that, if p_i evaluates to v_i at z, then p'_i evaluates to $z^{D-d_i} v_i$.

The receiver checks that the commitment for each p'_i corresponds to an evaluation $z^{D-d_i} v_i$ so that, if z is sampled from a super-polynomial subset of $\mathbb{F}_q$, the probability that $\deg(p_i) \neq d_i$ is negligible. This trick is similar to the one used in [BS08; BCRSVW19] to derive low-degree tests for specific degree bounds.

However, while sound, this approach is inefficient in our setting: the witness polynomial for p'_i has $\Omega(D)$ non-zero coefficients (instead of $O(d_i)$), and so constructing an evaluation proof for it requires $\Omega(D)$ scalar multiplications (instead of $O(d_i)$). To work around this, we instead produce a proof that the related polynomial $p^\star_i(X) := p'_i(X) - p_i(z)X^{D-d_i}$ evaluates to 0 at z. As we show in Lemma 2.12.2, the witness polynomial for this claim has $O(d_i)$ non-zero coefficients, and so constructing the evaluation proof can be done in $O(d_i)$ scalar multiplications. Completeness is preserved because the receiver can check the correct evaluation of $p^\star_i$ by subtracting $p_i(z)(\beta^{D-d_i}\mathbb{G})$ from the commitment to the shifted polynomial p'_i, thereby obtaining a commitment to $p^\star_i$, while security is preserved because $p'_i(z) = z^{D-d_i} v_i \iff p^\star_i(z) = 0$.

Evaluating at a query set instead of a single point. To support the case where the polynomials $[p_i]_{i=1}^n$ are evaluated at a set of points Q, the sender proceeds as follows. Say that there are k different points $[z_i]_{i=1}^k$ in Q. The sender partitions the polynomials $[p_i]_{i=1}^n$ into different groups such that every polynomial in a group is to be evaluated at the same point z_i. The sender runs $\mathsf{PC.Open}$ on each group, and outputs the resulting list of evaluation proofs.

Achieving hiding. To additionally achieve hiding, we follow the above blueprint, replacing $\mathsf{PolyCommit}_{\text{DL}}$ with the hiding scheme $\mathsf{PolyCommit}_{\text{Ped}}$ described in [KZG10].

2.3 Preliminaries

We denote by $[n]$ the set $\{1, \ldots, n\} \subseteq \mathbb{N}$. We use $\boldsymbol{a} = [a_i]_{i=1}^n$ as a short-hand for the tuple $(a_1, \ldots, a_n)$, and $[\boldsymbol{a}_i]_{i=1}^n = [[a_{i,j}]_{j=1}^m]_{i=1}^n$ as a short-hand for the tuple $(a_{1,1}, \ldots, a_{1,m}, \ldots, a_{n,1}, \ldots, a_{n,m})$; $|\boldsymbol{a}|$ denotes the number of entries in $\boldsymbol{a}$. If x is a binary string then $|x|$ denotes its bit length. If M is a matrix then $\|M\|$ denotes the number of nonzero entries in M. If S is a finite set then $|S|$ denotes its cardinality and $x \leftarrow S$ denotes that x is an element sampled at random from S. We denote by $\mathbb{F}$ a finite field, and whenever $\mathbb{F}$ is an input to an algorithm we implicitly assume that $\mathbb{F}$ is represented in a way that allows efficient field arithmetic. Given a finite set S, we denote by $\mathbb{F}^S$ the set of vectors indexed by elements in S. We denote by $\mathbb{F}[X]$ the ring of univariate polynomials over $\mathbb{F}$ in X, and by $\mathbb{F}^{<d}[X]$ the set of polynomials in $\mathbb{F}[X]$ with degree less than d.

We denote by $\lambda \in \mathbb{N}$ a security parameter. When we state that $n \in \mathbb{N}$ for some variable n, we implicitly assume that $n = \mathrm{poly}(\lambda)$. We denote by $\mathrm{negl}(\lambda)$ an unspecified function that is *negligible* in λ (namely, a function that vanishes faster than the inverse of any polynomial in λ). When a function can be expressed in the form $1 - \mathrm{negl}(\lambda)$, we say that it is *overwhelming* in λ. When we say that $\mathcal{A}$ is an *efficient adversary* we mean that $\mathcal{A}$ is a family $\{\mathcal{A}_\lambda\}_{\lambda \in \mathbb{N}}$ of non-uniform polynomial-size circuits. If the adversary consists of multiple circuit families $\mathcal{A}_1, \mathcal{A}_2, \ldots$ then we write $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \ldots)$.

Given two interactive algorithms A and B, we denote by $\langle A(x), B(y)\rangle(z)$ the output of $B(y, z)$ when interacting with $A(x, z)$. Note that this output could be a random variable. If we use this notation when A or B is a circuit, we mean that we are considering a circuit that implements a suitable next-message function to interact with the other party of the interaction.

2.3.1 Indexed relations

An *indexed relation* $\mathcal{R}$ is a set of triples $(\mathtt{i}, \mathtt{x}, \mathtt{w})$ where $\mathtt{i}$ is the index, $\mathtt{x}$ is the instance, and $\mathtt{w}$ is the witness; the corresponding *indexed language* $\mathcal{L}(\mathcal{R})$ is the set of pairs $(\mathtt{i}, \mathtt{x})$ for which there exists a witness $\mathtt{w}$ such that $(\mathtt{i}, \mathtt{x}, \mathtt{w}) \in \mathcal{R}$. For example, the indexed relation of satisfiable boolean circuits consists of triples where $\mathtt{i}$ is the description of a boolean circuit, $\mathtt{x}$ is a partial assignment to its input wires, and $\mathtt{w}$ is an assignment to the remaining wires that makes the circuit to output 0. Given a size bound $\mathsf{N} \in \mathbb{N}$, we denote by $\mathcal{R}_{\mathsf{N}}$ the restriction of $\mathcal{R}$ to triples $(\mathtt{i}, \mathtt{x}, \mathtt{w})$ with $|\mathtt{i}| \leq \mathsf{N}$.

2.4 Algebraic holographic proofs

We define *algebraic holographic proofs* (AHPs), the notion of proofs that we use. For simplicity, the formal definition below is tailored to univariate polynomials, because our AHP construction is in this setting. The definition can be modified in a straightforward way to consider the general case of multivariate polynomials.

We represent polynomials through the coefficients that define them, as opposed to through their evaluation over a sufficiently large domain (as is typically the case in probabilistic proofs). This definitional choice is due to the fact that we will consider verifiers that may query the polynomials at any location in the field of definition. Moreover, the field of definition itself can be chosen from a given field family, and so we make the field an additional input to all algorithms; this degree of freedom is necessary when combining this component with polynomial commitment schemes (see Section 2.8). Finally, we consider the setting of *indexed relations* (see Section 2.3.1), where the verifier's input has two parts, the index and the instance; in the definition below, the verifier receives the index encoded and the instance explicitly.

Formally, an **algebraic holographic proof** (AHP) over a field family $\mathcal{F}$ for an indexed relation $\mathcal{R}$ is specified by a tuple

$$\mathsf{AHP} = (\mathsf{k}, \mathsf{s}, \mathsf{d}, \mathbf{I}, \mathbf{P}, \mathbf{V})$$

where $\mathsf{k}, \mathsf{s}, \mathsf{d} \colon \{0,1\}^* \to \mathbb{N}$ are polynomial-time computable functions and $\mathbf{I}, \mathbf{P}, \mathbf{V}$ are three algorithms known as the *indexer*, *prover*, and *verifier*. The parameter k specifies the number of interaction rounds, s specifies the number of polynomials in each round, and d specifies degree bounds on these polynomials.

In the offline phase ("0-th round"), the indexer $\mathbf{I}$ receives as input a field $\mathbb{F} \in \mathcal{F}$ and an index $\mathbb{i}$ for $\mathcal{R}$, and outputs $\mathsf{s}(0)$ polynomials $p_{0,1}, \ldots, p_{0,\mathsf{s}(0)} \in \mathbb{F}[X]$ of degrees at most $\mathsf{d}(|\mathbb{i}|, 0, 1), \ldots, \mathsf{d}(|\mathbb{i}|, 0, \mathsf{s}(0))$ respectively. Note that the offline phase does not depend on any particular instance or witness, and merely considers the task of encoding the given index $\mathbb{i}$.

In the online phase, given an instance $\mathbb{x}$ and witness $\mathbb{w}$ such that $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \in \mathcal{R}$, the prover $\mathbf{P}$ receives $(\mathbb{F}, \mathbb{i}, \mathbb{x}, \mathbb{w})$ and the verifier $\mathbf{V}$ receives $(\mathbb{F}, \mathbb{x})$ and oracle access to the polynomials output by $\mathbf{I}(\mathbb{F}, \mathbb{i})$. The prover $\mathbf{P}$ and the verifier $\mathbf{V}$ interact over $\mathsf{k} = \mathsf{k}(|\mathbb{i}|)$ rounds.

For $i \in [\mathsf{k}]$, in the i-th round of interaction, the verifier $\mathbf{V}$ sends a message $\rho_i \in \mathbb{F}^*$ to the prover $\mathbf{P}$; then the prover $\mathbf{P}$ replies with $\mathsf{s}(i)$ oracle polynomials $p_{i,1}, \ldots, p_{i,\mathsf{s}(i)} \in \mathbb{F}[X]$. The verifier may query any of the polynomials it has received any number of times. A query consists of a location $z \in \mathbb{F}$ for an oracle $p_{i,j}$, and its corresponding answer is $p_{i,j}(z) \in \mathbb{F}$. After the interaction, the verifier accepts or rejects.

The function d determines which provers to consider for the completeness and soundness properties of the proof system. In more detail, we say that a (possibly malicious) prover $\tilde{\mathbf{P}}$ is **admissible** for AHP if, on every interaction with the verifier $\mathbf{V}$, it holds that for every round $i \in [\mathsf{k}]$ and oracle index $j \in [\mathsf{s}(i)]$ we have $\deg(p_{i,j}) \leq \mathsf{d}(|\mathbb{i}|, i, j)$. The honest prover $\mathbf{P}$ is required to be admissible under this definition.

We say that AHP has perfect completeness and soundness error ϵ if the following holds.

- **Completeness.** For every field $\mathbb{F} \in \mathcal{F}$ and index-instance-witness tuple $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \in \mathcal{R}$, the probability that $\mathbf{P}(\mathbb{F}, \mathbb{i}, \mathbb{x}, \mathbb{w})$ convinces $\mathbf{V}^{\mathbf{I}(\mathbb{F},\mathbb{i})}(\mathbb{F}, \mathbb{x})$ to accept in the interactive oracle protocol is 1.

- **Soundness.** For every field $\mathbb{F} \in \mathcal{F}$, index-instance pair $(\mathbb{i}, \mathbb{x}) \notin \mathcal{L}(\mathcal{R})$, and admissible prover $\tilde{\mathbf{P}}$, the probability that $\tilde{\mathbf{P}}$ convinces $\mathbf{V}^{\mathbf{I}(\mathbb{F},\mathbb{i})}(\mathbb{F}, \mathbb{x})$ to accept in the interactive oracle protocol is at most ϵ.

The *proof length* l is the sum of all degree bounds in the offline and online phases, $\mathsf{l}(|\mathbb{i}|) := \sum_{i=0}^{\mathsf{k}(|\mathbb{i}|)} \sum_{j=1}^{\mathsf{s}(i)} \mathsf{d}(|\mathbb{i}|, i, j)$. The intuition for this definition is that in a probabilistic proof each oracle would consist of the evaluation of a polynomial over a domain whose size (in field elements) is linearly related to its degree bound, so that the resulting proof length would be linearly related to the sum of all degree bounds.

The *query complexity* q is the total number of queries made by the verifier to the polynomials. This includes queries to the polynomials output by the indexer and those sent by the prover.

All AHPs that we construct achieve the stronger property of *knowledge soundness* (against admissible provers), and optionally also *zero knowledge*. We define both of these properties below.

Knowledge soundness. We say that AHP has knowledge error ϵ if there exists a probabilistic polynomial-time extractor $\mathbf{E}$ for which the following holds. For every field $\mathbb{F} \in \mathcal{F}$, index $\mathbb{i}$, instance $\mathbb{x}$, and admissible prover $\tilde{\mathbf{P}}$, the probability that $\mathbf{E}^{\tilde{\mathbf{P}}}(\mathbb{F}, \mathbb{i}, \mathbb{x}, 1^{\mathsf{l}(|\mathbb{i}|)})$ outputs $\mathbb{w}$ such that $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \in \mathcal{R}$ is at least the probability that $\tilde{\mathbf{P}}$ convinces $\mathbf{V}^{\mathbf{I}(\mathbb{F},\mathbb{i})}(\mathbb{F}, \mathbb{x})$ to accept minus ϵ. Here the notation $\mathbf{E}^{\tilde{\mathbf{P}}}$ means that the extractor $\mathbf{E}$ has black-box access to each of the next-message functions that define the interactive algorithm $\tilde{\mathbf{P}}$. (In particular, the extractor $\mathbf{E}$ can "rewind" the prover $\tilde{\mathbf{P}}$.) Note that since $\mathbf{E}$ receives the proof length $\mathsf{l}(|\mathbb{i}|)$ in unary, $\mathbf{E}$ has enough time to receive, and perform efficient computations on, polynomials output by $\tilde{\mathbf{P}}$.

Zero knowledge. We say that AHP has (perfect) zero knowledge with query bound b and query checker $\mathbf{C}$ if there exists a probabilistic polynomial-time simulator $\mathbf{S}$ such that for every field $\mathbb{F} \in \mathcal{F}$, index-instance-witness tuple $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \in \mathcal{R}$, and $(\mathsf{b}, \mathbf{C})$-query algorithm $\tilde{\mathbf{V}}$ the random variables $\mathrm{View}(\mathbf{P}(\mathbb{F}, \mathbb{i}, \mathbb{x}, \mathbb{w}), \tilde{\mathbf{V}})$ and $\mathbf{S}^{\tilde{\mathbf{V}}}(\mathbb{F}, \mathbb{i}, \mathbb{x})$, defined below, are identical. Here, we say that an algorithm is $(\mathsf{b}, \mathbf{C})$-query if it makes at most b queries to oracles it has access to, and each query individually leads the checker $\mathbf{C}$ to output "ok".

- $\mathrm{View}(\mathbf{P}(\mathbb{F}, \mathbb{i}, \mathbb{x}, \mathbb{w}), \tilde{\mathbf{V}})$ is the *view* of $\tilde{\mathbf{V}}$, namely, is the random variable $(r, a_1, \ldots, a_q)$ where r is $\tilde{\mathbf{V}}$'s randomness and $a_1, \ldots, a_q$ are the responses to $\tilde{\mathbf{V}}$'s queries determined by the oracles sent by $\mathbf{P}(\mathbb{F}, \mathbb{i}, \mathbb{x}, \mathbb{w})$.

- $\mathbf{S}^{\tilde{\mathbf{V}}}(\mathbb{F}, \mathbb{i}, \mathbb{x})$ is the output of $\mathbf{S}(\mathbb{F}, \mathbb{i}, \mathbb{x})$ when given straightline access to $\tilde{\mathbf{V}}$ ($\mathbf{S}$ may interact with $\tilde{\mathbf{V}}$, *without rewinding*, by exchanging messages with $\tilde{\mathbf{V}}$ and answering any oracle queries along the way), *prepended* with $\tilde{\mathbf{V}}$'s randomness r. Note that r could be of super-polynomial size, so $\mathbf{S}$ cannot sample r on $\tilde{\mathbf{V}}$'s behalf and then output it; instead, as in prior work, we restrict $\mathbf{S}$ to not see r, and prepend r to $\mathbf{S}$'s output.

A special case of interest. We **only consider** AHPs that satisfy the following properties.

– *Public coins:* AHP is *public-coin* if each verifier message to the prover is a uniformly random string of some prescribed length (or an empty string). Hence the verifier's randomness is its messages $\rho_1, \ldots, \rho_{\mathsf{k}} \in \mathbb{F}^*$ and possibly additional randomness $\rho_{\mathsf{k}+1} \in \mathbb{F}^*$ used after the interaction. All verifier queries can be postponed, without loss of generality, to a query phase that occurs after the interactive phase with the prover.
– *Non-adaptive queries:* AHP is *non-adaptive* if all of the verifier's query locations are solely determined by the verifier's randomness and inputs (the field $\mathbb{F}$ and the instance $\mathbb{x}$).

Given these properties, we can view the verifier as two subroutines that execute in the query phase: a query algorithm $\mathbf{Q_V}$ that produces query locations based on the verifier's randomness, and a decision algorithm $\mathbf{D_V}$ that accepts or rejects based on the answers to the queries (and the verifier's randomness). In more detail, $\mathbf{Q_V}$ receives as input the field $\mathbb{F}$, the instance $\mathbb{x}$, and randomness $\rho_1, \ldots, \rho_{\mathsf{k}}, \rho_{\mathsf{k}+1}$, and outputs a query set Q consisting of tuples $((i, j), z)$ to be interpreted as "query $p_{i,j}$ at $z \in \mathbb{F}$"; and $\mathbf{D_V}$ receives as input the field $\mathbb{F}$, the instance $\mathbb{x}$, answers $(v_{((i,j),z)})_{((i,j),z)\in Q}$, and randomness $\rho_1, \ldots, \rho_{\mathsf{k}}, \rho_{\mathsf{k}+1}$, and outputs the decision bit.

While the above properties are not strictly necessary for the compiler that we describe in Section 2.8, all "natural" protocols that we are aware of (including those that we construct in this work) satisfy these properties, and so we restrict our attention to public-coin non-adaptive protocols for simplicity.

2.5 AHP for constraint systems

We construct an AHP for *rank-1 constraint satisfiability* (R1CS) that has linear proof length and constant query complexity. Below we define the indexed relation that represents this problem, and then state our result.

Definition 2.5.1 (R1CS indexed relation)**.** *The indexed relation $\mathcal{R}_{\mathrm{R1CS}}$ is the set of all triples*

$$(\mathtt{i}, \mathtt{x}, \mathtt{w}) = \big((\mathbb{F}, H, K, A, B, C), x, w\big)$$

where $\mathbb{F}$ is a finite field, H and K are subsets of $\mathbb{F}$, A, B, C are $H \times H$ matrices over $\mathbb{F}$ with $|K| \geq \max\{\|A\|, \|B\|, \|C\|\}$, and $z := (x, w)$ is a vector in $\mathbb{F}^H$ such that $Az \circ Bz = Cz$.

Theorem 2.5.2. *There exists an AHP for the indexed relation $\mathcal{R}_{\mathrm{R1CS}}$ that is a zero knowledge proof of knowledge with the following features. The indexer uses $O(|K| \log |K|)$ field operations and outputs $O(|K|)$ field elements. The prover and verifier exchange 7 messages. To achieve zero knowledge against b queries (with a query checker $\mathbf{C}$ that rejects queries in H), the prover uses $O((|K| + \mathsf{b}) \log(|K| + \mathsf{b}))$ field operations and outputs a total of $O(|H| + \mathsf{b})$ field elements. The verifier makes $O(1)$ queries to the encoded index and to the prover's messages, has soundness error $O((|K| + \mathsf{b})/|\mathbb{F}|)$, and uses $O(|x| + \log |K|)$ field operations.*

Remark 2.5.3 (restrictions on domains)**.** Our protocol uses the univariate sumcheck of [BCRSVW19] as a subroutine, and in particular inherits the requirement that the domains H and K must be additive or multiplicative subgroups of the field $\mathbb{F}$. For simplicity, in our descriptions we use multiplicative subgroups because we use this case in our implementation; the case of additive subgroups involves only minor modifications. Moreover, the arithmetic complexities for the indexer and prover stated in Theorem 2.5.2 assume that the domains H and K are "FFT-friendly" (e.g., they have smooth sizes); this is not a requirement, since in general the arithmetic complexities will be that of an FFT over the domains H and K. Note that we can assume without loss of generality that $|H| = O(|K|)$, for otherwise (if $|K| < |H|/3$) then are empty rows or columns across the matrices that we can drop and reduce their size. Finally, we assume that $|H| \leq |\mathbb{F}|/2$.

This section is organized as follows: in Section 2.5.1 we introduce algebraic notations and facts used in this section; in Section 2.5.2 we describe an AHP for checking linear relations; and in Section 2.5.3 we build on this latter to obtain an AHP for R1CS.

Throughout we assume that H and K come equipped with bijections $\phi_{_H} : H \to [|H|]$ and $\phi_{_K} : K \to [|K|]$ that are computable in linear time. Moreover, we define the two sets $H[\leq k] := \{\kappa \in H : 1 \leq \phi_{_H}(\kappa) \leq k\}$ and $H[> k] := \{\kappa \in H : \phi_{_H}(\kappa) > k\}$ to denote the first k elements in H and the remaining elements, respectively. We can then write that $x \in \mathbb{F}^{H[\leq |x|]}$ and $w \in \mathbb{F}^{H[> |x|]}$.

2.5.1 Algebraic preliminaries

Polynomial encodings. For a finite field $\mathbb{F}$, subset $S \subseteq \mathbb{F}$, and function $f : S \to \mathbb{F}$ we denote by $\hat{f}$ the (unique) univariate polynomial over $\mathbb{F}$ with degree less than $|S|$ such that $\hat{f}(a) = f(a)$ for every

$a \in S$. We sometimes abuse notation and write $\hat{f}$ to denote *some* polynomial that agrees with f on S, which need not equal the (unique) such polynomial of smallest degree.

Vanishing polynomials. For a finite field $\mathbb{F}$ and subset $S \subseteq \mathbb{F}$, we denote by v_S the unique non-zero monic polynomial of degree at most $|S|$ that is zero everywhere on S; v_S is called the *vanishing polynomial* of S. If S is an additive or multiplicative coset in $\mathbb{F}$ then v_S can be evaluated in $\text{polylog}(|S|)$ field operations. For example, if S is a multiplicative subgroup of $\mathbb{F}$ then $v_S(X) = X^{|S|} - 1$ and, more generally, if S is a ξ-coset of a multiplicative subgroup S_0 (namely, $S = \xi S_0$) then $v_S(X) = \xi^{|S|} v_{S_0}(X/\xi) = X^{|S|} - \xi^{|S|}$; in either case, v_S can be evaluated in $O(\log |S|)$ field operations.

Derivative of vanishing polynomials. We rely on various properties of a bivariate polynomial u_S introduced in [BCGGRS19]. For a finite field $\mathbb{F}$ and subset $S \subseteq \mathbb{F}$, we define

$$u_S(X, Y) := \frac{v_S(X) - v_S(Y)}{X - Y} \ ,$$

which is a polynomial of individual degree $|S| - 1$ because $X - Y$ divides $X^i - Y^i$ for any positive integer i. Note that $u_S(X, X)$ is the formal derivative of the vanishing polynomial $v_S(X)$. The bivariate polynomial $u_S(X, Y)$ satisfies two useful algebraic properties. First, the univariate polynomials $(u_S(X, a))_{a \in S}$ are linearly independent, and $u_S(X, Y)$ is their (unique) low-degree extension. Second, $u_S(X, Y)$ vanishes on the square $S \times S$ except for on the diagonal, where it takes on the (non-zero) values $(u_S(a, a))_{a \in S}$.

If S is an additive or multiplicative coset in $\mathbb{F}$, $u_S(X, Y)$ can be evaluated at any $(\alpha, \beta) \in \mathbb{F}^2$ in $\text{polylog}(|S|)$ field operations because in this case both v_S (and its derivative) can be evaluated in $\text{polylog}(|S|)$ field operations. For example, if S is a multiplicative subgroup then $u_S(X, Y) = (X^{|S|} - Y^{|S|})/(X - Y)$ and $u_S(X, X) = |S|X^{|S|-1}$, so both can be evaluated in $O(\log |S|)$ field operations.

Univariate sumcheck for subgroups. Prior work [BCRSVW19] shows that, given a multiplicative subgroup S of $\mathbb{F}$, a polynomial $f(X)$ sums to σ over S if and only if $f(X)$ can be written as $h(X)v_S(X) + Xg(X) + \sigma/|S|$ for some $h(X)$ and $g(X)$ with $\deg(g) < |S| - 1$. This can be viewed as a univariate sumcheck protocol, and we shall rely on it throughout this section.

2.5.2 AHP for the lincheck problem

The *lincheck problem* for univariate polynomials considers the task of deciding whether two polynomials encode vectors that are linearly related in a prescribed way. In more detail, the problem is parametrized by a field $\mathbb{F}$, two subsets H and K of $\mathbb{F}$, and a matrix $M \in \mathbb{F}^{H \times H}$ with $|K| \geq \|M\| > 0$. Given oracle access to two low-degree polynomials $f_1, f_2 \in \mathbb{F}^{<d}[X]$, the problem asks to decide whether for every $a \in H$ it holds that $f_1(a) = \sum_{b \in H} M_{a,b} \cdot f_2(b)$, by asking a small number of queries to f_1 and f_2. The matrix M thus prescribes the linear relations that relate the values of f_1 and f_2 on H.

Ben-Sasson et al. [BCRSVW19] solve this problem by reducing the lincheck problem to a sumcheck problem, and then reducing the sumcheck problem to low-degree testing (of univariate

polynomials). In particular, this prior work achieves a 2-message algebraic *non-holographic* protocol that solves the lincheck problem with linear proof length and constant query complexity. In this section we show how to achieve a 6-message algebraic *holographic* protocol, again with linear proof length and constant query complexity. In Section 2.5.2.1 we describe the indexer algorithm, in Section 2.5.2.2 we describe the prover and verifier algorithms, and in Section 2.5.2.3 we analyze the protocol. Fig. 2.4 summarizes the protocol.

2.5.2.1 Offline phase: encoding the linear relation

The indexer $\mathbf{I}$ for the lincheck problem receives as input a field $\mathbb{F}$, two subsets H and K of $\mathbb{F}$, and a matrix $M \in \mathbb{F}^{H \times H}$ with $|K| \geq \|M\|$. The non-zero entries of M are assumed to be presented in some canonical order (e.g., row-wise or column-wise). The output of $\mathbf{I}$ is three univariate polynomials $\hat{\mathsf{row}}, \hat{\mathsf{col}}, \hat{\mathsf{val}}$ over $\mathbb{F}$ of degree less than $|K|$ such that the following polynomial is a low-degree extension of M:

$$\hat{M}(X,Y) := \sum_{\kappa \in K} u_H(X, \hat{\mathsf{row}}(\kappa)) u_H(Y, \hat{\mathsf{col}}(\kappa)) \hat{\mathsf{val}}(\kappa) \ . \tag{2.1}$$

The three aforementioned polynomials are the (unique) low-degree extensions of the three functions $\mathsf{row}, \mathsf{col}, \mathsf{val} \colon K \to \mathbb{F}$ that respectively represent the row index, column index, and value of the non-zero entries of the matrix M. In more detail, for every $\kappa \in K$ with $1 \leq \phi_{_K}(\kappa) \leq \|M\|$:

- $\mathsf{row}(\kappa) := \phi_{_H}^{-1}([(]_\kappa)$ where $[(]_\kappa$ is the row index of the $\phi_{_K}(\kappa)$-th nonzero entry in M;
- $\mathsf{col}(\kappa) := \phi_{_H}^{-1}([(]_\kappa)$ where $[(]_\kappa$ is the column index of the $\phi_{_K}(\kappa)$-th nonzero entry in M;
- $\mathsf{val}(\kappa)$ is the value of the $\phi_{_K}(\kappa)$-th nonzero entry in M, divided by $u_H(\mathsf{row}(\kappa), \mathsf{row}(\kappa)) \cdot u_H(\mathsf{col}(\kappa), \mathsf{col}(\kappa))$.

Also, $\mathsf{val}(\kappa)$ returns the element 0 for every $\kappa \in K$ with $\phi_{_K}(\kappa) > \|M\|$, while $\mathsf{row}(\kappa)$ and $\mathsf{col}(\kappa)$ return an arbitrary element in H for such κ. The evaluation tables of these functions can be found in $O(|K| \log |H|)$ operations, from which interpolation yields the desired polynomials in $O(|K| \log |K|)$ operations.

Recall from Section 2.5.1 that the bivariate polynomial $u_H(X,Y)$ vanishes on the square $H \times H$ except for on the diagonal, where it takes on the (non-zero) values $(u_H(a,a))_{a \in H}$. By construction of the polynomials $\hat{\mathsf{row}}, \hat{\mathsf{col}}, \hat{\mathsf{val}}$, the polynomial $\hat{M}(X,Y)$ agrees with the matrix M everywhere on the domain $H \times H$. The individual degree of $\hat{M}(X,Y)$ is less than $|H|$. Thus, $\hat{M}$ is the unique low-degree extension of M.

We rewrite the polynomial $\hat{M}(X,Y)$ in a form that will be useful later:

Claim 2.5.4.

$$\hat{M}(X,Y) = \sum_{\kappa \in K} \frac{v_H(X)}{(X - \hat{\mathsf{row}}(\kappa))} \cdot \frac{v_H(Y)}{(Y - \hat{\mathsf{col}}(\kappa))} \cdot \hat{\mathsf{val}}(\kappa) \ . \tag{2.2}$$

Proof. Note that $v_H(\hat{\mathsf{row}}(\kappa)) = v_H(\hat{\mathsf{col}}(\kappa)) = 0$ for every $\kappa \in K$ because $\hat{\mathsf{row}}(X)$ and $\hat{\mathsf{col}}(X)$ map K to H and v_H vanishes on H. Therefore:

$$\begin{aligned}\hat{M}(X,Y) &= \sum_{\kappa\in K} u_H(X, \hat{\mathsf{row}}(\kappa)) \cdot u_H(Y, \hat{\mathsf{col}}(\kappa)) \cdot \hat{\mathsf{val}}(\kappa) \\ &= \sum_{\kappa\in K} \frac{v_H(X) - v_H(\hat{\mathsf{row}}(\kappa))}{X - \hat{\mathsf{row}}(\kappa)} \cdot \frac{v_H(Y) - v_H(\hat{\mathsf{col}}(\kappa))}{Y - \hat{\mathsf{col}}(\kappa)} \cdot \hat{\mathsf{val}}(\kappa) \\ &= \sum_{\kappa\in K} \frac{v_H(X)}{(X - \hat{\mathsf{row}}(\kappa))} \cdot \frac{v_H(Y)}{(Y - \hat{\mathsf{col}}(\kappa))} \cdot \hat{\mathsf{val}}(\kappa) \ .\end{aligned}$$

□

2.5.2.2 Online phase: proving and verifying the linear relation

The prover $\mathbf{P}$ for the lincheck problem receives as input a field $\mathbb{F}$, two subsets H and K of $\mathbb{F}$, a matrix $M \in \mathbb{F}^{H\times H}$ with $|K| \geq \|M\|$, and two polynomials $f_1, f_2 \in \mathbb{F}^{<d}[X]$. The verifier $\mathbf{V}$ for the lincheck problem receives as input the field $\mathbb{F}$ and two subsets H and K of $\mathbb{F}$; $\mathbf{V}$ also has oracle access to the polynomials $\hat{\mathsf{row}}, \hat{\mathsf{col}}, \hat{\mathsf{val}}$ output by the indexer $\mathbf{I}$ invoked on appropriate inputs.

The protocol begins with a reduction from a lincheck problem to a sumcheck problem: $\mathbf{V}$ samples a random element $\alpha \in \mathbb{F}$ and sends it to $\mathbf{P}$. Indeed, letting $r(X,Y)$ denote the polynomial $u_H(X,Y)$, $\mathbf{P}$ is left to convince $\mathbf{V}$ that the following univariate polynomial sums to 0 on H:

$$q_1(X) := r(\alpha, X) f_1(X) - r_M(\alpha, X) f_2(X) \quad \text{where} \quad r_M(X,Y) := \sum_{\kappa\in H} r(X,\kappa)\hat{M}(\kappa, Y) \ . \tag{2.3}$$

We rely on the univariate sumcheck protocol for this step: $\mathbf{P}$ sends to $\mathbf{V}$ the polynomials $g_1(X)$ and $h_1(X)$ such that $q_1(X) = h_1(X)v_H(X) + Xg_1(X)$. In order to check this polynomial identity, $\mathbf{V}$ samples a random element $\beta_1 \in \mathbb{F}$ with the intention of checking the identity at $X := \beta_1$. For the right-hand side, $\mathbf{V}$ queries g_1 and h_1 at β_1, and then evaluates $h_1(\beta_1)v_H(\beta_1) + \beta_1 g_1(\beta_1)$ in $O(\log|H|)$ operations. For the left-hand side, $\mathbf{V}$ queries f_1 and f_2 at β_1 and then needs to ask help from $\mathbf{P}$ to evaluate $r(\alpha,\beta_1)f_1(\beta_1) - r_M(\alpha,\beta_1)f_2(\beta_1)$. The reason is that while $r(\alpha,\beta_1)$ is easy to evaluate (it requires $O(\log|H|)$ operations), $r_M(\alpha,\beta_1) = \sum_{\kappa\in H} r(\alpha,\kappa)\hat{M}(\kappa,\beta_1)$ in general requires $\Omega(|H||K|)$ operations.

We thus rely on the univariate sumcheck protocol again. We define

$$q_2(X) := r(\alpha, X)\hat{M}(X, \beta_1) \tag{2.4}$$

$\mathbf{V}$ sends β_1 to $\mathbf{P}$, and then $\mathbf{P}$ replies with the sum $\sigma_2 := \sum_{\kappa\in H} r(\alpha,\kappa)\hat{M}(\kappa,\beta_1)$ and the polynomials $g_2(X)$ and $h_2(X)$ such that $q_2(X) = h_2(X)v_H(X) + Xg_2(X) + \sigma_2/|H|$. In order to check this polynomial identity, $\mathbf{V}$ samples a random element $\beta_2 \in \mathbb{F}$ with the intention of checking the identity at $X := \beta_2$. For the right-hand side, $\mathbf{V}$ queries g_2 and h_2 at β_2, and then evaluates $h_2(\beta_2)v_H(\beta_2) + \beta_2 g_2(\beta_2) + \sigma_2/|H|$ in $O(\log|H|)$ operations. To evaluate the left-hand side,

however, $\mathbf{V}$ needs to ask help from $\mathbf{P}$. The reason is that while $r(\alpha, \beta_2)$ is easy to evaluate (it requires $O(\log |H|)$ operations), $\hat{M}(\beta_2, \beta_1)$ in general requires $\Omega(|K|)$ operations.

We thus rely on the univariate sumcheck protocol (yet) again: $\mathbf{V}$ sends β_2 to $\mathbf{P}$, and then $\mathbf{P}$ replies with the value $\sigma_3 := \hat{M}(\beta_2, \beta_1)$, which the verifier must check. Note though that we *cannot* use the sumcheck protocol directly to compute the sum obtained from Eq. (2.1):

$$\hat{M}(\beta_2, \beta_1) = \sum_{\kappa \in K} u_H(\beta_2, \hat{\mathsf{row}}(\kappa)) u_H(\beta_1, \hat{\mathsf{col}}(\kappa)) \hat{\mathsf{val}}(\kappa) \ .$$

This is because the degree of the above addend, if we replace κ with an indeterminate, is $\Omega(|H||K|)$, which means that the degree of the polynomial h_3 sent as part of a sumcheck protocol also has degree $\Omega(|H||K|)$, which is not within our budget of an AHP with proof length $O(|H| + |K|)$. Instead, we make the minor modification that in the earlier rounds β_1 and β_2 are sampled from $\mathbb{F} \setminus H$ instead of $\mathbb{F}$, and we will leverage the sumcheck protocol to verify the equivalent (well-defined) expression from Eq. (2.2):

$$\hat{M}(\beta_2, \beta_1) = \sum_{\kappa \in K} \frac{v_H(\beta_2) v_H(\beta_1) \hat{\mathsf{val}}(\kappa)}{(\beta_2 - \hat{\mathsf{row}}(\kappa))(\beta_1 - \hat{\mathsf{col}}(\kappa))} \ .$$

This may appear to be an odd choice, because if we replace κ with an indeterminate in the sum above, we obtain a rational function that is (in general) *not a polynomial*, and so does not immediately fit the sumcheck protocol. Nevertheless, we are still able to use the sumcheck protocol with it, as we now explain.

Define $f_3(X)$ to be the (unique) polynomial of degree less than $|K|$ such that

$$\forall\, \kappa \in K\ ,\ f_3(\kappa) = \frac{v_H(\beta_2) v_H(\beta_1) \hat{\mathsf{val}}(\kappa)}{(\beta_2 - \hat{\mathsf{row}}(\kappa))(\beta_1 - \hat{\mathsf{col}}(\kappa))} \ . \tag{2.5}$$

The prover computes the polynomials $g_3(X)$ and $h_3(X)$ such that

$$\begin{aligned} &f_3(X) = X g_3(X) + \sigma_3/|K| \ , \\ &v_H(\beta_2) v_H(\beta_1) \hat{\mathsf{val}}(X) - (\beta_2 - \hat{\mathsf{row}}(X))(\beta_1 - \hat{\mathsf{col}}(X)) f_3(X) = h_3(X) v_K(X) \ . \end{aligned}$$

The first equation demonstrates that f_3 sums to σ_3 over K, and the second equation demonstrates that f_3 agrees with the correct addends over K. These two equations can be combined in a single equation that involves only $g_3(X)$ and $h_3(X)$:

$$v_H(\beta_2) v_H(\beta_1) \hat{\mathsf{val}}(X) - (\beta_2 - \hat{\mathsf{row}}(X))(\beta_1 - \hat{\mathsf{col}}(X))(X g_3(X) + \sigma_3/|K|) = h_3(X) v_K(X) \ .$$

The prover thus only sends the two polynomials $g_3(X)$ and $h_3(X)$. In order to check this polynomial identity, $\mathbf{V}$ samples a random element $\beta_3 \in \mathbb{F}$ with the intention of checking the identity at $X := \beta_3$. Then $\mathbf{V}$ queries g_3, h_3, $\hat{\mathsf{row}}$, $\hat{\mathsf{col}}$, $\hat{\mathsf{val}}$ at β_3, and then evaluates $v_H(\beta_2) v_H(\beta_1) \hat{\mathsf{val}}(\beta_3) - (\beta_2 - \hat{\mathsf{row}}(\beta_3))(\beta_1 - \hat{\mathsf{col}}(\beta_3))(\beta_3 g_3(\beta_3) + \sigma_3/|K|) = h_3(\beta_3) v_K(\beta_3)$ in $O(\log |K|)$ operations.

If this third test passes then $\mathbf{V}$ can use the value σ_3 in place of $\hat{M}(\beta_2, \beta_1)$ to finish the second test. If this latter passes, $\mathbf{V}$ can in turn use the value σ_2 in place of $r_M(\alpha, \beta_1)$ to finish the first test.

2.5.2.3 Analysis

Soundness. We argue that the soundness error is at most

$$\frac{|H| + 3|K|}{|\mathbb{F}|} + \frac{d + 3|H|}{|\mathbb{F} \setminus H|} \ .$$

There are four ways in which the verifier could still accept if the lincheck statement is false: if the randomized reduction to the first sumcheck produces a polynomial that sums to zero; or if any one of the three sumchecks accepts despite the claimed sum being incorrect. The probability that the randomized reduction to sumcheck fails is at most the individual degree in X of $r(X, Y)$ divided by $|\mathbb{F}|$, which is less than $|H|/|\mathbb{F}|$. The probability that any one of the sumchecks fail to detect an incorrectly declared sum is at most the maximum degree of the polynomial equation tested in the respective sumcheck divided by the size from which the test element is sampled. The innermost sumcheck has maximum degree less than $3|K|$, the intermediate sumcheck has maximum degree less than $2|H|$, and the outermost sumcheck has maximum degree less than $|H| + d$. These errors add up to the soundness error claimed above.

Efficiency. The protocol consists of 6 messages, with the verifier moving first. The verifier makes a constant number of queries, evaluates v_H and v_K at a constant number of locations, and then performs a constant number of field operations. In particular, the arithmetic complexity of the verifier is $O(\log |H| + \log |K|)$. The prover sends a constant number of polynomials with degrees linearly related to d (the bound on the degrees of f_1 and f_2), $|H|$, and $|K|$. We now argue that prover time is $O((|H| + \mathsf{d}) \log(|H| + \mathsf{d}) + |K| \log |K|)$. In the first round, the prover sends the coefficients of the polynomials $g_1(X)$ and $h_1(X)$, which can be found in time $O(|K| + (|H| + \mathsf{d}) \log(|H| + \mathsf{d}))$, as we argue in Lemma 2.5.5. In the second round, the prover sends the field element σ_2 and the polynomials $g_2(X)$ and $h_2(X)$, which can be found in time $O(|K| + |H| \log |H|)$, as we argue in Lemma 2.5.6. In the third round, the prover sends the field element σ_3 and the polynomials $g_3(X)$ and $h_3(X)$, which can be found in time $O(|K| \log |K|)$, as we argue in Lemma 2.5.7.

Lemma 2.5.5 (first round)**.** *The coefficients of the polynomials $g_1(X)$ and $h_1(X)$ can be found in $O(|K| + (|H| + \mathsf{d}) \log(|H| + \mathsf{d}))$ field operations, when given coefficients of the polynomials $f_1(X)$ and $f_2(X)$, the subsets H and K, and the matrix M (in sparse form).*

Proof. It suffices to find the coefficients of the polynomial $q_1(X)$ from Eq. (2.3), which has degree at most $|H| + \mathsf{d} - 2$, because the polynomials $g_1(X)$ and $h_1(X)$ can be found via polynomial long division of $q_1(X)$ by v_H in time $O((|H| + \mathsf{d}) \log |H|)$. In turn, $q_1(X)$ can be computed from the coefficients of $f_1(X)$, $f_2(X)$, $r(\alpha, X)$, and $r_M(\alpha, X)$ in time $O((|H| + \mathsf{d}) \log(|H| + \mathsf{d}))$ via fast polynomial multiplication and polynomial addition. The first two are given to us in coefficient form; to find the coefficients of the latter two polynomials, we can evaluate each of them over H and then interpolate.

The values of $r(\alpha, X)$ on H can be obtained in $O(|H| \log |H|)$ operations via direct computation of formulas described in Section 2.5.1. The problem is now reduced to finding the values of $r_M(\alpha, X)$ on H — this is the "hard part" that motivates the present proof.

Observe that, by definition of r_M (see Eq. (2.3)) and $\hat{M}$ (see Eq. (2.1)), the following holds:

$$\begin{aligned} r_M(\alpha, X) &= \sum_{\kappa_1 \in H} r(\alpha, \kappa_1) \sum_{\kappa_2 \in K} u_H(\kappa_1, \hat{\mathsf{row}}(\kappa_2)) u_H(X, \hat{\mathsf{col}}(\kappa_2)) \hat{\mathsf{val}}(\kappa_2) \\ &= \sum_{\kappa_2 \in K} u_H(X, \hat{\mathsf{col}}(\kappa_2)) \hat{\mathsf{val}}(\kappa_2) \sum_{\kappa_1 \in H} r(\alpha, \kappa_1) u_H(\kappa_1, \hat{\mathsf{row}}(\kappa_2)) \\ &= \sum_{\kappa_2 \in K} u_H(X, \hat{\mathsf{col}}(\kappa_2)) \hat{\mathsf{val}}(\kappa_2) r(\alpha, \hat{\mathsf{row}}(\kappa_2)) u_H(\hat{\mathsf{row}}(\kappa_2), \hat{\mathsf{row}}(\kappa_2)) \ . \end{aligned}$$

The last equality uses the fact that for every $\kappa_2 \in K$, the sum $\sum_{\kappa_1 \in H} r(\alpha, \kappa_1) u_H(\kappa_1, \hat{\mathsf{row}}(\kappa_2))$ collapses to a single term corresponding to $\kappa_1 = \hat{\mathsf{row}}(\kappa_2)$; the other terms, which correspond to $\kappa_1 \neq \hat{\mathsf{row}}(\kappa_2)$, are zero due to the fact that the polynomial u_H vanishes on the square $H \times H$ except for on its diagonal.

Next, again using the fact that u_H vanishes on the square $H \times H$ except for on its diagonal, we note that for every $\kappa_1 \in H$

$$r_M(\alpha, \kappa_1) = \sum_{\kappa_2 \in K \text{ s.t. } \hat{\mathsf{col}}(\kappa_2) = \kappa_1} u_H(\kappa_1, \hat{\mathsf{col}}(\kappa_2)) \hat{\mathsf{val}}(\kappa_2) \cdot r(\alpha, \hat{\mathsf{row}}(\kappa_2)) u_H(\hat{\mathsf{row}}(\kappa_2), \hat{\mathsf{row}}(\kappa_2)) \ .$$

In other words, as κ_1 ranges over H, each element of the sum in $r_M(\alpha, \kappa_1)$ contributes a nonzero value precisely when κ_1 equals a particular element of H, namely, when $\kappa_1 = \hat{\mathsf{col}}(\kappa_2)$. Also, since κ_2 ranges only in K, $\hat{\mathsf{row}}(\kappa_2) = \mathsf{row}(\kappa_2)$, $\hat{\mathsf{col}}(\kappa_2) = \mathsf{col}(\kappa_2)$, and $\hat{\mathsf{val}}(\kappa_2) = \mathsf{val}(\kappa_2)$ are just the row index, column index, and value of the κ_2-th entry of M (or zero).

This immediately leads to the following strategy to finding the values of $r_M(\alpha, X)$ on H. Initialize for each $\kappa_1 \in H$ a variable for $r_M(\alpha, \kappa_1)$ that is initially set to 0. Then, for each $\kappa_2 \in K$, compute the term $u_H(\mathsf{col}(\kappa_2), \mathsf{col}(\kappa_2))\mathsf{val}(\kappa_2) r(\alpha, \mathsf{row}(\kappa_2)) u_H(\mathsf{row}(\kappa_2), \mathsf{row}(\kappa_2))$ and add it to the variable for $r_M(\alpha, \mathsf{col}(\kappa_2))$. Since the values $(u_H(\kappa_1, \kappa_1))_{\kappa_1 \in H}$ and $(r(\alpha, \kappa_1))_{\kappa_1 \in H}$ can be precomputed in $O(|H| \log |H|)$ operations, the foregoing strategy can be carried out in $O(|K| + |H| \log |H|)$ operations. □

Lemma 2.5.6 (second round)**.** *The field element σ_2 and the coefficients of the polynomials $g_2(X)$ and $h_2(X)$ can be found in $O(|K| + |H| \log |H|)$ field operations, when given the subsets H and K and the matrix M (in sparse form).*

Proof. It suffices to find the coefficients of the polynomial $q_2(X)$ from Eq. (2.4), which has degree at most $2|H| - 2$, because the polynomials $g_2(X)$ and $h_2(X)$ can be found via polynomial long division of $q_2(X)$ by v_H in time $O(|H| \log |H|)$, and the sum σ_2 can be found by evaluating $q_2(X)$ over H in time $O(|H| \log |H|)$ and summing in time $O(|H|)$. In turn, $q_2(X)$ can be computed from the coefficients of $r(\alpha, X)$ and of $\hat{M}(X, \beta_1)$ in time $O(|H| \log |H|)$ using fast polynomial multiplication. To find the coefficients of these two polynomials, we can evaluate each of them over H and then interpolate. The values of $r(\alpha, X)$ on H can be obtained in $O(|H| \log |H|)$ operations. We now need to find the values of $\hat{M}(X, \beta_1)$ on H.

Recall that

$$\hat{M}(X, \beta_1) = \sum_{\kappa \in K} u_H(X, \hat{\mathsf{row}}(\kappa)) u_H(\beta_1, \hat{\mathsf{col}}(\kappa)) \hat{\mathsf{val}}(\kappa) \ .$$

Using the fact that u_H vanishes on the square $H \times H$ except for the diagonal, we note that for every $\kappa_1 \in H$

$$\hat{M}(\kappa_1, \beta_1) = \sum_{\kappa_2 \in K \text{ s.t. } \hat{\mathsf{row}}(\kappa_2) = \kappa_1} u_H(\kappa_1, \hat{\mathsf{row}}(\kappa_2)) u_H(\beta_1, \hat{\mathsf{col}}(\kappa_2)) \hat{\mathsf{val}}(\kappa_2) \ .$$

Thus, to find the values of $\hat{M}(X, \beta_1)$ on H, we initialize for each $\kappa_1 \in H$ a variable for $\hat{M}(\kappa_1, \beta_1)$ that is initially 0. Then, $\forall \kappa_2 \in K$, we compute the term $u_H(\hat{\mathsf{row}}(\kappa_2), \hat{\mathsf{row}}(\kappa_2)) u_H(\beta_1, \hat{\mathsf{col}}(\kappa_2)) \hat{\mathsf{val}}(\kappa_2)$ and add it to the variable for $\hat{M}(\hat{\mathsf{row}}(\kappa_2), \beta_1)$. Since the values $(u_H(\kappa, \kappa))_{\kappa \in H}$ and $(u_H(\beta_1, \kappa))_{\kappa \in H}$ can be precomputed in $O(|H| \log |H|)$ operations, the foregoing strategy can be carried out in $O(|K| + |H| \log |H|)$ operations. □

Lemma 2.5.7 (third round)**.** *The field element σ_3 and the coefficients of the polynomials $g_3(X)$ and $h_3(X)$ can be found in $O(|K| \log |K|)$ field operations, when given the subsets H and K and the matrix M (in sparse form).*

Proof. First, we find the coefficients of the polynomial $f_3(X)$ from Eq. (2.5), which has degree at most $|K| - 1$. We traverse the matrix M to find the values of $\hat{\mathsf{row}}(\kappa) = \mathsf{row}(\kappa)$, $\hat{\mathsf{col}}(\kappa) = \mathsf{col}(\kappa)$, and $\hat{\mathsf{val}}(\kappa) = \mathsf{val}(\kappa)$, for every $\kappa \in K$. Then, for each $\kappa \in K$, we calculate $f_3(\kappa) = \frac{v_H(\beta_2) v_H(\beta_1) \hat{\mathsf{val}}(\kappa)}{(\beta_2 - \hat{\mathsf{row}}(\kappa))(\beta_1 - \hat{\mathsf{col}}(\kappa))}$, and interpolate those $|K|$ values, in time $O(|K| \log |K|)$. Those values can also be summed, in time $O(|K|)$, to obtain σ_3. Then $g_3(X)$ can be found easily, by subtracting $\sigma_3/|K|$ from $f_3(X)$ and dividing by X.

Next, the prover interpolates the values from M to find the three polynomials $\hat{\mathsf{row}}$, $\hat{\mathsf{col}}$, and $\hat{\mathsf{val}}$. Using fast polynomial multiplication, the prover calculates $v_H(\beta_2) v_H(\beta_1) \hat{\mathsf{val}}(X) - (\beta_2 - \hat{\mathsf{row}}(X))(\beta_1 - \hat{\mathsf{col}}(X)) f_3(X)$, and divides this polynomial by $v_K(X)$ to find $h_3(X)$. This too can be done in time $O(|K| \log |K|)$. □

2.5.3 AHP for R1CS

We prove Theorem 2.5.2. In Section 2.5.3.1 we describe the indexer algorithm, in Section 2.5.3.2 we describe the prover and verifier algorithms, and in Section 2.5.3.3 we analyze the protocol. Fig. 2.5 summarizes the protocol.

The AHP for R1CS directly builds on the AHP for the lincheck problem, analogously to how in [BCRSVW19] the non-holographic protocol for R1CS builds on the non-holographic lincheck protocol. The three lincheck problems associated to the three matrices in the index are bundled together via random coefficients, while the entry-wise product is checked with a polynomial identity. Zero knowledge is achieved via bounded independence and random masks [BCGV16; BCRSVW19]. Consistency with the instance is achieved by having the verifier combine a low-degree extension of the instance and the low-degree extension of the (alleged) witness sent by the prover, in order to create a low-degree extension of the full assignment.

2.5.3.1 Offline phase: encoding the constraint system

The indexer $\mathbf{I}$ for R1CS receives as input a field $\mathbb{F}$, two subsets H and K of $\mathbb{F}$, and three matrices $A, B, C \in \mathbb{F}^{H \times H}$ with $|K| \geq \max\{\|A\|, \|B\|, \|C\|\}$. The non-zero entries of A, B, C are assumed to be presented in some common canonical order. The output of $\mathbf{I}$ consists of the output of the lincheck indexer separately invoked on A, B, C. This produces nine univariate polynomials $\{\hat{\mathsf{row}}_M, \hat{\mathsf{col}}_M, \hat{\mathsf{val}}_M\}_{M \in \{A,B,C\}}$ over $\mathbb{F}$ of degree less than $|K|$ that can be used to compute the low-degree extensions of A, B, C.

2.5.3.2 Online phase: proving and verifying satisfiability

The prover $\mathbf{P}$ for R1CS receives as input a field $\mathbb{F}$, two subsets H and K of $\mathbb{F}$, three matrices $A, B, C \in \mathbb{F}^{H \times H}$ with $|K| \geq \max\{\|A\|, \|B\|, \|C\|\}$, input $x \in \mathbb{F}^{H[\leq |x|]}$, and witness $w \in \mathbb{F}^{H[> |x|]}$. The verifier $\mathbf{V}$ for R1CS receives as input the field $\mathbb{F}$, two subsets H and K of $\mathbb{F}$, and input $x \in \mathbb{F}^{H[\leq |x|]}$; $\mathbf{V}$ also has oracle access to the polynomials $\{\hat{\mathsf{row}}_M, \hat{\mathsf{col}}_M, \hat{\mathsf{val}}_M\}_{M \in \{A,B,C\}}$ output by the indexer $\mathbf{I}$ invoked on appropriate inputs.

The protocol begins with the prover sending randomized encodings for (a certain shift of) the assignment and its linear combinations. Define $\hat{x}(X)$ to be the polynomial of degree less than $|x|$ that agrees with the instance x in $H[\leq |x|]$. Define the shifted witness $\bar{w}\colon H[> |x|] \to \mathbb{F}$ according to the equation

$$\forall\, \gamma\,,\ \bar{w}(\gamma) := \frac{w(\gamma) - \hat{x}(\gamma)}{v_{H[\leq |x|]}(\gamma)}\ .$$

The prover $\mathbf{P}$ sends to $\mathbf{V}$ a random $\hat{w}(X) \in \mathbb{F}^{<|w|+\mathsf{b}}[X]$ that agrees with $\bar{w}$ on $H[> |x|]$; $\mathbf{P}$ also sets $z := (x, w) \in \mathbb{F}^H$ to be the full assignment, computes the three linear combinations $z_A := Az$, $z_B := Bz$, and $z_C := Cz$, and sends to $\mathbf{V}$ random $\hat{z}_A(X), \hat{z}_B(X), \hat{z}_C(X) \in \mathbb{F}^{<|H|+\mathsf{b}}[X]$ that agree with z_A, z_B, z_C on H. Note that the values of up to b locations in each of $\hat{w}(X), \hat{z}_A(X), \hat{z}_B(X), \hat{z}_C(X)$ reveal no information about the witness w, provided the locations are in $\mathbb{F} \setminus H$. Note also that $\hat{z}(X) := \hat{w}(X) v_{H[\leq |x|]}(X) + \hat{x}(X)$ agrees with z on H; moreover, $\mathbf{V}$ can evaluate $\hat{z}(X)$ at any location γ with $O(|x|)$ operations by querying $\hat{w}$ at γ and computing the expression $\hat{w}(\gamma) v_{H[\leq |x|]}(\gamma) + \hat{x}(\gamma)$ by using x.

The rest of the protocol is for $\mathbf{P}$ to convince $\mathbf{V}$ that $z_A \circ z_B = z_C$ and also that z_A, z_B, z_C are obtained as linear combinations from z.

In the same message as above, $\mathbf{P}$ also sends to $\mathbf{V}$ the polynomial $h_0(X)$ such that $\hat{z}_A(X)\hat{z}_B(X) - \hat{z}_C(X) = h_0(X) v_H(X)$. In addition, $\mathbf{P}$ sends to $\mathbf{V}$ a (fully) random $s(X) \in \mathbb{F}^{<2|H|+\mathsf{b}-1}[X]$ and its sum $\sigma_1 := \sum_{\kappa \in H} s(\kappa)$ over H. This random polynomial will be used as a "mask" to make the univariate sumcheck zero knowledge.

Next, $\mathbf{V}$ samples random elements $\alpha, \eta_A, \eta_B, \eta_C \in \mathbb{F}$ and sends them to $\mathbf{P}$. The element α is used to reduce lincheck problems to sumcheck, while the elements η_A, η_B, η_C are used to bundle the three sumcheck problems into one. Indeed, $\mathbf{P}$ is left to convince $\mathbf{V}$ that the following univariate

polynomial sums to σ_1 on H:

$$q_1(X) := \; s(X) + r(\alpha, X)\left(\sum_{M\in\{A,B,C\}} \eta_M \hat{z_M}(X)\right) - \left(\sum_{M\in\{A,B,C\}} \eta_M r_M(\alpha, X)\right)\hat{z}(X) \quad (2.6)$$

where $r_M(X,Y) := \sum_{\kappa\in H} r(X,\kappa)\hat{M}(\kappa, Y)$.

We now rely on the univariate sumcheck protocol: $\mathbf{P}$ sends to $\mathbf{V}$ the polynomials $g_1(X)$ and $h_1(X)$ such that $q_1(X) = h_1(X)v_H(X) + Xg_1(X)$. In order to check this polynomial identity, $\mathbf{V}$ samples a random element $\beta_1 \in \mathbb{F}\setminus H$ with the intention of checking the identity at $X := \beta_1$. For the right-hand side, $\mathbf{V}$ queries g_1 and h_1 at β_1 and then evaluates $h_1(\beta_1)v_H(\beta_1) + \beta_1 g_1(\beta_1)$ in $O(\log|H|)$ operations. For the left-hand side, $\mathbf{V}$ queries $s, \hat{z}_A, \hat{z}_B, \hat{z}_C, \hat{w}$ at β_1 and then needs to ask help from $\mathbf{P}$ to evaluate $q_1(\beta_1)$. The reason is that the term $\eta_A r_A(\alpha,\beta_1) + \eta_B r_B(\alpha,\beta_1) + \eta_C r_C(\alpha,\beta_1)$ in general requires $\Omega(|H||K|)$ operations to compute.

Observe that

$$\begin{aligned}
&\eta_A r_A(\alpha,\beta_1) + \eta_B r_B(\alpha,\beta_1) + \eta_C r_C(\alpha,\beta_1)\\
&= \eta_A \sum_{\kappa\in H} r(\alpha,\kappa)\hat{A}(\kappa,\beta_1) + \eta_B \sum_{\kappa\in H} r(\alpha,\kappa)\hat{B}(\kappa,\beta_1) + \eta_C \sum_{\kappa\in H} r(\alpha,\kappa)\hat{C}(\kappa,\beta_1)\\
&= \sum_{\kappa\in H} r(\alpha,\kappa)(\eta_A\hat{A}(\kappa,\beta_1) + \eta_B\hat{B}(\kappa,\beta_1) + \eta_C\hat{C}(\kappa,\beta_1)) \;\; .
\end{aligned}$$

We define the polynomial

$$q_2(X) := r(\alpha, X)(\eta_A\hat{A}(X,\beta_1) + \eta_B\hat{B}(X,\beta_1) + \eta_C\hat{C}(X,\beta_1)) \quad (2.7)$$

and rely on the univariate sumcheck protocol again: $\mathbf{V}$ sends β_1 to $\mathbf{P}$, and then $\mathbf{P}$ replies with the sum $\sigma_2 := \sum_{\kappa\in H} q_2(\kappa)$ and the polynomials $g_2(X)$ and $h_2(X)$ such that $q_2(X) = h_2(X)v_H(X) + Xg_2(X) + \sigma_2/|H|$. In order to check this polynomial identity, $\mathbf{V}$ samples a random element $\beta_2 \in \mathbb{F}\setminus H$ with the intention of checking the identity at $X := \beta_2$. (Excluding H is needed later in the protocol, as discussed below.) For the right-hand side, $\mathbf{V}$ queries g_2 and h_2 at β_2, and then evaluates $h_2(\beta_2)v_H(\beta_2) + \beta_2 g_2(\beta_2) + \sigma_2/|H|$ in $O(\log|H|)$ operations. To evaluate the left-hand side, however, $\mathbf{V}$ needs to ask help from $\mathbf{P}$. The reason is that while $r(\alpha,\beta_2)$ is easy to evaluate (it requires $O(\log|H|)$ operations), each term $\hat{M}(\beta_2,\beta_1)$ in general requires $\Omega(|K|)$ operations.

We thus rely on the univariate sumcheck protocol (yet) again: $\mathbf{V}$ sends β_2 to $\mathbf{P}$, and then $\mathbf{P}$ replies with the value $\sigma_3 := \eta_A\hat{A}(\beta_2,\beta_1) + \eta_B\hat{B}(\beta_2,\beta_1) + \eta_C\hat{C}(\beta_2,\beta_1)$, which the verifier much check. Observe that

$$\eta_A\hat{A}(\beta_2,\beta_1) + \eta_B\hat{B}(\beta_2,\beta_1) + \eta_C\hat{C}(\beta_2,\beta_1) = \sum_{\kappa\in K}\sum_{M\in\{A,B,C\}} \eta_M \frac{v_H(\beta_2)v_H(\beta_1)\hat{\mathsf{val}}_M(\kappa)}{(\beta_2 - \hat{\mathsf{row}}_M(\kappa))(\beta_1 - \hat{\mathsf{col}}_M(\kappa))} \;\; .$$

Define $f_3(X)$ to be the (unique) polynomial of degree less than $|K|$ such that

$$\forall\, \kappa \in K\,,\; f_3(\kappa) = \sum_{M\in\{A,B,C\}} \eta_M \frac{v_H(\beta_2)v_H(\beta_1)\hat{\mathsf{val}}_M(\kappa)}{(\beta_2 - \hat{\mathsf{row}}_M(\kappa))(\beta_1 - \hat{\mathsf{col}}_M(\kappa))} \;\; . \quad (2.8)$$

The prover computes the polynomials $g_3(X)$ and $h_3(X)$ such that

$$f_3(X) = Xg_3(X) + \sigma_3/|K| \quad \text{and} \quad a(X) - b(X)f_3(X) = h_3(X)v_K(X)$$

where

$$a(X) := \sum_{M \in \{A,B,C\}} \eta_M v_H(\beta_2) v_H(\beta_1) \hat{\mathsf{val}}_M(X) \prod_{N \in \{A,B,C\} \setminus \{M\}} (\beta_2 - \hat{\mathsf{row}}_N(X))(\beta_1 - \hat{\mathsf{col}}_N(X)) \ ,$$

$$b(X) := \prod_{M \in \{A,B,C\}} (\beta_2 - \hat{\mathsf{row}}_M(X))(\beta_1 - \hat{\mathsf{col}}_M(X)) \ .$$

The first equation demonstrates that f_3 sums to σ_3 over K, and the second equation demonstrates that f_3 agrees with the correct addends over K. These two equations can be combined in a single equation that involves only $g_3(X)$ and $h_3(X)$:

$$a(X) - b(X)(Xg_3(X) + \sigma_3/|K|) = h_3(X)v_K(X) \ .$$

The prover thus only sends the two polynomials $g_3(X)$ and $h_3(X)$. In order to check this polynomial identity, $\mathbf{V}$ samples a random element $\beta_3 \in \mathbb{F}$ with the intention of checking the identity at $X := \beta_3$. Then $\mathbf{V}$ queries g_3, h_3, $\{\hat{\mathsf{row}}_M, \hat{\mathsf{col}}_M, \hat{\mathsf{val}}_M\}_{M \in \{A,B,C\}}$ at β_3, and checks the identity in $O(\log |H|)$ operations.

If this third test passes then $\mathbf{V}$ can use the value σ_3 in place of $\sum_{M \in \{A,B,C\}} \eta_M \hat{M}(\beta_2, \beta_1)$ to finish the second test. If this latter passes, $\mathbf{V}$ can in turn use the value σ_2 in place of $\sum_{M \in \{A,B,C\}} \eta_M r_M(\alpha, \beta_1)$ to finish the first test.

2.5.3.3 Analysis

Soundness. We argue that the soundness error is at most

$$\max \left\{ \frac{2|H| + 2\mathsf{b}}{|\mathbb{F}|} \ , \ \frac{3|K| + |H| + 1}{|\mathbb{F}|} + \frac{4|H| + \mathsf{b}}{|\mathbb{F} \setminus H|} \right\} \ .$$

Suppose that for the given index $\mathbb{i} = (\mathbb{F}, H, K, A, B, C)$ and instance $\mathbb{x} = x$ there is no witness $\mathbb{w} = w$ such that $Az \circ Bz = Cz$ for $z := (x, w)$ is a vector in $\mathbb{F}^H$. In particular, this holds for the witness w that is encoded in the polynomial $\hat{w}(X)$ sent by the prover. Let z_A, z_B, z_C be the vectors encoded in the polynomials $\hat{z}_A(X), \hat{z}_B(X), \hat{z}_C(X)$ sent by the prover, respectively. We know that either $z_A \circ z_B \neq z_C$ or one of z_A, z_B, z_C is not the correct linear combination of z. In the first case, the polynomial identity $\hat{z}_A \hat{z}_B - \hat{z}_C = h_0 v_H$ does not hold, so the probability that the verifier still accepts is at most $(2|H| + 2\mathsf{b})/|\mathbb{F}|$. In the second case, we rely on the randomized reduction to sumcheck, which fails with probability at most $(|H| + 1)/|\mathbb{F}|$. Next we have to account for the soundness errors of the three sequential sumchecks, which are bounded by the maximum degree in the respective polynomial equation divided by the size of the set from which the test point is chosen. Thus, the innermost sumcheck has soundness error at most $3|K|/|\mathbb{F}|$; the intermediate sumcheck

has soundness error at most $2|H|/(|\mathbb{F} \setminus H|)$; the outermost sumcheck has soundness error at most $(2|H| + \mathsf{b})/(|\mathbb{F} \setminus H|)$.

Proof of knowledge. If the verifier accepts with probability greater than the soundness error argued above, then the prover's polynomial $\hat{w}$ must encode a valid witness w.

Zero knowledge. We only sketch the intuition because a full proof (which includes constructing a simulator) is similar to the non-holographic setting described in [BCRSVW19]. The first message of the prover includes an encoding of the witness and encodings of its linear combinations. These encodings are protected against up to b queries outside H because the encodings are b-wise independent over $\mathbb{F} \setminus H$. The first message also includes the polynomial $h_0(X)$, which in fact is b-wise independent everywhere on $\mathbb{F}$. Subsequent messages from the prover do not reveal any further information because they are produced for a sumcheck instance that is shifted by a random polynomial (the polynomial $s(X)$). This leads to (perfect) zero knowledge with query bound b and a query checker $\mathbf{C}$ that rejects any query to any of $\hat{w}(X), \hat{z}_A(X), \hat{z}_B(X), \hat{z}_C(X)$ that lies in H.

Efficiency. The indexer computes and outputs a constant number of polynomials of degree less than $|K|$, using time $O(|K| \log |K|)$. The subsequent protocol between the prover and verifier consists of 7 messages, with the prover moving first. The verifier makes a constant number of queries, evaluates $\hat{x}, v_H, v_K$ at a constant number of locations, and then performs a constant number of field operations. Thus, verifier time is $O(|x| + \log |H| + \log |K|)$. The prover sends a constant number of polynomials whose degree is linearly related to $|H| + \mathsf{b}$ or $|K|$. In the first round, the prover computes the linear combinations Az, Bz, Cz and interpolates them, which can be done in time $O(|K| + (|H| + \mathsf{b}) \log(|H| + \mathsf{b}))$; in the second round, the prover finds the coefficients of the polynomials $g_1(X)$ and $h_1(X)$ in time $O(|K| + (|H| + \mathsf{b}) \log(|H| + \mathsf{b}))$, similarly to the proof of Lemma 2.5.5; in the third round, the prover finds the sum σ_2 and the coefficients of $g_2(X)$ and $h_2(X)$ in time $O(|K| + |H| \log |H|)$, similarly to the proof of Lemma 2.5.6; and in the final round, the prover finds the sum σ_3 and the coefficients of $g_3(X)$ and $h_3(X)$ in time $O(|K| \log |K|)$, similarly to the proof of Lemma 2.5.7. Thus, prover time is $O((|H| + \mathsf{b}) \log(|H| + \mathsf{b}) + |K| \log |K|)$, which is $O((|K| + \mathsf{b}) \log(|K| + \mathsf{b}))$ since $|H| = O(|K|)$ (see Remark 2.5.3).

$\mathcal{P}(\mathbb{F}, H, K, M, f_1, f_2)$ $\quad$ $\mathcal{V}^{f_1, f_2, \hat{\mathsf{row}}, \hat{\mathsf{col}}, \hat{\mathsf{val}}}(\mathbb{F}, H, K)$

$\alpha \in \mathbb{F}$ $\longleftarrow$ $\quad$ $\alpha \leftarrow \mathbb{F}$

sumcheck for $r(\alpha, X)f_1(X) - r_M(\alpha, X)f_2(X)$ **over** H
to evaluate $\sum_{\kappa \in H} r(\alpha, \kappa)f_1(\kappa) - r_M(\alpha, \kappa)f_2(\kappa)$

find $g_1(X)$ and $h_1(X)$ such that
$r(\alpha, X)f_1(X) - r_M(\alpha, X)f_2(X)$
$= h_1(X)v_H(X) + Xg_1(X)$

$g_1 \in \mathbb{F}^{<|H|-1}[X], h_1 \in \mathbb{F}^{<\mathsf{d}-1}[X]$ $\longrightarrow$

$\beta_1 \in \mathbb{F}$ $\longleftarrow$ $\quad$ $\beta_1 \leftarrow \mathbb{F} \setminus H$

sumcheck for $r(\alpha, X)\hat{M}(X, \beta_1)$ **over** H
to evaluate $r_M(\alpha, \beta_1) = \sum_{\kappa \in H} r(\alpha, \kappa)\hat{M}(\kappa, \beta_1)$

compute sum $\sigma_2 := \sum_{\kappa \in H} r(\alpha, \kappa)\hat{M}(\kappa, \beta_1)$
and find $g_2(X)$ and $h_2(X)$ such that
$r(\alpha, X)\hat{M}(X, \beta_1)$
$= h_2(X)v_H(X) + Xg_2(X) + \sigma_2/|H|$

$\sigma_2 \in \mathbb{F}, g_2, h_2 \in \mathbb{F}^{<|H|-1}[X]$ $\longrightarrow$

$\beta_2 \in \mathbb{F}$ $\longleftarrow$ $\quad$ $\beta_2 \leftarrow \mathbb{F} \setminus H$

sumcheck for $\frac{v_H(\beta_2)v_H(\beta_1)\hat{\mathsf{val}}(X)}{(\beta_2 - \hat{\mathsf{row}}(X))(\beta_1 - \hat{\mathsf{col}}(X))}$ **over** K
to evaluate $\hat{M}(\beta_2, \beta_1) = \sum_{\kappa \in K} \frac{v_H(\beta_2)v_H(\beta_1)\hat{\mathsf{val}}(\kappa)}{(\beta_2 - \hat{\mathsf{row}}(\kappa))(\beta_1 - \hat{\mathsf{col}}(\kappa))}$

compute sum $\sigma_3 := \sum_{\kappa \in K} \frac{v_H(\beta_2)v_H(\beta_1)\hat{\mathsf{val}}(\kappa)}{(\beta_2 - \hat{\mathsf{row}}(\kappa))(\beta_1 - \hat{\mathsf{col}}(\kappa))}$
and find $g_3(X)$ and $h_3(X)$ such that
$v_H(\beta_2)v_H(\beta_1)\hat{\mathsf{val}}(X) - (\beta_2 - \hat{\mathsf{row}}(X))(\beta_1 - \hat{\mathsf{col}}(X))(Xg_3(X) + \sigma_3/|K|)$
$= h_3(X)v_K(X)$

$\sigma_3 \in \mathbb{F}, g_3 \in \mathbb{F}^{<|K|-1}[X], h_3 \in \mathbb{F}^{<2|K|-2}[X]$ $\longrightarrow$

$\beta_3 \leftarrow \mathbb{F}$
$v_H(\beta_2)v_H(\beta_1)\hat{\mathsf{val}}(\beta_3) - (\beta_2 - \hat{\mathsf{row}}(\beta_3))(\beta_1 - \hat{\mathsf{col}}(\beta_3))(\beta_3 g_3(\beta_3) + \sigma_3/|K|)$
$\stackrel{?}{=} h_3(\beta_3)v_K(\beta_3)$

$r(\alpha, \beta_2)\sigma_3 \stackrel{?}{=} h_2(\beta_2)v_H(\beta_2) + \beta_2 g_2(\beta_2) + \sigma_2/|H|$

$r(\alpha, \beta_1)f_1(\beta_1) - \sigma_2 f_2(\beta_1) \stackrel{?}{=} h_1(\beta_1)v_H(\beta_1) + \beta_1 g_1(\beta_1)$

Figure 2.4: AHP for the lincheck problem.

$\mathcal{P}(\mathbb{F}, H, K, A, B, C, x, w)$ $\quad$ $\mathcal{V}^{\hat{\mathsf{row}}_{\{A,B,C\}}, \hat{\mathsf{col}}_{\{A,B,C\}}, \hat{\mathsf{val}}_{\{A,B,C\}}}(\mathbb{F}, H, K, x)$

$z := (x, w)$ $\; z_A := Az$ $\; z_B := Bz$ $\; z_C := Cz$
sample $\hat{w}(X) \in \mathbb{F}^{<|w|+\mathsf{b}}[X]$ and $\hat{z}_A(X), \hat{z}_B(X), \hat{z}_C(X) \in \mathbb{F}^{<|H|+\mathsf{b}}[X]$
find $h_0(X)$ s.t. $\hat{z}_A(X)\hat{z}_B(X) - \hat{z}_C(X) = h_0(X)v_H(X)$
sample $s(X) \in \mathbb{F}^{<2|H|+\mathsf{b}-1}[X]$ and compute sum $\sigma_1 := \sum_{\kappa \in H} s(\kappa)$

$\sigma_1 \in \mathbb{F}, \hat{w} \in \mathbb{F}^{<|w|+\mathsf{b}}[X], \hat{z}_A, \hat{z}_B, \hat{z}_C \in \mathbb{F}^{<|H|+\mathsf{b}}[X],$
$h_0 \in \mathbb{F}^{<|H|+2\mathsf{b}-1}[X], s \in \mathbb{F}^{<2|H|+\mathsf{b}-1}[X]$ $\longrightarrow$

$\alpha, \eta_A, \eta_B, \eta_C \leftarrow \mathbb{F}$

$\longleftarrow$ $\alpha, \eta_A, \eta_B, \eta_C \in \mathbb{F}$

sumcheck for $s(X) + r(\alpha, X)(\sum_M \eta_M \hat{z_M}(X)) - (\sum_M \eta_M r_M(\alpha, X))\hat{z}(X)$ **over** H

find $g_1(X)$ and $h_1(X)$ such that
$s(X) + r(\alpha, X)(\sum_M \eta_M \hat{z_M}(X)) - (\sum_M \eta_M r_M(\alpha, X))\hat{z}(X)$
$= h_1(X)v_H(X) + Xg_1(X) + \sigma_1/|H|$

$g_1 \in \mathbb{F}^{<|H|-1}[X], h_1 \in \mathbb{F}^{<|H|+\mathsf{b}-1}[X]$ $\longrightarrow$

$\beta_1 \leftarrow \mathbb{F} \setminus H$

$\longleftarrow$ $\beta_1 \in \mathbb{F}$

sumcheck for $r(\alpha, X)(\eta_A \hat{A}(X, \beta_1) + \eta_B \hat{B}(X, \beta_1) + \eta_C \hat{C}(X, \beta_1))$ **over** H

$\sigma_2 := \sum_{\kappa \in H} r(\alpha, \kappa) \sum_{M \in \{A,B,C\}} \eta_M \hat{M}(\kappa, \beta_1)$
and find $g_2(X)$ and $h_2(X)$ such that
$r(\alpha, X) \sum_{M \in \{A,B,C\}} \eta_M \hat{M}(X, \beta_1)$
$= h_2(X)v_H(X) + Xg_2(X) + \sigma_2/|H|$

$\sigma_2 \in \mathbb{F}, g_2, h_2 \in \mathbb{F}^{<|H|-1}[X]$ $\longrightarrow$

$\beta_2 \leftarrow \mathbb{F} \setminus H$

$\longleftarrow$ $\beta_2 \in \mathbb{F}$

sumcheck for $\sum_{M \in \{A,B,C\}} \eta_M \frac{v_H(\beta_2) v_H(\beta_1) \hat{\mathsf{val}}_M(X)}{(\beta_2 - \hat{\mathsf{row}}_M(X))(\beta_1 - \hat{\mathsf{col}}_M(X))}$ **over** K

to evaluate $\eta_A \hat{A}(\beta_2, \beta_1) + \eta_B \hat{B}(\beta_2, \beta_1) + \eta_C \hat{C}(\beta_2, \beta_1)$

$\sigma_3 := \sum_{\kappa \in K} \sum_{M \in \{A,B,C\}} \eta_M \frac{v_H(\beta_2) v_H(\beta_1) \hat{\mathsf{val}}_M(\kappa)}{(\beta_2 - \hat{\mathsf{row}}_M(\kappa))(\beta_1 - \hat{\mathsf{col}}_M(\kappa))}$
and find $g_3(X)$ and $h_3(X)$ such that
$h_3(X)v_K(X) = a(X) - b(X)(Xg_3(X) + \sigma_3/|K|)$

$\sigma_3 \in \mathbb{F}, g_3 \in \mathbb{F}^{<|K|-1}[X], h_3 \in \mathbb{F}^{<6|K|-6}[X]$ $\longrightarrow$

$\beta_3 \leftarrow \mathbb{F}$

$h_3(\beta_3)v_K(\beta_3) \stackrel{?}{=} a(\beta_3) - b(\beta_3)(\beta_3 g_3(\beta_3) + \sigma_3/|K|)$

The polynomials $a(X), b(X)$ are defined as follows:
$a(X) := \sum_{M \in \{A,B,C\}} \eta_M v_H(\beta_2) v_H(\beta_1) \hat{\mathsf{val}}_M(X) \prod_{N \in \{A,B,C\} \setminus \{M\}} (\beta_2 - \hat{\mathsf{row}}_N(X))(\beta_1 - \hat{\mathsf{col}}_N(X))$
$b(X) := \prod_{M \in \{A,B,C\}} (\beta_2 - \hat{\mathsf{row}}_M(X))(\beta_1 - \hat{\mathsf{col}}_M(X))$

$r(\alpha, \beta_2)\sigma_3 \stackrel{?}{=} h_2(\beta_2)v_H(\beta_2) + \beta_2 g_2(\beta_2) + \sigma_2/|H|$

$s(\beta_1) + r(\alpha, \beta_1)(\sum_M \eta_M \hat{z_M}(\beta_1)) - \sigma_2 \hat{z}(\beta_1) \stackrel{?}{=} h_1(\beta_1)v_H(\beta_1) + \beta_1 g_1(\beta_1) + \sigma_1/|H|$

$\hat{z}_A(\beta_1)\hat{z}_B(\beta_1) - \hat{z}_C(\beta_1) \stackrel{?}{=} h_0(\beta_1)v_H(\beta_1)$

Figure 2.5: AHP for R1CS.

2.6 Polynomial commitment schemes with extractability

We use *polynomial commitment schemes*, a class of commitment schemes specialized to work with univariate polynomials. This notion was introduced by Kate, Zaverucha, and Goldberg [KZG10], who gave an elegant construction using bilinear groups. The security properties in [KZG10], however, do not appear sufficient for standalone use (nor for use in this work). This limitation was recently noted in [MBKM19], which relies on a different construction for which certain properties are proved in the algebraic group model [FKL18]. However, [MBKM19] stops short of formulating a cryptographic primitive that captures the features of the construction.

In this section we propose definitions for polynomial commitment schemes that incorporate the functionality and security that we believe to be a bare minimum for standalone use. (In particular, in Section 2.8 we generically rely on these definitions to build preprocessing arguments with universal SRS.) We also describe a "knowledge" variant of the construction in [KZG10], which we prove secure under knowledge of exponent assumptions. To learn more about the insights motivating our definitions, we refer the reader back to Section 2.2.2.

The rest of this section is organized as follows. In Section 2.6.1 we present the definitions that we propose. In Section 2.6.2 we provide a theorem statement for constructions that realize the definitions, and then sketch these constructions. We formal descriptions of the constructions are in Sections 2.11 to 2.12.

2.6.1 Definition

A polynomial commitment scheme over a field family $\mathcal{F}$ is a tuple of algorithms $\mathsf{PC} = (\mathsf{Setup}, \mathsf{Trim}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Check})$ with the following syntax.

- $\mathsf{PC.Setup}(1^\lambda, D) \to \mathsf{pp}$. On input a security parameter λ (in unary), and a maximum degree bound $D \in \mathbb{N}$, PC.Setup samples public parameters pp. The parameters contain the description of a finite field $\mathbb{F} \in \mathcal{F}$.

- $\mathsf{PC.Trim}^{\mathsf{pp}}(1^\lambda, \boldsymbol{d}) \to (\mathsf{ck}, \mathsf{rk})$. Given oracle access to public parameters pp, and on input a security parameter λ (in unary), and degree bounds $\boldsymbol{d}$, PC.Trim deterministically computes a key pair $(\mathsf{ck}, \mathsf{rk})$ that is specialized to $\boldsymbol{d}$.

- $\mathsf{PC.Commit}(\mathsf{ck}, \boldsymbol{p}, \boldsymbol{d}; \boldsymbol{\omega}) \to \boldsymbol{c}$. On input ck, univariate polynomials $\boldsymbol{p} = [p_i]_{i=1}^n$ over the field $\mathbb{F}$, and degree bounds $\boldsymbol{d} = [d_i]_{i=1}^n$ with $\deg(p_i) \leq d_i \leq D$, PC.Commit outputs commitments $\boldsymbol{c} = [c_i]_{i=1}^n$ to the polynomials $\boldsymbol{p} = [p_i]_{i=1}^n$. The randomness $\boldsymbol{\omega} = [\omega_i]_{i=1}^n$ is used if the commitments $\boldsymbol{c} = [c_i]_{i=1}^n$ are hiding.

- $\mathsf{PC.Open}(\mathsf{ck}, \boldsymbol{p}, \boldsymbol{d}, Q, \xi; \boldsymbol{\omega}) \to \pi$. On input ck, univariate polynomials $\boldsymbol{p} = [p_i]_{i=1}^n$, degree bounds $\boldsymbol{d} = [d_i]_{i=1}^n$, a query set Q consisting of tuples $(i, z) \in [n] \times \mathbb{F}$, and opening challenge ξ, PC.Open outputs an evaluation proof π. The randomness $\boldsymbol{\omega}$ must equal the one previously used in PC.Commit.

- $\mathsf{PC.Check}(\mathsf{rk}, \boldsymbol{c}, \boldsymbol{d}, Q, \boldsymbol{v}, \pi, \xi) \in \{0,1\}$. On input rk, commitments $\boldsymbol{c} = [c_i]_{i=1}^n$, degree bounds $\boldsymbol{d} = [d_i]_{i=1}^n$, query set Q consisting of tuples $(i,z) \in [n] \times \mathbb{F}$, alleged evaluations $\boldsymbol{v} = (v_{(i,z)})_{(i,z)\in Q}$, evaluation proof π, and opening challenge ξ, $\mathsf{PC.Check}$ outputs 1 if π attests that, for every $(i,z) \in Q$, the polynomial p_i committed in c_i has degree at most d_i and evaluates to $v_{(i,z)}$ at z.

A polynomial commitment scheme PC must satisfy the completeness and extractability properties defined below. We also consider two additional properties, efficiency and hiding, also defined below. To simplify notation, we denote by $\deg(\boldsymbol{p})$ the degrees $[\deg(p_i)]_{i=1}^n$ of polynomials $\boldsymbol{p} = [p_i]_{i=1}^n$, and denote by $\boldsymbol{p}(Q)$ the evaluations $(p_i(z))_{(i,z)\in Q}$ of the polynomials $\boldsymbol{p} = [p_i]_{i=1}^n$ at a query set $Q \subseteq [n] \times \mathbb{F}$.

Definition 2.6.1 (Completeness). *For every maximum degree bound $D \in \mathbb{N}$ and efficient adversary $\mathcal{A}$,*

$$\Pr\left[\begin{array}{c} \deg(\boldsymbol{p}) \leq \boldsymbol{d} \leq D \\ \Downarrow \\ \mathsf{PC.Check}(\mathsf{rk}, \boldsymbol{c}, \boldsymbol{d}, Q, \boldsymbol{v}, \pi, \xi) = 1 \end{array} \middle| \begin{array}{r} \mathsf{pp} \leftarrow \mathsf{PC.Setup}(1^\lambda, D) \\ (\boldsymbol{p}, \boldsymbol{d}, Q, \xi, \boldsymbol{\omega}) \leftarrow \mathcal{A}(\mathsf{pp}) \\ (\mathsf{ck}, \mathsf{rk}) \leftarrow \mathsf{PC.Trim}^{\mathsf{pp}}(1^\lambda, \boldsymbol{d}) \\ \boldsymbol{c} \leftarrow \mathsf{PC.Commit}(\mathsf{ck}, \boldsymbol{p}, \boldsymbol{d}; \boldsymbol{\omega}) \\ \boldsymbol{v} \leftarrow \boldsymbol{p}(Q) \\ \pi \leftarrow \mathsf{PC.Open}(\mathsf{ck}, \boldsymbol{p}, \boldsymbol{d}, Q, \xi; \boldsymbol{\omega}) \end{array}\right] = 1 \ .$$

Definition 2.6.2 (Extractability). *For every maximum degree bound $D \in \mathbb{N}$ and efficient adversary $\mathcal{A}$ there exists an efficient extractor $\mathcal{E}$ such that for every round bound $\mathsf{r} \in \mathbb{N}$, efficient public-coin challenger $\mathcal{C}$ (each of its messages is a uniformly random string of prescribed length, or an empty string), efficient query sampler $\mathcal{Q}$, and efficient adversary $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ the probability below is negligibly close to 1 (as a function of λ):*

$$\Pr\left[\begin{array}{c} \mathsf{PC.Check}(\mathsf{rk}, \boldsymbol{c}, \boldsymbol{d}, Q, \boldsymbol{v}, \pi, \xi) = 1 \\ \Downarrow \\ \deg(\boldsymbol{p}) \leq \boldsymbol{d} \leq D \text{ and } \boldsymbol{v} = \boldsymbol{p}(Q) \end{array} \middle| \begin{array}{r} \mathsf{pp} \leftarrow \mathsf{PC.Setup}(1^\lambda, D) \\ \hline \textit{For } i = 1, \ldots, \mathsf{r}\textit{:} \\ \rho_i \leftarrow \mathcal{C}(\mathsf{pp}, i) \\ (\boldsymbol{c}_i, \boldsymbol{d}_i) \leftarrow \mathcal{A}(\mathsf{pp}, [\rho_j]_{j=1}^i) \\ \boldsymbol{p}_i \leftarrow \mathcal{E}(\mathsf{pp}, [\rho_j]_{j=1}^i) \\ \hline Q \leftarrow \mathcal{Q}(\mathsf{pp}, [\rho_j]_{j=1}^{\mathsf{r}}) \\ (\boldsymbol{v}, \mathsf{st}) \leftarrow \mathcal{B}_1(\mathsf{pp}, [\rho_j]_{j=1}^{\mathsf{r}}, Q) \\ \textit{Sample opening challenge } \xi \\ \pi \leftarrow \mathcal{B}_2(\mathsf{st}, \xi) \\ \textit{Set } [c_i]_{i=1}^n := [\boldsymbol{c}_i]_{i=1}^{\mathsf{r}}, [p_i]_{i=1}^n := [\boldsymbol{p}_i]_{i=1}^{\mathsf{r}}, [d_i]_{i=1}^n := [\boldsymbol{d}_i]_{i=1}^{\mathsf{r}} \\ (\mathsf{ck}, \mathsf{rk}) \leftarrow \mathsf{PC.Trim}^{\mathsf{pp}}(1^\lambda, [d_i]_{i=1}^n) \\ \textit{Define the set of queried polynomials } T := \{i \in [n] \mid (i,z) \in Q\} \\ \textit{Set } \boldsymbol{c} := [c_i]_{i\in T}, \boldsymbol{p} := [p_i]_{i\in T}, \boldsymbol{d} := [d_i]_{i\in T} \end{array}\right] .$$

(The above definition captures the case where $\mathcal{A}, \mathcal{Q}, \mathcal{B}$ share the same random string to win the game.)

Definition 2.6.3 (Efficiency). *We say that a polynomial commitment scheme PC is:*

- ***degree-efficient*** *if the time to run* $\mathsf{PC.Commit}$ *and* $\mathsf{PC.Open}$ *is proportional to the maximum degree* $\max(\boldsymbol{d})$ *(as opposed to the maximum supported degree* D*). In particular this implies that* $|\mathsf{ck}| = O_\lambda(\max(\boldsymbol{d}))$.
- ***succinct*** *if the size of commitments, the size of evaluation proofs, and the time to check an opening are all independent of the degree of the committed polynomials. That is,* $|\boldsymbol{c}| = n \cdot \mathrm{poly}(\lambda)$, $|\pi| = |Q| \cdot \mathrm{poly}(\lambda)$, $|\mathsf{rk}| = O_\lambda(n)$, *and* $\mathsf{time}(\mathsf{Check}) = (n + |Q|) \cdot \mathrm{poly}(\lambda)$.

Definition 2.6.4 (Hiding). *There exists a polynomial-time simulator* $\mathcal{S} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open})$ *such that, for every maximum degree bound* $D \in \mathbb{N}$*, and efficient adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$*, the probability that* $b = 1$ *in the following two experiments is identical:*

$\mathsf{Real}(1^\lambda, D, \mathcal{A})$*:*

1. $\mathsf{pp} \leftarrow \mathsf{PC.Setup}(1^\lambda, D)$.
2. *Letting* $\boldsymbol{c}_0 := \bot$*, for* $i = 1, \ldots, \mathsf{r}$*:*
   a) $(\boldsymbol{p}_i, \boldsymbol{d}_i, \mathsf{h}_i) \leftarrow \mathcal{A}_1(\mathsf{pp}, \boldsymbol{c}_0, \boldsymbol{c}_1, \ldots, \boldsymbol{c}_{i-1})$.
   b) $(\mathsf{ck}_i, \mathsf{rk}_i) \leftarrow \mathsf{PC.Trim}^{\mathsf{pp}}(1^\lambda, \boldsymbol{d}_i)$.
   c) *If* $\mathsf{h}_i = 0$*: sample commitment randomness* $\boldsymbol{\omega}_i$.
   d) *If* $\mathsf{h}_i = 1$*: set randomness* $\boldsymbol{\omega}_i$ *to* $\bot$.
   e) $\boldsymbol{c}_i \leftarrow \mathsf{PC.Commit}(\mathsf{ck}_i, \boldsymbol{p}_i, \boldsymbol{d}_i; \boldsymbol{\omega}_i)$.
3. $\boldsymbol{c} := [\boldsymbol{c}_i]_{i=1}^{\mathsf{r}}, \boldsymbol{p} := [\boldsymbol{p}_i]_{i=1}^{\mathsf{r}}, \boldsymbol{d} := [\boldsymbol{d}_i]_{i=1}^{\mathsf{r}}, \boldsymbol{\omega} := [\boldsymbol{\omega}_i]_{i=1}^{\mathsf{r}}$.
4. $(\mathsf{ck}, \mathsf{rk}) \leftarrow \mathsf{PC.Trim}^{\mathsf{pp}}(1^\lambda, \boldsymbol{d})$.
5. $([Q_j]_{j=1}^{\tau}, [\xi_j]_{j=1}^{\tau}, \mathsf{st}) \leftarrow \mathcal{A}_2(\mathsf{pp}, \boldsymbol{c})$.
6. *For* $j \in [\tau]$*:*
   $\pi_j \leftarrow \mathsf{PC.Open}(\mathsf{ck}, \boldsymbol{p}, \boldsymbol{d}, Q_j, \xi_j; \boldsymbol{\omega})$.
7. $b \leftarrow \mathcal{A}_3(\mathsf{st}, [\pi]_{j=1}^{\tau})$.

$\mathsf{Ideal}(1^\lambda, D, \mathcal{A})$*:*

1. $(\mathsf{pp}, \mathsf{trap}) \leftarrow \mathcal{S}.\mathsf{Setup}(1^\lambda, D)$.
2. *Letting* $\boldsymbol{c}_0 := \bot$*, for* $i = 1, \ldots, \mathsf{r}$*:*
   a) $(\boldsymbol{p}_i, \boldsymbol{d}_i, \mathsf{h}_i) \leftarrow \mathcal{A}_1(\mathsf{pp}, \boldsymbol{c}_0, \boldsymbol{c}_1, \ldots, \boldsymbol{c}_{i-1})$.
   b) $(\mathsf{ck}_i, \mathsf{rk}_i) \leftarrow \mathsf{PC.Trim}^{\mathsf{pp}}(1^\lambda, \boldsymbol{d}_i)$.
   c) *If* $\mathsf{h}_i = 0$*: sample randomness* $\boldsymbol{\omega}_i$ *and compute simulated commitments* $\boldsymbol{c}_i \leftarrow \mathcal{S}.\mathsf{Commit}(\mathsf{trap}, \boldsymbol{d}_i; \boldsymbol{\omega}_i)$.
   d) *If* $\mathsf{h}_i = 1$*: set* $\boldsymbol{\omega}_i := \bot$ *and compute (real) commitments* $\boldsymbol{c}_i \leftarrow \mathsf{PC.Commit}(\mathsf{ck}_i, \boldsymbol{p}_i, \boldsymbol{d}_i; \boldsymbol{\omega}_i)$.
3. $\boldsymbol{c} := [\boldsymbol{c}_i]_{i=1}^{\mathsf{r}}, \boldsymbol{p} := [\boldsymbol{p}_i]_{i=1}^{\mathsf{r}}, \boldsymbol{d} := [\boldsymbol{d}_i]_{i=1}^{\mathsf{r}}, \boldsymbol{\omega} := [\boldsymbol{\omega}_i]_{i=1}^{\mathsf{r}}$.
4. $(\mathsf{ck}, \mathsf{rk}) \leftarrow \mathsf{PC.Trim}^{\mathsf{pp}}(1^\lambda, \boldsymbol{d})$.
5. $([Q_j]_{j=1}^{\tau}, [\xi_j]_{j=1}^{\tau}, \mathsf{st}) \leftarrow \mathcal{A}_2(\mathsf{pp}, \boldsymbol{c})$.
6. *Zero out hidden polynomials:* $\boldsymbol{p}' := [\mathsf{h}_i \boldsymbol{p}_i]_{i=1}^{\mathsf{r}}$.
7. *For* $j \in [\tau]$*:*
   $\pi_j \leftarrow \mathcal{S}.\mathsf{Open}(\mathsf{trap}, \boldsymbol{p}', \boldsymbol{p}(Q_j), \boldsymbol{d}, Q_j, \xi_j; \boldsymbol{\omega})$.
8. $b \leftarrow \mathcal{A}_3(\mathsf{st}, [\pi]_{j=1}^{\tau})$.

(We implicitly assume that $\mathcal{A}_1$ *outputs* $\mathrm{poly}(\lambda)$ *polynomials overall and that* $\mathcal{A}_2$ *outputs* $\mathrm{poly}(\lambda)$ *query sets each consisting of* $\mathrm{poly}(\lambda)$ *points, ensuring that* $\mathsf{PC_s.Commit}, \mathsf{PC_s.Open}, \mathcal{S}.\mathsf{Commit}, \mathcal{S}.\mathsf{Open}$ *are efficient.)*

2.6.2 Construction

The theorem below states the properties of our constructions. For simplicity, our construction are restricted to work with respect to "admissible" query samplers.

Definition 2.6.5. *A query sampler* $\mathcal{Q}$ *is* **admissible** *if it outputs query sets such that each polynomial to be evaluated is evaluated at a point sampled uniformly at random from a super-polynomially large subset of the field, and possibly also at other points that can be arbitrarily chosen.*

Theorem 2.6.6. *There exist succinct polynomial commitment schemes that: (a) achieve extractability against admissible query samplers under knowledge assumptions, or in the algebraic group model; (b) achieve hiding; and (c) have an updatable SRS. See Table 2.2 for the efficiency of these schemes under these assumptions.*

We note that the restriction to admissible query samplers is minor because one can transform an arbitrary query sampler $\mathcal{Q}$ into an admissible query sampler $\mathcal{Q}'$ as follows: $\mathcal{Q}'$ invokes $\mathcal{Q}$ to obtain a

query set Q, and then outputs $Q' := Q \cup \{(i, t)\}_{i \in [n]}$, where $t \in \mathbb{F}$ is a random field element and n is the number of polynomials. This transformation yields evaluation proofs that are twice as a large, a minor cost. That said, this transformation is often not even needed because "natural" query samplers are often already admissible, as is the case for those that we consider in this work.

assumption	hiding	communication complexity				time complexity			
		$\|\mathsf{ck}\|$	$\|\mathsf{rk}\|$	$\|[c_i]_{i=1}^n\|$	$\|\pi\|$	Setup	Commit	Open	Check
PKE	no	$2d\ \mathbb{G}_1$	$2\ \mathbb{G}_2$	$4n\ \mathbb{G}_1$	$1\ \mathbb{G}_1$	2 f-MSM(D)	$4n$ v-MSM(d)	1 v-MSM(d)	2 v-MSM$(2n)$ + 4 pairings
dPKE	yes	$4d\ \mathbb{G}_1$	$2\ \mathbb{G}_2$	$4n\ \mathbb{G}_1$	$1\ \mathbb{G}_1$ + $1\ \mathbb{F}_q$	4 f-MSM(D)	$8n$ v-MSM(d)	2 v-MSM(d)	2 v-MSM$(2n)$ + 4 pairings
AGM	no	$d\ \mathbb{G}_1$	$1\ \mathbb{G}_2$	$2n\ \mathbb{G}_1$	$1\ \mathbb{G}_1$	1 f-MSM(D)	$2n$ v-MSM(d)	1 v-MSM(d)	1 v-MSM$(2n)$ + 2 pairings
AGM	yes	$2d\ \mathbb{G}_1$	$1\ \mathbb{G}_2$	$2n\ \mathbb{G}_1$	$1\ \mathbb{G}_1$ + $1\ \mathbb{F}_q$	2 f-MSM(D)	$4n$ v-MSM(d)	2 v-MSM(d)	1 v-MSM$(2n)$ + 2 pairings

Table 2.2: Efficiency of our polynomial commitment schemes. Here f-MSM(m) and v-MSM(m) denote fixed-base and variable-base multi-scalar multiplications (MSM) each of size m, respectively. All MSMs are carried out over $\mathbb{G}_1$. For simplicity, we assume above that the query set evaluates each polynomial at the same point. If there are multiple points in the set, then proof size and time for checking proofs scales linearly with the number of points. Furthermore, we assume above that the n committed polynomials all have degree d.

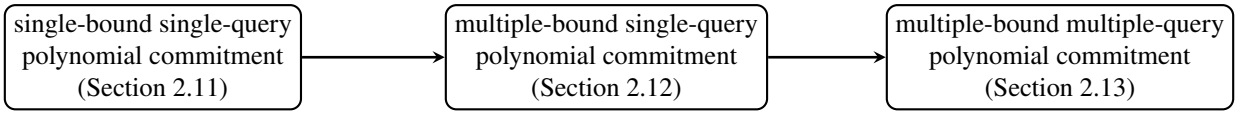

Figure 2.6: Our approach to construct polynomial commitment schemes.

The constructions behind Theorem 2.6.6 are achieved in three steps, as summarized in Fig. 2.6. The rest of this section is organized in three parts sketching these three steps respectively: (1) opening multiple polynomials with the same degree bound at a single point; (2) opening multiple polynomials with multiple degree bounds at a single point; (3) opening multiple polynomials with multiple degree bounds at multiple points. Detailed descriptions, along with security proofs, are provided in the corresponding appendices.

2.6.2.1 Single-bound single-query (see Section 2.11 for details)

We begin by discussing the case of opening multiple polynomials with the same degree bound at a single point. We describe a non-hiding construction based on $\mathsf{PolyCommit}_{\mathrm{DL}}$ from [KZG10] (see Section 2.2.5) and a hiding construction based on $\mathsf{PolyCommit}_{\mathrm{Ped}}$ from [KZG10], using "knowledge commitments" [Gro10] or the algebraic group model [FKL18] to achieve extractability for a single degree bound D chosen at setup.

Extractability with knowledge commitments. While $\mathsf{PolyCommit}_{\mathrm{DL}}$ guarantees correctness of evaluations, it does not ensure extractability: there is no guarantee that a commitment actually "contains" a polynomial. To achieve extraction, we modify the construction in such a way that the PKE assumption [Gro10] forces the sender to demonstrate knowledge of the committed polynomial. In more detail, we extend ck to encode of powers of β with respect to a different generator αG: $\mathsf{ck} := \{(G, \beta G, \ldots, \beta^D G), (G, \alpha\beta G, \ldots, \alpha\beta^D G)\} \in \mathbb{G}_1^{2(D+1)}$. (Note that this modification does not affect the updatability of the SRS.) To commit to a polynomial p of degree at most D, the sender now provides a "knowledge commitment": $c := (U, V) := (p(\beta)G, \alpha p(\beta)G)$. Proving correctness of evaluations proceeds unchanged, while verification additionally requires checking extractability of the commitment by checking the pairing equation $e(U, \alpha H) = e(V, H)$.

Extractability in the AGM. Knowledge commitments require, unfortunately, two group elements instead of one. Alternatively, we could keep each commitment as one group element, by relying on the algebraic group model (AGM) [FKL18]. Informally, whenever an adversary in the AGM outputs a group element G_n, it is required to additionally output scalar coefficients $a_1, \ldots, a_{n-1}$ which "explain" G_n as a linear combination of any group elements $G_1, \ldots, G_{n-1}$ that it has seen previously. In our setting, this means that whenever the adversarial sender outputs a group element c representing a commitment, it must additionally output scalar coefficients that explain c in terms of the group elements in ck. An extractor can use these coefficients to reconstruct the underlying polynomial, thus achieving extractability.

Efficiently opening multiple polynomials at the same point. To enable the sender to simultaneously commit to multiple polynomials $[p_i]_{i=1}^n$ of degree at most D and then open these at the same point z, we rely on the fact that the commitments for both variants above are *additively homomorphic*. That is, if commitments $[c_i]_{i=1}^n$ commit to $[p_i]_{i=1}^n$, then $\sum_{i=1}^n c_i$ commits to $\sum_{i=1}^n p_i$ (where $c_1 + c_2$ is defined as $(U_1 + U_2, V_1 + V_2)$).

We take advantage of this by simultaneously verifying the evaluations of each polynomial $p_i \in [p_i]_{i=1}^n$ as follows. Before generating a proof of evaluation for $[p_i]_{i=1}^n$, the sender requests from the receiver a random field element ξ. The sender then uses this to take a random linear combination of the polynomials: $p := \sum_{i=1}^n \xi^i p_i$, and generates a single evaluation proof π for this derived polynomial p.

To verify π, the receiver uses the additive homomorphism of the input commitments to derive the linear combination $c = \sum_{i=1}^n \xi^i c_i$ induced by ξ. It does the same with the claimed evaluations, thus deriving the evaluation $v = \sum_{i=1}^n \xi^i v_i$. Finally, it checks that the pairing equations are satisfied for c, π, and v.

This works because if the sender is honest, then c is a commitment to $p := \sum_{i=1}^n \xi^i p_i$, and π is a proof of evaluation of p at z. On the other hand, if the sender is dishonest, then with high probability over the choice of ξ, c is not a commitment to p, and the pairing equations would fail.

Hiding. To additionally achieve hiding, we follow the above blueprint, replacing $\mathsf{PolyCommit}_{\mathrm{DL}}$ with the hiding scheme $\mathsf{PolyCommit}_{\mathrm{Ped}}$. Extraction now follows from an assumption related to PKE called dPKE (see Section 2.11.2.3 for details). Our constructions in Section 2.11 in fact use both variants to provide optional hiding on a per-polynomial basis. Further, the near-identical form of the

commitment variants makes it possible to open a combination of hiding and non-hiding polynomials at the same point.

2.6.2.2 Multiple-bound single-query (see Section 2.12 for details)

Thus far, we have focused on commitment schemes for polynomials of degree D where the cost of committing and providing evaluation proofs grows as $\Omega(D)$. However, when working with polynomials of degree $d < D$, we would like to pay a cost that instead grows as $O(d)$. Furthermore, the foregoing schemes only guarantee that committed polynomials have degree at most D, whereas in many cases it is desirable to enforce more specific degree bounds. Below we show how to adapt the foregoing construction to achieve these desirable properties.

To achieve extractability with respect to a different degree bound d_i for each polynomial p_i, we require the sender to commit not only to each p_i, but also to "shifted polynomials" $p'_i(X) := X^{D-d_i}p_i(X)$. During $\mathsf{PC.Open}$, one could then produce an evaluation proofs that attests that if p_i evaluates to v_i at z then p'_i evaluates to $z^{D-d_i}v_i$ at z.

The receiver checks that the commitment for each p'_i corresponds to an evaluation $z^{D-d_i}v_i$ so that, if z is sampled from a super-polynomial subset of $\mathbb{F}_q$, the probability that $\deg(p_i) \neq d_i$ is negligible. This trick is similar to the one used in [BS08; BCRSVW19] to enforce derive low-degree tests for specific degree bounds.

However, while sound, this approach is inefficient in our setting: the witness polynomial for p'_i has $\Omega(D)$ non-zero coefficients (instead of $O(d_i)$), and so constructing an evaluation proof for it requires $\Omega(D)$ scalar multiplications (instead of $O(d_i)$). To work around this, we instead produce a proof that the related polynomial $p^\star_i(X) := p'_i(X) - p_i(z)X^{D-d_i}$ evaluates to 0 at z. As we show in Lemma 2.12.2, the witness polynomial for this claim has $O(d_i)$ non-zero coefficients, and so constructing the evaluation proof can be done in $O(d_i)$ scalar multiplications. Completeness is preserved because the receiver can check the correct evaluation of $p^\star_i$ by subtracting $p_i(z)(\beta^{D-d_i}\mathbb{G})$ from the commitment to the shifted polynomial p'_i, thereby obtaining a commitment to $p^\star_i$, while security is preserved because $p'_i(z) = z^{D-d_i}v_i \iff p^\star_i(z) = 0$.

Note that to commit to the shifted polynomial p'_i, the committer must obtain $\{\beta^{D-d_i}\mathbb{G}, \ldots, \beta^D\mathbb{G}\}$ from ck, while to adjust the shifted commitment, the receiver must obtain $\beta^{D-d_i}\mathbb{G}$ from rk. Thus $\mathsf{PC.Trim}$ must produce $(\mathsf{ck}, \mathsf{rk})$ containing these group elements.

2.6.2.3 Multiple-bound multiple-query (see Section 2.13 for details)

Assume that we have *any* construction that achieves extractability with respect to individual degree bounds, and evaluation of multiple polynomials $\boldsymbol{p} = [p_i]_{i=1}^n$ at the same point z.

We extend this construction to support query sets Q consisting of multiple evaluation points (as required in Section 2.6.1). If there are k distinct points $[z_i]_{i=1}^k$ in the query set Q, the sender partitions the polynomials $\boldsymbol{p}$ into different (possibly overlapping) groups $[\boldsymbol{p}_i]_{i=1}^k$ such that every polynomial in $\boldsymbol{p}_i$ is to be evaluated at the same point z_i. It then runs $\mathsf{PC.Open}$ on each $\boldsymbol{p}_i$, and outputs the resulting list of k evaluation proofs.

We note that [KZG10] describe how one can enable the sender to produce a *single* evaluation proof attesting to the correct evaluation of the same polynomial at multiple points. While we could use this to enable batch evaluation of $\boldsymbol{p}$ at multiple points, we avoid doing so for efficiency reasons in our setting.

2.7 Preprocessing arguments with universal SRS

An *argument system* [BCC88] is an interactive proof where the soundness property is only required to hold against all efficient adversaries, as opposed to all (possibly computationally unbounded) adversaries. In this work we consider argument systems for indexed relations (see Section 2.3.1) that have the following features.

- Security is proved, under cryptographic assumptions, in a model where all parties have access to a "long" structured reference string (SRS) that is *universal*. (In fact, the SRS in our constructions will also be *updatable* [GKMMM18] but for simplicity we do not formally discuss this property; see Remark 2.7.1.)
- Anyone can publicly *preprocess* a given index (e.g., a circuit) in an offline phase, in order to avoid incurring costs related to the index in (any number of) subsequent online phases that check different instances.

We refer to argument systems with the above properties as **preprocessing arguments with universal SRS**. All interactive constructions in this work are public-coin zero-knowledge succinct arguments of knowledge so that, via the Fiat–Shamir transformation [FS86], we obtain their non-interactive analogues: **preprocessing zkSNARKs with universal SRS**. See Section 2.9 for an efficient construction of such a zkSNARK.

A preprocessing argument with universal SRS is a tuple of four algorithms $\mathsf{ARG} = (\mathcal{G}, \mathcal{I}, \mathcal{P}, \mathcal{V})$. The probabilistic polynomial-time generator $\mathcal{G}$, given a size bound $\mathsf{N} \in \mathbb{N}$, samples an SRS srs that supports indices of size up to N. The indexer $\mathcal{I}$ is a *deterministic* polynomial-time algorithm that, given oracle access to srs and an index $\mathbb{i}$ of size at most N, outputs an index proving key ipk used by the prover $\mathcal{P}$ in place of $\mathbb{i}$ and an index verification key ivk used by the verifier $\mathcal{V}$ in place of $\mathbb{i}$; the verifier $\mathcal{V}$ will be able to use ivk for significant efficiency gains compared to just using $\mathbb{i}$ directly. The prover $\mathcal{P}$ and verifier $\mathcal{V}$ are probabilistic polynomial-time interactive algorithms.

Formally, $\mathsf{ARG} = (\mathcal{G}, \mathcal{I}, \mathcal{P}, \mathcal{V})$ is a preprocessing argument with universal SRS for an indexed relation $\mathcal{R}$ if the following properties hold.

- **Completeness.** For all size bounds $\mathsf{N} \in \mathbb{N}$ and efficient $\mathcal{A}$,

$$\Pr\left[\begin{array}{c} (\mathbb{i}, \mathbb{x}, \mathbb{w}) \notin \mathcal{R}_{\mathsf{N}} \\ \vee \\ \langle \mathcal{P}(\mathsf{ipk}, \mathbb{x}, \mathbb{w}), \mathcal{V}(\mathsf{ivk}, \mathbb{x}) \rangle = 1 \end{array} \middle| \begin{array}{r} \mathsf{srs} \leftarrow \mathcal{G}(1^\lambda, \mathsf{N}) \\ (\mathbb{i}, \mathbb{x}, \mathbb{w}) \leftarrow \mathcal{A}(\mathsf{srs}) \\ (\mathsf{ipk}, \mathsf{ivk}) \leftarrow \mathcal{I}^{\mathsf{srs}}(\mathbb{i}) \end{array}\right] = 1 \ .$$

- **Soundness.** For all size bounds $\mathsf{N} \in \mathbb{N}$ and efficient $\tilde{\mathcal{P}} = (\tilde{\mathcal{P}}_1, \tilde{\mathcal{P}}_2)$,

$$\Pr\left[\begin{array}{c} (\mathbb{i}, \mathbb{x}) \notin \mathcal{L}(\mathcal{R}_{\mathsf{N}}) \\ \wedge \\ \langle \tilde{\mathcal{P}}_2(\mathsf{st}), \mathcal{V}(\mathsf{ivk}, \mathbb{x}) \rangle = 1 \end{array} \middle| \begin{array}{r} \mathsf{srs} \leftarrow \mathcal{G}(1^\lambda, \mathsf{N}) \\ (\mathbb{i}, \mathbb{x}, \mathsf{st}) \leftarrow \tilde{\mathcal{P}}_1(\mathsf{srs}) \\ (\mathsf{ipk}, \mathsf{ivk}) \leftarrow \mathcal{I}^{\mathsf{srs}}(\mathbb{i}) \end{array}\right] = \mathrm{negl}(\lambda) \ .$$

Our definition of completeness allows $(\mathbb{i}, \mathbb{x}, \mathbb{w})$ to depend on srs, while our formulation of soundness allows $(\mathbb{i}, \mathbb{x})$ to depend on srs.

All constructions in this work achieve the stronger property of *knowledge soundness*, and optionally also the property of (perfect) *zero knowledge*. We define these properties below.

Knowledge soundness. We say that $\mathsf{ARG} = (\mathcal{G}, \mathcal{I}, \mathcal{P}, \mathcal{V})$ has knowledge soundness if for every size bound $\mathsf{N} \in \mathbb{N}$ and efficient adversary $\tilde{\mathcal{P}} = (\tilde{\mathcal{P}}_1, \tilde{\mathcal{P}}_2)$ there exists an efficient extractor $\mathcal{E}$ such that

$$\Pr\left[\begin{array}{c|r} (\mathbb{i}, \mathbb{x}, \mathbb{w}) \notin \mathcal{R}_{\mathsf{N}} & \mathsf{srs} \leftarrow \mathcal{G}(1^\lambda, \mathsf{N}) \\ \wedge & (\mathbb{i}, \mathbb{x}, \mathsf{st}) \leftarrow \tilde{\mathcal{P}}_1(\mathsf{srs}) \\ \langle \tilde{\mathcal{P}}_2(\mathsf{st}), \mathcal{V}(\mathsf{ivk}, \mathbb{x}) \rangle = 1 & \mathbb{w} \leftarrow \mathcal{E}(\mathsf{srs}) \\ & (\mathsf{ipk}, \mathsf{ivk}) \leftarrow \mathcal{I}^{\mathsf{srs}}(\mathbb{i}) \end{array}\right] = \mathrm{negl}(\lambda) \ .$$

Zero knowledge. We say that $\mathsf{ARG} = (\mathcal{G}, \mathcal{I}, \mathcal{P}, \mathcal{V})$ has (perfect) zero knowledge if there exists an efficient simulator $\mathcal{S} = (\mathsf{Setup}, \mathsf{Prove})$ such that for every efficient adversary $\tilde{\mathcal{V}} = (\tilde{\mathcal{V}}_1, \tilde{\mathcal{V}}_2)$ it holds that

$$\begin{aligned} &\Pr\left[\begin{array}{c|r} (\mathbb{i}, \mathbb{x}, \mathbb{w}) \in \mathcal{R}_{\mathsf{N}} & \mathsf{srs} \leftarrow \mathcal{G}(1^\lambda, \mathsf{N}) \\ \wedge & (\mathbb{i}, \mathbb{x}, \mathbb{w}, \mathsf{st}) \leftarrow \tilde{\mathcal{V}}_1(\mathsf{srs}) \\ \langle \mathcal{P}(\mathsf{ipk}, \mathbb{x}, \mathbb{w}), \tilde{\mathcal{V}}_2(\mathsf{st}) \rangle = 1 & (\mathsf{ipk}, \mathsf{ivk}) \leftarrow \mathcal{I}^{\mathsf{srs}}(\mathbb{i}) \end{array}\right] \\ &= \Pr\left[\begin{array}{c|r} (\mathbb{i}, \mathbb{x}, \mathbb{w}) \in \mathcal{R}_{\mathsf{N}} & (\mathsf{srs}, \mathsf{trap}) \leftarrow \mathcal{S}.\mathsf{Setup}(1^\lambda, \mathsf{N}) \\ \wedge & (\mathbb{i}, \mathbb{x}, \mathbb{w}, \mathsf{st}) \leftarrow \tilde{\mathcal{V}}_1(\mathsf{srs}) \\ \langle \mathcal{S}.\mathsf{Prove}(\mathsf{trap}, \mathbb{i}, \mathbb{x}), \tilde{\mathcal{V}}_2(\mathsf{st}) \rangle = 1 & \end{array}\right] . \end{aligned}$$

Efficiency. We say that $\mathsf{ARG} = (\mathcal{G}, \mathcal{I}, \mathcal{P}, \mathcal{V})$ is:

- *index efficient* if the running time of the prover $\mathcal{P}(\mathsf{ipk}, \mathbb{x}, \mathbb{w})$ is $\mathrm{poly}_\lambda(|\mathbb{i}|)$, i.e., it does not depend on the size of the universal structured reference string srs;
- *proof succinct* if the size of the communication transcript between the prover $\mathcal{P}(\mathsf{ipk}, \mathbb{x}, \mathbb{w})$ and verifier $\mathcal{V}(\mathsf{ivk}, \mathbb{x})$ is $\mathrm{poly}(\lambda)$, i.e., the size is bounded by a universal polynomial in the security parameter λ;
- *verifier succinct* if the running time of $\mathcal{V}(\mathsf{ivk}, \mathbb{x})$ is $\mathrm{poly}(\lambda + |\mathbb{x}|)$, i.e., the time is bounded by a universal polynomial in the security parameter λ and the size of the instance $\mathbb{x}$ and *does not* depend on the size of the index $\mathbb{i}$ that led to ivk.

Index efficiency implies that ipk output by $\mathcal{I}$ is of size $\mathrm{poly}_\lambda(|\mathbb{i}|)$, while verifier succinctness implies that ivk output by $\mathcal{I}$ is of size $\mathrm{poly}(\lambda)$. All constructions in this work are index efficient, proof succinct, and verifier succinct.

Public coins. We say that $\mathsf{ARG} = (\mathcal{G}, \mathcal{I}, \mathcal{P}, \mathcal{V})$ is *public-coin* if every message output by the verifier $\mathcal{V}$ is a uniform random string of some prescribed length. All constructions in this work are public-coin, and have a (small) constant number of rounds; in particular, they can be "squashed" to non-interactive arguments that are publicly verifiable by additionally using random oracles via the Fiat–Shamir transformation [FS86]. Hence, due to their succinctness, our constructions directly lead to preprocessing zkSNARKs with universal SRS.

Remark 2.7.1 (updatable SRS)**.** An SRS is *updatable* [GKMMM18] if there exists an update algorithm that can be run at any time by anyone to update the SRS, with the guarantee that security holds as long as there is at least one honest updater since the beginning of time. This property significantly simplifies cryptographic ceremonies to sample the SRS. All preprocessing arguments that we construct in this work have updatable SRS because they only contain "monomial terms", and thus fall within the framework of [GKMMM18].

Remark 2.7.2 (auxiliary inputs)**.** The definition of knowledge soundness above does not consider auxiliary inputs, for simplicity. One could consider a stronger definition, where the adversary and extractor additionally receive an auxiliary input $\mathbb{z}$ sampled from a *fixed* distribution $\mathcal{Z}(1^\lambda)$, or even sampled from *any* distribution $\mathcal{Z}(1^\lambda)$ that belongs to a given class. Such stronger definitions are useful when using argument systems as subroutines within other protocols. When relying on auxiliary inputs, however, one must be careful to ensure that they come from "benign" distributions, or else extraction is impossible, as discussed in [BP15; BCPR16]. We stress that all of our constructions of argument systems directly extend to hold with respect to an auxiliary-input distribution $\mathcal{Z}(1^\lambda)$ under the assumption that the relevant underlying knowledge assumptions are extended to hold with respect to the auxiliary-input distribution $\mathcal{Z}(1^\lambda)$ concatenated with some randomness. (In other words, our security reduction adds to the auxiliary input some random strings.)

2.8 From AHPs to preprocessing arguments with universal SRS

The following theorems capture key properties of our compiler.

Theorem 2.8.1. *Let $\mathcal{F}$ be a field family and let $\mathcal{R}$ be an indexed relation. Consider the following components:*
– $\mathsf{AHP} = (\mathsf{k}, \mathsf{s}, \mathsf{d}, \mathbf{I}, \mathbf{P}, \mathbf{V})$ *is an AHP over $\mathcal{F}$ for $\mathcal{R}$ with negligible soundness error (see Section 2.4);*
– $\mathsf{PC} = (\mathsf{Setup}, \mathsf{Trim}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Check})$ *is a polynomial commitment scheme over $\mathcal{F}$ (see Section 2.6).*
Then $\mathsf{ARG} = (\mathcal{G}, \mathcal{I}, \mathcal{P}, \mathcal{V})$ described in Section 2.8.1 is a preprocessing argument with universal SRS for $\mathcal{R}$ (see Section 2.7). Moreover, if q is the query complexity of AHP*,* ARG *has the following efficiency:*
- **round complexity** *is* $\mathsf{k} + 2$*;*
- **communication complexity** *is $O_\lambda(q)$ bits if* PC *is additionally succinct (see Definition 2.6.3);*
- **indexer time** *is the sum of the indexer time in* AHP *and the time to commit to* $\mathsf{s}(0)$ *polynomials in* PC*;*
- **prover time** *is the sum of the prover time in* AHP*, the time to commit to* $\sum_{i=1}^{\mathsf{k}} \mathsf{s}(i)$ *polynomials in* PC*, the time to produce evaluations that answer the q queries along with a batch evaluation proof for them in* PC*;*
- **verifier time** *is the sum of the verifier time in* AHP *and the time to batch verify q evaluations in* PC*.*

Remark 2.8.2 (updatable SRS)**.** If the SRS for PC is updatable then so is the SRS for ARG. All constructions of polynomial commitments in this work satisfy this property, including the one used in Section 2.9.

The construction underlying the above theorem preserves knowledge soundness and, if the polynomial commitment scheme is also hiding, preserves zero knowledge.

Theorem 2.8.3. *In Theorem 2.8.1, if* AHP *has a negligible knowledge soundness error, then* ARG *has knowledge soundness.*

Theorem 2.8.4. *In Theorem 2.8.1, if* PC *is hiding and if* AHP *is zero knowledge with query bound q (the query complexity of* AHP*) and some polynomial-time query checker $\mathbf{C}$, then* ARG *is (perfect) zero knowledge.*

Remark 2.8.5 (the multivariate case)**.** In this work we give definitions for algebraic holographic proofs and polynomial commitment schemes that are restricted to the case of univariate polynomials, because the constructions that we consider are univariate. Theorems 2.8.1, 2.8.3 and 2.8.4, however, directly extend to the multivariate case when considering an AHP in the general case of multivariate polynomials and a polynomial commitment scheme for multivariate polynomials. This provides a proof of security for several prior works that considered constructions that are special cases of this paradigm but did not prove security (because the polynomial commitment schemes were only assumed to satisfy evaluation binding as discussed in Section 2.1.2).

2.8.1 Construction

We describe the construction behind Theorem 2.8.1, and then discuss its efficiency features.

Generator $\mathcal{G}$. The generator $\mathcal{G}$, on input a security parameter $\lambda \in \mathbb{N}$ and size bound $\mathsf{N} \in \mathbb{N}$, uses N to compute a maximum degree bound $D \in \mathbb{N}$, samples public parameters $\mathsf{pp} \leftarrow \mathsf{PC.Setup}(1^\lambda, D)$ for the polynomial commitment scheme PC, and then outputs $\mathsf{srs} := \mathsf{pp}$. The integer D is computed to be the maximum degree bound in AHP for indices of size N. In other words,

$$D := \max \left\{ \mathsf{d}(\mathsf{N}, i, j) \,\middle|\, i \in \{0, 1, \ldots, \mathsf{k}(\mathsf{N})\}\,,\ j \in \{1, \ldots, \mathsf{s}(i)\} \right\}\ . \tag{2.9}$$

Indexer $\mathcal{I}$. The indexer $\mathcal{I}$ upon input $\mathbb{i}$ and given oracle access to srs, deduces the field $\mathbb{F} \in \mathcal{F}$ contained in $\mathsf{srs} = \mathsf{pp}$, runs the AHP indexer $\mathbf{I}$ on $(\mathbb{F}, \mathbb{i})$ to obtain $\mathsf{s}(0)$ polynomials $p_{0,1}, \ldots, p_{0,\mathsf{s}(0)} \in \mathbb{F}[X]$ of degrees at most $\mathsf{d}(|\mathbb{i}|, 0, 1), \ldots, \mathsf{d}(|\mathbb{i}|, 0, \mathsf{s}(0))$, computes the degree bounds for the index and prover polynomials, invokes $\mathsf{PC.Trim}$ on these bounds to compute $(\mathsf{ck}, \mathsf{rk})$ that are specialized for these degree bounds, and computes commitments to all of the index polynomials.

Namely, $\mathcal{I}$ calculates the bounds $\boldsymbol{d} := \{\mathsf{d}(|\mathbb{i}|, i, \mathsf{s}(i))\}_{i=0}^{\mathsf{k}(|\mathbb{i}|)}$, invokes $(\mathsf{ck}, \mathsf{rk}) := \mathsf{PC.Trim}^{\mathsf{srs}}(\boldsymbol{d})$, and then computes $[c_{0,j}]_{j=1}^{\mathsf{s}(0)} := \mathsf{PC.Commit}(\mathsf{ck}, [p_{0,j}]_{j=1}^{\mathsf{s}(0)}, [\mathsf{d}(|\mathbb{i}|, 0, j)]_{j=1}^{\mathsf{s}(0)}; [\omega_{0,j}]_{j=1}^{\mathsf{s}(0)})$ for "empty randomness" $[\omega_{0,j}]_{j=1}^{\mathsf{s}(0)} := \bot$. The indexer $\mathcal{I}$ outputs $\mathsf{ipk} := (\mathsf{ck}, \mathbb{i}, [p_{0,j}]_{j=1}^{\mathsf{s}(0)}, [c_{0,j}]_{j=1}^{\mathsf{s}(0)})$ and $\mathsf{ivk} := (\mathsf{rk}, [c_{0,j}]_{j=1}^{\mathsf{s}(0)})$. (Note that $[c_{0,j}]_{j=1}^{\mathsf{s}(0)}$ are commitments to non-secret information, and so no randomness is used in producing them. In particular, $\mathcal{I}$ is a deterministic polynomial-time algorithm, as required. Also see Remark 2.8.6 below for additional considerations.)

Prover $\mathcal{P}$ and verifier $\mathcal{V}$. The prover $\mathcal{P}$ receives $(\mathsf{ipk}, \mathbb{x}, \mathbb{w})$ and the verifier $\mathcal{V}$ receives $(\mathsf{ivk}, \mathbb{x})$, where $(\mathsf{ipk}, \mathsf{ivk})$ is the index key pair output by $\mathcal{I}^{\mathsf{srs}}(\mathbb{i})$, and $(\mathbb{i}, \mathbb{x}, \mathbb{w})$ is in the indexed relation $\mathcal{R}$. By construction of $\mathcal{I}$, ipk contains a trimmed committer key ck and ivk contains a trimmed receiver key rk for the polynomial commitment scheme PC. Let $\mathbb{F} \in \mathcal{F}$ be the field described by $(\mathsf{ck}, \mathsf{rk})$ (each of ck and rk individually contain a description of $\mathbb{F}$), and let $\mathsf{k} := \mathsf{k}(|\mathbb{i}|)$ be the number of rounds in AHP. For $i \in \{1, \ldots, \mathsf{k}\}$, $\mathcal{P}$ and $\mathcal{V}$ simulate the i-th round of the interaction between the AHP prover $\mathbf{P}(\mathbb{F}, \mathbb{i}, \mathbb{x}, \mathbb{w})$ and the AHP verifier $\mathbf{V}(\mathbb{F}, \mathbb{x})$.

1. $\mathcal{V}$ receives $\rho_i \in \mathbb{F}^*$ from $\mathbf{V}$, and forwards it to $\mathcal{P}$.
2. $\mathcal{P}$ forwards ρ_i to $\mathbf{P}$, which replies with polynomials $p_{i,1}, \ldots, p_{i,\mathsf{s}(i)} \in \mathbb{F}[X]$ with $\deg(p_{i,j}) \leq \mathsf{d}(|\mathbb{i}|, i, j)$.
3. $\mathcal{P}$ samples commitment randomness $[\omega_{i,j}]_{j=1}^{\mathsf{s}(i)}$ and sends to $\mathcal{V}$ the polynomial commitments below

$$[c_{i,j}]_{j=1}^{\mathsf{s}(i)} := \mathsf{PC.Commit}(\mathsf{ck}, [p_{i,j}]_{j=1}^{\mathsf{s}(i)}, [\mathsf{d}(|\mathbb{i}|, i, j)]_{j=1}^{\mathsf{s}(i)}; [\omega_{i,j}]_{j=1}^{\mathsf{s}(i)})\ .$$

4. $\mathcal{V}$ notifies $\mathbf{V}$ that the i-th round has finished.

The prover $\mathcal{P}$ and verifier $\mathcal{V}$ are done simulating the interactive phase of AHP, and in the remaining two rounds simulate the (non-adaptive) query phase of AHP. Below we use $\boldsymbol{c}$ to denote the commitments $[[c_{i,j}]_{j=1}^{\mathsf{s}(i)}]_{i=0}^{\mathsf{k}}$, $\boldsymbol{p}$ to denote the polynomials $[[p_{i,j}]_{j=1}^{\mathsf{s}(i)}]_{i=0}^{\mathsf{k}}$, $\boldsymbol{d}$ to denote the degree bounds

$[[\mathsf{d}(|\mathbb{i}|, i, j)]_{j=1}^{\mathsf{s}(i)}]_{i=0}^{\mathsf{k}}$, and $\boldsymbol{\omega}$ to denote the randomness $[[\omega_{i,j}]_{j=1}^{\mathsf{s}(i)}]_{i=0}^{\mathsf{k}}$ with $[\omega_{0,j}]_{j=1}^{\mathsf{s}(0)} := \bot$. Note that these three vectors include the commitments, polynomials, degrees, and randomness of the "0-th round".

- $\mathcal{V}$ sends a message $\rho_{\mathsf{k}+1} \in \mathbb{F}^*$ that represents randomness for the query phase of $\mathbf{V}(\mathbb{F}, \mathbb{x})$ to $\mathcal{P}$.
- $\mathcal{P}$ uses the query algorithm of $\mathbf{V}$ to compute the query set $Q := \mathbf{Q}_{\mathbf{V}}(\mathbb{F}, \mathbb{x}; \rho_1, \ldots, \rho_{\mathsf{k}}, \rho_{\mathsf{k}+1})$.
- $\mathcal{P}$ replies with answers $\boldsymbol{v} := \boldsymbol{p}(Q)$.
- $\mathcal{V}$ samples and sends an opening challenge $\xi \in \mathbb{F}$ to $\mathcal{P}$.
- $\mathcal{P}$ replies with an evaluation proof to demonstrate correctness of all claimed evaluations:

$$\pi := \mathsf{PC.Open}(\mathsf{ck}, \boldsymbol{p}, \boldsymbol{d}, Q, \xi; \boldsymbol{\omega}) \ .$$

- $\mathcal{V}$ accepts if and only if the following conditions hold:
  - the decision algorithm of $\mathbf{V}$ accepts the answers, i.e., $\mathbf{D}_{\mathbf{V}}(\mathbb{F}, \mathbb{x}, \boldsymbol{v}; \rho_1, \ldots, \rho_{\mathsf{k}}, \rho_{\mathsf{k}+1}) = 1$;
  - the alleged answers pass the test, i.e., $\mathsf{PC.Check}(\mathsf{rk}, \boldsymbol{c}, \boldsymbol{d}, Q, \boldsymbol{v}, \pi, \xi) = 1$.

Completeness of the preprocessing argument ARG follows in a straightforward way from completeness of the AHP AHP and completeness of the polynomial commitment scheme PC.

We now discuss the efficiency features of the construction above.

- *Round complexity.* The first k rounds simulate the interactive phase of AHP, with polynomials sent as commitments; one round is to answer the desired queries; and one round is to certify the queries' answers.
- *Communication complexity.* The argument prover $\mathcal{P}$ sends $\sum_{i=1}^{\mathsf{k}} \mathsf{s}(i)$ commitments, q field elements representing query answers, and an evaluation proof that certifies the q answers. The argument verifier $\mathcal{V}$ sends $|\rho_1| + \cdots + |\rho_{\mathsf{k}}| + |\rho_{\mathsf{k}+1}| + 1$ field elements. In Theorem 2.8.1 we state that the communication complexity is $O_\lambda(q)$ because typically it holds that $\sum_{i=0}^{\mathsf{k}} \mathsf{s}(i) \leq q$ (each polynomial is queried at least once) and $|\rho_1| + \cdots + |\rho_{\mathsf{k}}| + |\rho_{\mathsf{k}+1}|$ is a small constant (each verifier message is a few field elements).
- *Indexer time.* The time complexity of $\mathcal{I}$ equals the time complexity of the AHP indexer $\mathbf{I}$ plus the time to trim the PC public parameters pp, and then to commit to the $\mathsf{s}(0)$ polynomials output by $\mathbf{I}$.
- *Prover time.* The time complexity of $\mathcal{P}$ equals the time complexity of the AHP prover $\mathbf{P}$ plus the time to commit to the $\sum_{i=1}^{\mathsf{k}} \mathsf{s}(i)$ polynomials output by $\mathbf{P}$, evaluate $\sum_{i=0}^{\mathsf{k}} \mathsf{s}(i)$ polynomials at the query set Q, and produce an evaluation proof that certifies the correctness of these evaluations.
- *Verifier time.* The time complexity of $\mathcal{V}$ equals the time complexity of the AHP verifier $\mathbf{V}$ plus the time to verify the batch evaluation proof for the q evaluations that provide answers to the query set Q.

Remark 2.8.6 (commitments to index polynomials)**.** The construction described above uses the same polynomial commitment scheme PC for committing to polynomials output by the AHP indexer and to polynomials output by the AHP prover. This simplifies exposition, and allows for a single evaluation proof to certify all query answers. For security, however, it would suffice (even for Theorem 2.8.4) to commit to index polynomials via a commitment scheme that merely satisfies "evaluation binding" (Definition 2.11.8), which is strictly weaker than the notion of extractability

that we use for the other commitments. This is because the commitments in the index verification key are honestly produced in the preprocessing phase. Moreover, for Theorem 2.8.3 to hold we do *not* need the commitments to index polynomials to be hiding.

2.8.2 Proof of Theorem 2.8.1

Suppose that $\tilde{\mathcal{P}} = (\tilde{\mathcal{P}}_1, \tilde{\mathcal{P}}_2)$ is an efficient adversarial prover for ARG that wins with probability at least ϵ, that is,

$$\Pr\left[\begin{array}{c|c} (\mathbb{i},\mathbb{x}) \notin \mathcal{L}(\mathcal{R}_{\mathsf{N}}) & \mathsf{srs} \leftarrow \mathcal{G}(1^\lambda, \mathsf{N}) \\ \wedge & (\mathbb{i},\mathbb{x},\mathsf{st}) \leftarrow \tilde{\mathcal{P}}_1(\mathsf{srs}) \\ \langle \tilde{\mathcal{P}}_2(\mathsf{st}), \mathcal{V}(\mathsf{ivk},\mathbb{x})\rangle = 1 & (\mathsf{ipk},\mathsf{ivk}) \leftarrow \mathcal{I}^{\mathsf{srs}}(\mathbb{i}) \end{array}\right] \geq \epsilon(\lambda) \ .$$

We assume without loss of generality that st output by $\tilde{\mathcal{P}}_1$ contains the public parameters $\mathsf{srs} = \mathsf{pp}$. Also note that $\tilde{\mathcal{P}}_2$ can be represented via its $\mathsf{k}+2$ next-message functions:

$$\tilde{\mathcal{P}}_2(\mathsf{st};\rho_1) \ , \ \tilde{\mathcal{P}}_2(\mathsf{st};\rho_1,\rho_2) \ , \ \ldots \ , \ \tilde{\mathcal{P}}_2(\mathsf{st};\rho_1,\ldots,\rho_{\mathsf{k}}) \ , \ \tilde{\mathcal{P}}_2(\mathsf{st};\rho_1,\ldots,\rho_{\mathsf{k}},Q) \ , \ \tilde{\mathcal{P}}_2(\mathsf{st};\rho_1,\ldots,\rho_{\mathsf{k}},Q,\xi) \ .$$

We describe how to construct a prover $\tilde{\mathbf{P}}$, which is admissible for AHP, and an efficient adversary $\mathcal{A}_{\mathsf{PC}}$ against the extractability of PC such that

$$\Pr\left[\begin{array}{c|c} (\mathbb{i},\mathbb{x}) \notin \mathcal{L}(\mathcal{R}_{\mathsf{N}}) & \mathsf{pp} \leftarrow \mathsf{PC.Setup}(1^\lambda, D) \\ \wedge & \mathbb{F} \leftarrow \mathrm{field}(\mathsf{pp}) \\ \langle \tilde{\mathbf{P}}(\mathsf{st}), \mathbf{V}^{\mathbf{I}(\mathbb{F},\mathbb{i})}(\mathbb{F},\mathbb{x})\rangle = 1 & (\mathbb{i},\mathbb{x},\mathsf{st}) \leftarrow \tilde{\mathcal{P}}_1(\mathsf{pp}) \end{array}\right] + \Pr\left[\begin{array}{c} \mathcal{A}_{\mathsf{PC}} \text{ wins the} \\ \text{extractability game} \end{array}\right] \geq \epsilon(\lambda) \ ,$$

Above, D is computed according to Eq. (2.9) and $\mathbb{F}$ is the field described in pp. This concludes the proof because if $\epsilon(\lambda)$ were to be non-negligible then either: (i) by averaging there would exist a choice of public parameters pp that yields a state st, field $\mathbb{F} \in \mathcal{F}$, and $(\mathbb{i},\mathbb{x}) \notin \mathcal{L}(\mathcal{R})$ for which $\Pr[\langle \tilde{\mathbf{P}}(\mathsf{st}), \mathbf{V}^{\mathbf{I}(\mathbb{F},\mathbb{i})}(\mathbb{F},\mathbb{x})\rangle = 1]$ is non-negligible, contradicting our hypothesis AHP has negligible soundness error; or (ii) there would exist an efficient adversary $\mathcal{A}_{\mathsf{PC}}$ that, for any given efficient extractor, succeeds in the extractability game for PC (Definition 2.6.2) with non-negligible probability, contradicting our hypothesis PC is extractable.

Constructing $\mathcal{A}_{\mathsf{PC}}$. The adversary $\mathcal{A}_{\mathsf{PC}}$ is built from the argument prover $\tilde{\mathcal{P}}$ (and the argument indexer $\mathcal{I}$ and degree bounds d) as follows. For round $i \in \{0,\ldots,\mathsf{k}\}$ and verifier messages $\rho_0,\ldots,\rho_i$:

$\mathcal{A}_{\mathsf{PC}}(\mathsf{ck},\mathsf{rk},\rho_0,\rho_1,\ldots,\rho_i)$:

1. Set $\mathsf{srs} := \mathsf{pp}$ and compute $(\mathbb{i},\mathbb{x},\mathsf{st}) \leftarrow \tilde{\mathcal{P}}_1(\mathsf{srs})$.
2. If $i = 0$, ignore ρ_0, compute index keys $(\mathsf{ipk},\mathsf{ivk}) \leftarrow \mathcal{I}^{\mathsf{srs}}(\mathbb{i})$, and parse ivk as polynomial commitments $[c_{0,j}]_{j=1}^{\mathsf{s}(0)}$. If $i > 0$, compute polynomial commitments $[c_{i,j}]_{j=1}^{\mathsf{s}(i)} \leftarrow \tilde{\mathcal{P}}_2(\mathsf{st};\rho_1,\ldots,\rho_i)$.
3. For each $j \in \{1,\ldots,\mathsf{s}(i)\}$, compute the degree $d_{i,j} := \mathsf{d}(|\mathbb{i}|,i,j)$.
4. Output $([c_{i,j}]_{j=1}^{\mathsf{s}(i)}, [d_{i,j}]_{j=1}^{\mathsf{s}(i)})$.

Since $\tilde{\mathcal{P}}, \mathcal{I}, \mathsf{d}$ are all efficient, so is $\mathcal{A}_{\mathsf{PC}}$. Let $\mathcal{E}_{\mathsf{PC}}$ be the extractor for $\mathcal{A}_{\mathsf{PC}}$. Note that in the "0-th round", $\mathcal{A}_{\mathsf{PC}}$ outputs the commitments generated by the indexer $\mathcal{I}$. To capture that these "0-th round" commitments need only satisfy evaluation binding (unlike the commitments in all other rounds), we consider an extractor $\mathcal{E}'_{\mathsf{PC}}$ that works as follows. For round $i \in \{0, \ldots, \mathsf{k}\}$ and verifier messages $\rho_0, \ldots, \rho_i$:

$\mathcal{E}'_{\mathsf{PC}}(\mathsf{pp}, \rho_0, \rho_1, \ldots, \rho_i)$:
1. Set $\mathsf{srs} := \mathsf{pp}$ and compute $(\mathbb{i}, \mathbb{x}, \mathsf{st}) \leftarrow \tilde{\mathcal{P}}_1(\mathsf{srs})$.
   Obtain the field description $\mathbb{F} \leftarrow \mathrm{field}(\mathsf{pp})$.
2. If $i = 0$, output polynomials $[p_{0,j}]_{j=1}^{\mathsf{s}(0)} \leftarrow \mathbf{I}(\mathbb{F}, \mathbb{i})$.
   If $i > 0$, output polynomials $[p_{i,j}]_{j=1}^{\mathsf{s}(i)} \leftarrow \mathcal{E}_{\mathsf{PC}}(\mathsf{st}; \rho_1, \ldots, \rho_i)$.

Observe that the probability that $\mathcal{E}'_{\mathsf{PC}}$ succeeds for $\mathcal{A}_{\mathsf{PC}}$ is at least the probability that $\mathcal{E}_{\mathsf{PC}}$ succeeds for $\mathcal{A}_{\mathsf{PC}}$.

Constructing $\tilde{\mathbf{P}}$. We define $\tilde{\mathbf{P}}$ via its k next-message functions, by relying on the polynomial commitment extractor $\mathcal{E}'_{\mathsf{PC}}$ defined above. For round number $i \in \{1, \ldots, \mathsf{k}\}$ and verifier messages $\rho_1, \ldots, \rho_i$:

$\tilde{\mathbf{P}}(\mathsf{st}; \rho_1, \ldots, \rho_i)$:
1. Set $\rho_0 := \perp$ and run $\mathcal{E}'_{\mathsf{PC}}(\mathsf{pp}, \rho_0, \rho_1, \ldots, \rho_i)$ to obtain polynomials $p_{i,1}, p_{i,2}, \ldots, p_{i,\mathsf{s}(i)} \in \mathbb{F}[X]$.
2. Check that for every $j \in [\mathsf{s}(i)]$ it holds that $\deg(p_{i,j}) \leq \mathsf{d}(|\mathbb{i}|, i, j)$. (If not, output $\perp$.)
3. Output the polynomials $p_{i,1}, p_{i,2}, \ldots, p_{i,\mathsf{s}(i)}$.

Observe that, by construction, $\tilde{\mathbf{P}}$ is an admissible prover for AHP.

Analyzing $\tilde{\mathbf{P}}$ and $\mathcal{A}_{\mathsf{PC}}$. Define $\epsilon_{\mathsf{PC}}(\lambda) := \Pr[\mathcal{A}_{\mathsf{PC}} \text{ wins the extractability game}]$. We want to argue that

$$\Pr\left[\begin{array}{c|c} (\mathbb{i}, \mathbb{x}) \notin \mathcal{L}(\mathcal{R}_{\mathsf{N}}) & \mathsf{srs} \leftarrow \mathcal{G}(1^\lambda, \mathsf{N}) \\ \wedge & (\mathbb{i}, \mathbb{x}, \mathsf{st}) \leftarrow \tilde{\mathcal{P}}_1(\mathsf{srs}) \\ \langle \tilde{\mathcal{P}}_2(\mathsf{st}), \mathcal{V}(\mathsf{ivk}, \mathbb{x}) \rangle = 1 & (\mathsf{ipk}, \mathsf{ivk}) \leftarrow \mathcal{I}^{\mathsf{srs}}(\mathbb{i}) \end{array}\right]$$
$$\leq \Pr\left[\begin{array}{c|c} (\mathbb{i}, \mathbb{x}) \notin \mathcal{L}(\mathcal{R}_{\mathsf{N}}) & \mathsf{pp} \leftarrow \mathsf{PC.Setup}(1^\lambda, D) \\ \wedge & \mathbb{F} \leftarrow \mathrm{field}(\mathsf{pp}) \\ \langle \tilde{\mathbf{P}}(\mathsf{st}), \mathbf{V}^{\mathbf{I}(\mathbb{F}, \mathbb{i})}(\mathbb{F}, \mathbb{x}) \rangle = 1 & (\mathbb{i}, \mathbb{x}, \mathsf{st}) \leftarrow \tilde{\mathcal{P}}_1(\mathsf{pp}) \end{array}\right] + \epsilon_{\mathsf{PC}}(\lambda) \ .$$

First recall that by construction it holds that $\mathcal{G}(1^\lambda, \mathsf{N}) = \mathsf{PC.Setup}(1^\lambda, D)$. It follows that the distributions of srs/pp, $\mathbb{i}, \mathbb{x}, \mathsf{st}$, as well as the underlying field $\mathbb{F}$, are identical in the two probability expressions above.

Next recall that we have constructed $\tilde{\mathbf{P}}$ in such a way that, in round $i \in \{1, \ldots, \mathsf{k}\}$, $\tilde{\mathbf{P}}$ outputs polynomials that (provided $\mathcal{E}'_{\mathsf{PC}}$ has succeeded) correspond to the commitments output in round i by $\tilde{\mathcal{P}}_2$. At the same time, we have constructed $\mathcal{V}$ in such a way that in the first k rounds $\mathcal{V}$ behaves exactly as $\mathbf{V}$, and in the remaining two rounds $\mathcal{V}$ uses the polynomial commitment scheme to

validate, against commitments received from the prover and contained in ivk, the answers claimed by $\tilde{\mathcal{P}}_2$ in response to $\mathbf{V}$'s query set Q, and then checks that $\mathbf{V}$ accepts these answers.

Hence, as long as $\tilde{\mathcal{P}}$ provides correct evaluations for polynomials committed to in ivk, and as long as $\tilde{\mathbf{P}}$ outputs polynomials that correspond to the commitments output by $\tilde{\mathcal{P}}_2$, it holds that $\mathbf{V}$ accepts whenever $\mathcal{V}$ accepts. Since $\tilde{\mathbf{P}}$ relies on the extractor $\mathcal{E}'_{\mathsf{PC}}$ for the polynomial commitment to find such polynomials (if they exist) and to correctly answer queries to polynomials in ivk, $\tilde{\mathbf{P}}$ "works" whenever $\mathcal{E}'_{\mathsf{PC}}$ and $\tilde{\mathcal{P}}$ do.

We now argue that, whenever $\mathcal{V}$ accepts, $\mathcal{E}'_{\mathsf{PC}}$ has succeeded, up to the error $\epsilon_{\mathsf{PC}}(\lambda)$. This is because the interaction between $\tilde{\mathcal{P}}_2$ and $\mathcal{V}$ can be re-cast as an extractability game for PC, as we now explain. Define a public-coin challenger $\mathcal{C}$ to output randomness $\rho_0 := \bot$ in the 0-th round, and to equal the interactive phase of $\mathbf{V}(\mathbb{F}, \mathbb{x})$ in the remaining rounds. This means that in the i-th round (for $i \in \{1, \ldots, \mathsf{k}\}$) the challenger $\mathcal{C}$ will output the randomness ρ_i output by $\mathbf{V}(\mathbb{F}, \mathbb{x})$ in round i. Also, define a query sampler $\mathcal{Q}$ to equal the query phase of $\mathbf{V}(\mathbb{F}, \mathbb{x})$: given all challenger outputs $[\rho_j]_{j=1}^{\mathsf{k}}$ so far and auxiliary input $\rho_{\mathsf{k}+1}$, compute the query set $Q := \mathbf{Q}_{\mathbf{V}}(\mathbb{F}, \mathbb{x}; \rho_1, \ldots, \rho_{\mathsf{k}}, \rho_{\mathsf{k}+1})$. Finally, let $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ be the adversary defined below.

$\mathcal{B}_1(\mathsf{pp}, [\rho_j]_{j=0}^{\mathsf{k}}, Q)$:
1. Set $\mathsf{srs} := \mathsf{pp}$.
2. Compute $(\mathbb{i}, \mathbb{x}, \mathsf{st}) \leftarrow \tilde{\mathcal{P}}_1(\mathsf{srs})$.
3. Compute $\boldsymbol{v} \leftarrow \tilde{\mathcal{P}}_2(\mathsf{st}; \rho_1, \ldots, \rho_{\mathsf{k}}, Q)$.
4. Set $\mathsf{st}_{\mathsf{PC}} := (\mathsf{st}, \rho_1, \ldots, \rho_{\mathsf{k}}, Q)$.
5. Output $(\boldsymbol{v}, \mathsf{st}_{\mathsf{PC}})$.

$\mathcal{B}_2(\mathsf{st}_{\mathsf{PC}}, \xi)$:
1. Parse $\mathsf{st}_{\mathsf{PC}}$ as $(\mathsf{st}, \rho_1, \ldots, \rho_{\mathsf{k}}, Q)$
2. Compute $\pi \leftarrow \tilde{\mathcal{P}}_2(\mathsf{st}; \rho_1, \ldots, \rho_{\mathsf{k}}, Q, \xi)$.
3. Output π.

Using the above definitions of $\mathcal{C}$, $\mathcal{Q}$, $\mathcal{B}$, and $\mathcal{E}'_{\mathsf{PC}}$ we obtain the following inequality:

$$\Pr\left[\begin{array}{c|c} (\mathbb{i}, \mathbb{x}) \notin \mathcal{L}(\mathcal{R}_{\mathsf{N}}) & \mathsf{srs} \leftarrow \mathcal{G}(1^\lambda, \mathsf{N}) \\ \wedge & (\mathbb{i}, \mathbb{x}, \mathsf{st}) \leftarrow \tilde{\mathcal{P}}_1(\mathsf{srs}) \\ \langle \tilde{\mathcal{P}}_2(\mathsf{st}), \mathcal{V}(\mathsf{ivk}, \mathbb{x}) \rangle = 1 & (\mathsf{ipk}, \mathsf{ivk}) \leftarrow \mathcal{I}^{\mathsf{srs}}(\mathbb{i}) \end{array}\right]$$

$$\leq \Pr\left[\begin{array}{c|c} & \mathsf{pp} \leftarrow \mathsf{PC.Setup}(1^\lambda, D) \\ & \mathbb{F} \leftarrow \mathrm{field}(\mathsf{rk}) \\ & (\mathbb{i}, \mathbb{x}, \mathsf{st}) \leftarrow \tilde{\mathcal{P}}_1(\mathsf{pp}) \\ \hline (\mathbb{i}, \mathbb{x}) \notin \mathcal{L}(\mathcal{R}_{\mathsf{N}}) & \text{For } i = 0, \ldots, \mathsf{k}: \\ \wedge & \rho_i \leftarrow \mathcal{C}(\mathsf{pp}, i) \\ \mathsf{PC.Check}(\mathsf{rk}, \boldsymbol{c}, \boldsymbol{d}, Q, \boldsymbol{v}, \pi, \xi) = 1 & ([c_{i,j}]_{j=1}^{\mathsf{s}(i)}, [d_{i,j}]_{j=1}^{\mathsf{s}(i)}) \leftarrow \mathcal{A}_{\mathsf{PC}}(\mathsf{pp}, [\rho_j]_{j=0}^{i}) \\ \wedge & [p_{i,j}]_{j=1}^{\mathsf{s}(i)} \leftarrow \mathcal{E}'_{\mathsf{PC}}(\mathsf{pp}, [\rho_j]_{j=0}^{i}) \\ \deg(\boldsymbol{p}) \leq \boldsymbol{d} \leq D \text{ and } \boldsymbol{v} = \boldsymbol{p}(Q) & \hline \\ \wedge & Q \leftarrow \mathcal{Q}(\mathsf{pp}, [\rho_j]_{j=0}^{\mathsf{k}}; \rho_{\mathsf{k}+1}) \\ \langle \tilde{\mathbf{P}}(\mathsf{st}), \mathbf{V}^{\mathbf{I}(\mathbb{F}, \mathbb{i})}(\mathbb{F}, \mathbb{x}; \rho_1, \ldots, \rho_{\mathsf{k}}, \rho_{\mathsf{k}+1}) \rangle = 1 & (\boldsymbol{v}, \mathsf{st}) \leftarrow \mathcal{B}_1(\mathsf{pp}, [\rho_j]_{j=0}^{\mathsf{k}}, Q) \\ & \text{Sample opening challenge } \xi \\ & \pi \leftarrow \mathcal{B}_2(\mathsf{st}, \xi) \\ & \text{Set } \boldsymbol{c} := [[c_{i,j}]_{j=1}^{\mathsf{s}(i)}]_{i=0}^{\mathsf{k}}, \boldsymbol{d} := [[d_{i,j}]_{j=1}^{\mathsf{s}(i)}]_{i=0}^{\mathsf{k}} \\ & (\mathsf{ck}, \mathsf{rk}) \leftarrow \mathsf{PC.Trim}^{\mathsf{pp}}(1^\lambda, \boldsymbol{d}) \end{array}\right] + \epsilon_{\mathsf{PC}}(\lambda) \ .$$

As argued above, whenever $\mathcal{E}'_{\mathsf{PC}}$ and $\tilde{\mathcal{P}}$ succeed, $\tilde{\mathbf{P}}$ does. The first term after the inequality captures the case where the AHP verifier $\mathbf{V}$ is convinced to accept a pair $(\mathbb{i}, \mathbb{x})$ not in the indexed language $\mathcal{L}(\mathcal{R}_{\mathsf{N}})$. If $\mathcal{A}_{\mathsf{PC}}$ succeeds, then there is still some chance that $\tilde{\mathbf{P}}$ succeeds assuming it holds that

$\deg(\boldsymbol{p}) \leq \boldsymbol{d} \leq D$ for the polynomials output by $\mathcal{E}'_{\mathsf{PC}}$ (otherwise $\tilde{\mathbf{P}}$ outputs $\perp$). This joint success probability is upper bounded by the probability that just $\mathcal{A}_{\mathsf{PC}}$ succeeds, which is in turn upper bounded by $\epsilon_{\mathsf{PC}}(\lambda)$. Hence the $\epsilon_{\mathsf{PC}}(\lambda)$ term above and the inequality rather than equality above. Since the above inequality implies our claim, we have concluded the proof.

2.8.3 Proof of Theorem 2.8.3

Let $\mathbf{E}$ be the extractor for AHP, which by hypothesis has a negligible knowledge soundness error $\epsilon_{\mathsf{AHP}}(\lambda)$. Suppose that $\tilde{\mathcal{P}} = (\tilde{\mathcal{P}}_1, \tilde{\mathcal{P}}_2)$ is an efficient adversary for ARG. We use $\tilde{\mathcal{P}}$ to construct an admissible prover $\tilde{\mathbf{P}}$ for AHP, exactly as in the proof of soundness (see Section 2.8.2). Then we define the extractor $\mathcal{E}$ for $\tilde{\mathcal{P}}$ to be as follows.

$\mathcal{E}(\mathsf{srs})$:

1. Compute $(\mathbb{i}, \mathbb{x}, \mathsf{st}) \leftarrow \tilde{\mathcal{P}}_1(\mathsf{srs})$.
2. Compute $\mathbb{F} \leftarrow \mathrm{field}(\mathsf{srs})$.
3. Compute $\mathbb{w} \leftarrow \mathbf{E}^{\tilde{\mathbf{P}}(\mathsf{st})}(\mathbb{F}, \mathbb{i}, \mathbb{x}, 1^{\mathsf{I}(|\mathbb{i}|)})$.
4. Output $\mathbb{w}$.

Observe that by construction we have the equality:

$$\Pr\left[\begin{array}{c} (\mathbb{i}, \mathbb{x}, \mathbb{w}) \notin \mathcal{R}_{\mathsf{N}} \\ \wedge \\ \langle \tilde{\mathcal{P}}_2(\mathsf{st}), \mathcal{V}(\mathsf{ivk}, \mathbb{x})\rangle = 1 \end{array} \middle| \begin{array}{r} \mathsf{srs} \leftarrow \mathcal{G}(1^\lambda, \mathsf{N}) \\ (\mathbb{i}, \mathbb{x}, \mathsf{st}) \leftarrow \tilde{\mathcal{P}}_1(\mathsf{srs}) \\ (\mathsf{ipk}, \mathsf{ivk}) \leftarrow \mathcal{I}^{\mathsf{srs}}(\mathbb{i}) \\ \mathbb{w} \leftarrow \mathcal{E}(\mathsf{srs}) \end{array}\right]$$

$$= \Pr\left[\begin{array}{c} (\mathbb{i}, \mathbb{x}, \mathbb{w}) \notin \mathcal{R}_{\mathsf{N}} \\ \wedge \\ \langle \tilde{\mathcal{P}}_2(\mathsf{st}), \mathcal{V}(\mathsf{rk}, \mathsf{ivk}, \mathbb{x})\rangle = 1 \end{array} \middle| \begin{array}{r} \mathsf{pp} \leftarrow \mathsf{PC.Setup}(1^\lambda, D) \\ \mathbb{F} \leftarrow \mathrm{field}(\mathsf{pp}) \\ (\mathbb{i}, \mathbb{x}, \mathsf{st}) \leftarrow \tilde{\mathcal{P}}_1(\mathsf{pp}) \\ (\mathsf{ipk}, \mathsf{ivk}) \leftarrow \mathcal{I}^{\mathsf{pp}}(\mathbb{i}) \\ \mathbb{w} \leftarrow \mathbf{E}^{\tilde{\mathbf{P}}(\mathsf{st})}(\mathbb{F}, \mathbb{i}, \mathbb{x}, 1^{\mathsf{I}(|\mathbb{i}|)}) \end{array}\right] .$$

Similarly to the proof of soundness (see Section 2.8.2), we can argue the following inequality:

$$\Pr\left[\begin{array}{c} (\mathbb{i}, \mathbb{x}, \mathbb{w}) \notin \mathcal{R}_{\mathsf{N}} \\ \wedge \\ \langle \tilde{\mathcal{P}}_2(\mathsf{st}), \mathcal{V}(\mathsf{ivk}, \mathbb{x})\rangle = 1 \end{array} \middle| \begin{array}{r} \mathsf{pp} \leftarrow \mathsf{PC.Setup}(1^\lambda, D) \\ \mathbb{F} \leftarrow \mathrm{field}(\mathsf{pp}) \\ (\mathbb{i}, \mathbb{x}, \mathsf{st}) \leftarrow \tilde{\mathcal{P}}_1(\mathsf{pp}) \\ (\mathsf{ipk}, \mathsf{ivk}) \leftarrow \mathcal{I}^{\mathsf{pp}}(\mathbb{i}) \\ \mathbb{w} \leftarrow \mathbf{E}^{\tilde{\mathbf{P}}(\mathsf{st})}(\mathbb{F}, \mathbb{i}, \mathbb{x}, 1^{\mathsf{I}(|\mathbb{i}|)}) \end{array}\right]$$

$$\leq \Pr\left[\begin{array}{c} (\mathbb{i}, \mathbb{x}, \mathbb{w}) \notin \mathcal{R}_{\mathsf{N}} \\ \wedge \\ \langle \tilde{\mathbf{P}}(\mathsf{st}), \mathbf{V}^{\mathbf{I}(\mathbb{F}, \mathbb{i})}(\mathbb{F}, \mathbb{x})\rangle = 1 \end{array} \middle| \begin{array}{r} \mathsf{pp} \leftarrow \mathsf{PC.Setup}(1^\lambda, D) \\ \mathbb{F} \leftarrow \mathrm{field}(\mathsf{pp}) \\ (\mathbb{i}, \mathbb{x}, \mathsf{st}) \leftarrow \tilde{\mathcal{P}}_1(\mathsf{pp}) \\ \mathbb{w} \leftarrow \mathbf{E}^{\tilde{\mathbf{P}}(\mathsf{st})}(\mathbb{F}, \mathbb{i}, \mathbb{x}, 1^{\mathsf{I}(|\mathbb{i}|)}) \end{array}\right] + \epsilon_{\mathsf{PC}}(\lambda) .$$

The knowledge soundness of AHP implies that the probability above is at most $\epsilon_{\mathsf{AHP}}(\lambda)$. Since $\epsilon_{\mathsf{AHP}}(\lambda) + \epsilon_{\mathsf{PC}}(\lambda)$ is negligible, we have established that the extractor $\mathcal{E}$ for $\tilde{\mathcal{P}}$ works.

2.8.4 Proof of Theorem 2.8.4

Let $\mathbf{S}$ be the zero knowledge simulator for AHP (see definition in Section 2.4), and let $\mathcal{S}_{\mathsf{PC}}$ be the simulator for PC (see definition in Section 2.6). We describe how to construct a (perfect) zero knowledge simulator $\mathcal{S} = (\mathsf{Setup}, \mathsf{Prove})$ for ARG (see definition in Section 2.7). Let $\tilde{\mathcal{V}} = (\tilde{\mathcal{V}}_1, \tilde{\mathcal{V}}_2)$ be any malicious verifier.

The simulated setup algorithm $\mathcal{S}.\mathsf{Setup}$ receives a security parameter $\lambda \in \mathbb{N}$ and size bound $\mathsf{N} \in \mathbb{N}$ as input, and then proceeds as follows. First, $\mathcal{S}.\mathsf{Setup}$ uses N to compute the same maximum degree bound $D \in \mathbb{N}$ computed by the generator $\mathcal{G}$ (see Eq. (2.9)). Second, it runs $\mathcal{S}_{\mathsf{PC}}.\mathsf{Setup}(1^\lambda, D)$ to sample simulated public parameters pp for the polynomial commitment and their trapdoor trap, and outputs $(\mathsf{srs}, \mathsf{trap}) := (\mathsf{pp}, \mathsf{trap})$. Let $\mathbb{F} \in \mathcal{F}$ be the field described in the public parameters pp.

The zero knowledge game states that first $\tilde{\mathcal{V}}_1$ receives srs, and then outputs an index-instance-witness tuple $(\mathbb{i}, \mathbb{x}, \mathbb{w})$ and a state st to pass onto $\tilde{\mathcal{V}}_2$. The proving subroutine of the simulator, $\mathcal{S}.\mathsf{Prove}$, receives $(\mathsf{trap}, \mathbb{i}, \mathbb{x})$ as input, and interacts with $\tilde{\mathcal{V}}_2(\mathsf{st})$ over $\mathsf{k}+2$ rounds. We construct $\mathcal{S}.\mathsf{Prove}$ as follows.

1. For $i \in \{1, \ldots, \mathsf{k}\}$, simulate the polynomial commitments for round i as follows:
   a) Receive a message $\rho_i \in \mathbb{F}^*$ from $\tilde{\mathcal{V}}_2$, and forward it to the AHP simulator $\mathbf{S}(\mathbb{F}, \mathbb{i}, \mathbb{x})$.
   b) Sample commitment randomness $[\omega_{i,j}]_{j=1}^{\mathsf{s}(i)}$, and then send to $\tilde{\mathcal{V}}_2$ the simulated commitments below
   $$[c_{i,j}]_{j=1}^{\mathsf{s}(i)} \leftarrow \mathcal{S}_{\mathsf{PC}}.\mathsf{Commit}(\mathsf{trap}, [\mathsf{d}(|\mathbb{i}|, i, j)]_{j=1}^{\mathsf{s}(i)}; [\omega_{i,j}]_{j=1}^{\mathsf{s}(i)}) \ .$$
2. Simulate the evaluations in round $\mathsf{k}+1$ as follows:
   a) Receive a message $\rho_{\mathsf{k}+1} \in \mathbb{F}^*$ from $\tilde{\mathcal{V}}_2$.
   b) Use the query algorithm of AHP to compute the query set $Q := \mathbf{Q}_{\mathbf{V}}(\mathbb{F}, \mathbb{x}; \rho_1, \ldots, \rho_{\mathsf{k}}, \rho_{\mathsf{k}+1})$, and abort if any query does not satisfy the query checker $\mathbf{C}$. (The honest prover would also abort.)
   c) We need to assemble a list of evaluations $\boldsymbol{v}$, containing actual evaluations of index polynomials and simulated evaluations of prover polynomials. In more detail, first run the AHP indexer $\mathbf{I}(\mathbb{F}, \mathbb{i})$ to obtain polynomials $[p_{0,j}]_{j=1}^{\mathsf{s}(0)}$, and evaluate these on (the relevant queries in) the query set Q. Next, forward the query set Q to the AHP simulator $\mathbf{S}(\mathbb{F}, \mathbb{i}, \mathbb{x})$ in order to obtain a simulated view, which in particular contains simulated answers for queries to the AHP prover's polynomials.
3. Simulate the evaluation proof in round $\mathsf{k}+2$ as follows:
   a) Receive a challenge ξ from $\tilde{\mathcal{V}}_2$.
   b) Compute proof $\pi \leftarrow \mathcal{S}_{\mathsf{PC}}.\mathsf{Open}(\mathsf{trap}, [[p_{i,j}]_{j=1}^{\mathsf{s}(i)}]_{i=0}^{\mathsf{k}}, \boldsymbol{v}, [[\mathsf{d}(|\mathbb{i}|, i, j)]_{j=1}^{\mathsf{s}(i)}]_{i=0}^{\mathsf{k}}, Q, \xi; [[\omega_{i,j}]_{j=1}^{\mathsf{s}(i)}]_{i=0}^{\mathsf{k}})$, where all polynomials $[p_{i,j}]_{j=1}^{\mathsf{s}(i)}$ with $i > 0$ are defined to be zero and the randomness $[\omega_{0,j}]_{j=1}^{\mathsf{s}(0)}$ is set to $\perp$.
   c) Send π to $\tilde{\mathcal{V}}_2$.

Lemma 2.8.7. *The view of the malicious verifier $\tilde{\mathcal{V}} = (\tilde{\mathcal{V}}_1, \tilde{\mathcal{V}}_2)$ while interacting with the honest prover is identically distributed as its view while interacting with the simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$ described above.*

Proof. The zero knowledge property of AHP states that interaction with the honest prover $\mathbf{P}(\mathbb{F}, \mathbb{i}, \mathbb{x}, \mathbb{w})$ can be replaced with interaction with the simulator $\mathbf{S}(\mathbb{F}, \mathbb{i}, \mathbb{x})$, which adaptively answers oracle queries of the malicious verifier to prover oracles, *provided* the number of oracle queries is below the zero knowledge query bound and each query satisfies the query checker. In our setting, the number of oracle queries is bounded by the query complexity q of the *honest* AHP verifier, because the query set Q is derived via the honest query algorithm run on the messages sent by the malicious argument verifier. Moreover, the honest prover and simulator ensure that each query in Q satisfies the query checker. This explains why the zero knowledge query bound in Theorem 2.8.4 is q, and why we consider any *polynomial-time* query checker in Theorem 2.8.4.

Next, given that $\mathbf{S}(\mathbb{F}, \mathbb{i}, \mathbb{x})$ provides oracle responses that are identically distributed to those of polynomials output by $\mathbf{P}(\mathbb{F}, \mathbb{i}, \mathbb{x}, \mathbb{w})$, we are left to discuss the other information received by the malicious verifier: the commitments (in the first k rounds) and the evaluation proof (in round $\mathsf{k} + 2$). The hiding property of the polynomial commitment scheme ensures that the simulator $\mathcal{S}_{\mathsf{PC}}$, by using the trapdoor trap, can perfectly simulate these commitments and this evaluation proof. □

2.9 MARLIN: an efficient preprocessing zkSNARK with universal SRS

We describe how to obtain a preprocessing zkSNARK with universal and updatable SRS that achieves the efficiency reported in Fig. 2.1.

The first step is to apply our compiler (Section 2.8) to two ingredients: the AHP described in Section 2.5, and the AGM-based polynomial commitment scheme described in Sections 2.6.2 and 2.13.1. The second step is to apply the Fiat–Shamir transformation to the resulting public-coin preprocessing argument. These "generic" steps immediately yield a *preprocessing zkSNARK with universal and updatable SRS that has the same asymptotics as Sonic [MBKM19].*[11] Moreover, in terms of concrete efficiency, this zkSNARK achieves argument size comparable to Sonic [MBKM19], and also achieves proving and verification times that are close to the state of the art for *circuit-specific* zkSNARKs [Gro16].

Below in Sections 2.9.1 and 2.9.2 we describe optimizations that further reduce argument size, and as a positive side effect also reduce prover and verifier costs. Fig. 2.1 includes these optimizations.

Before we discuss optimizations, we summarize the argument size that we obtain directly from the compilation mentioned above. Recall that in the offline phase, the AHP indexer, given an index $\mathbb{i} = (\mathbb{F}, H, K, A, B, C)$, outputs for each matrix $M \in \{A, B, C\}$ three polynomials that together define the low-degree extension of M. Then, during the interactive online phase, the prover outputs twelve proof oracles. The verifier queries each of the nine indexer polynomials and the twelve prover polynomials at exactly one location, which amounts to 21 queries.

After compilation, the argument indexer outputs 9 polynomial commitments, and the argument prover outputs 12 commitments, 21 evaluations, and 3 evaluation proofs. In more detail, the argument indexer outputs commitments to $\hat{\mathsf{row}}_M, \hat{\mathsf{col}}_M, \hat{\mathsf{val}}_M$ for each $M \in \{A, B, C\}$; and the argument prover outputs commitments to the following twelve polynomials: $\hat{w}, \hat{z}_A, \hat{z}_B, \hat{z}_C$, h_0, s, h_1, g_1, h_2, g_2, h_3, g_3. The polynomials $\hat{w}, \hat{z}_A, \hat{z}_B, \hat{z}_C$, h_0, s, h_1, g_1 are all evaluated at the same point β_1; h_2 and g_2 are evaluated at the same point β_2; and h_3, g_3, and $\hat{\mathsf{row}}_M, \hat{\mathsf{col}}_M, \hat{\mathsf{val}}_M$ for each $M \in \{A, B, C\}$ are all evaluated at the same point β_3. Overall our argument consists of 27 $\mathbb{G}_1$ elements and 24 $\mathbb{F}_q$ elements.

2.9.1 Optimizations for the AHP

Eliminating h_0 and $\hat{z}_C$. The AHP prover $\mathbf{P}$ sends a polynomial $h_0(X)$ in the first round, and the AHP verifier $\mathbf{V}$ checks the polynomial equation $\hat{z}_A(X)\hat{z}_B(X) - \hat{z}_C(X) = h_0(X)v_H(X)$ at a random point. This is a standard technique from the probabilistic proof literature to ensure that $\hat{z}_A(X)\hat{z}_B(X)$ and $\hat{z}_C(X)$ agree on H. An alternative (used, e.g., in [BCGRS17]) is to replace each

[11] The SRS of the zkSNARK is updatable because the SRS of the polynomial commitment scheme is updatable (see Remark 2.8.2 and Section 2.6.2). Note also that the query algorithm in the AHP fulfills the admissibility requirement imposed by the polynomial commitment scheme (see Section 2.6.2), as each query location is sampled at random from a set of super-polynomial size.

occurrence of $\hat{z}_C(X)$ in the protocol with the product $\hat{z}_A(X)\hat{z}_B(X)$, which "forces" the desired property without any checks. This increases the degree of certain expressions by $\deg(\hat{z}_C) = |H| - 1$, but this cost in our setting is negligible because it leads to a negligible increase in the soundness error. This eliminates the need to commit to $h_0(X)$ and $\hat{z}_C(X)$ and later reveal their evaluations, which reduces argument size by two polynomial commitments and two field elements.

Minimal zero knowledge query bound. The query algorithm of the AHP verifier $\mathbf{V}$ queries each prover polynomial at exactly one location, regardless of the randomness used to generate the queries. In particular, $\hat{w}(X), \hat{z}_A(X), \hat{z}_B(X), \hat{z}_C(X)$ are queried at exactly one location. So it suffices to set the parameter $\mathsf{b} := 1$.

Eliminating $\boldsymbol{\sigma_1}$. We can sample the random polynomial $s(X)$ conditioned on it summing to zero on H. The prover can thus omit σ_1, because it will always be zero, without affecting zero knowledge.

Single low-degree extension for each matrix (unimplemented). The AHP indexer $\mathbf{I}$ constructs the low-degree extensions of the nine functions $\{\mathsf{row}_M, \mathsf{col}_M, \mathsf{val}_M\}_{M \in \{A,B,C\}}$, which define the low-degree extensions of A, B, C. The AHP verifier $\mathbf{V}$ queries each of these at a single location. This means that, after compilation, the argument prover must provide *nine* field elements (the evaluations) as part of the proof.

We can reduce this to only *three* field elements as follows. We modify the AHP indexer $\mathbf{I}$ to construct, for each $M \in \{A, B, C\}$, a *single* low-degree extension of the functions $\mathsf{row}_M, \mathsf{col}_M, \mathsf{val}_M$. Namely, let $s_1, s_2 \in \mathbb{F}$ be "shifts" such that K, $K + s_1$, and $K + s_2$ are pairwise disjoint, and define the set $\bar{K} := K \cup (K + s_1) \cup (K + s_2)$. Define the function $\mathsf{m}_M \colon \bar{K} \to \mathbb{F}$ where

$$\mathsf{m}_M(\kappa) := \begin{cases} \mathsf{row}_M(\kappa) & \kappa \in K \\ \mathsf{col}_M(\kappa - s_1) & \kappa \in K + s_1 \\ \mathsf{val}_M(\kappa - s_2) & \kappa \in K + s_2 \end{cases} .$$

Then Eq. (2.1) can be rewritten as

$$\hat{M}(X, Y) := \sum_{\kappa \in K} u_H(X, \hat{\mathsf{m}}_M(\kappa)) u_H(Y, \hat{\mathsf{m}}_M(\kappa + s_1)) \hat{\mathsf{m}}_M(\kappa + s_2) \ . \tag{2.10}$$

The modified AHP indexer $\mathbf{I}$ constructs the three polynomials $\hat{\mathsf{m}}_A, \hat{\mathsf{m}}_B, \hat{\mathsf{m}}_C$, and the modified AHP verifier $\mathbf{V}$ will query each of these at a single location. Thus, after compilation, the argument prover will only need to provide three field elements, instead of nine, as part of the proof. Note that this optimization triples the degree of the polynomials output by the AHP indexer $\mathbf{I}$, which after compilation increases the SRS size. Even given this tradeoff our SRS is still shorter than prior work, and furthermore it represents a one-time offline cost (in contrast to argument size, which is a recurring online cost).

2.9.2 Optimizations for the polynomial commitment scheme

Reducing the cost of hiding commitments. The hiding property that we adopt for polynomial commitments (Definition 2.6.4) ensures that no information is revealed about the committed

polynomial regardless of how many evaluations are revealed. Achieving this strong notion has a cost: in our constructions we randomize a commitment c to a polynomial p by additionally committing to a random polynomial $\bar{p}$ of degree $\deg(p)$. Compared to the non-hiding variant, this requires $\deg(p)$ additional elements in the SRS, and also requires $\mathsf{PC.Commit}$ and $\mathsf{PC.Open}$ to perform an additional variable-base MSM of size $\deg(p)$.

In our compiler, however, the only evaluations the argument verifier sees are those sent by the argument prover, and these are determined by the query sets produced by the query algorithm. This, together with the fact in our AHP each polynomial is queried at exactly one location, implies that we can relax our construction to provide hiding only for a single evaluation per polynomial. Concretely, we can set $\bar{p}$ to have degree 1. (Note that $\bar{p}$ cannot be a constant because it is used to hide both the commitment to p and to hide the commitment to the witness polynomial w.) This allows us to eliminate (most of) the additional generators from the SRS, and the additional variable-base MSM for $\mathsf{PC.Commit}$ and $\mathsf{PC.Open}$.

Reducing the number of hiding commitments. Each hiding commitment, even taking into account the above optimizations, requires an evaluation proof that is one field element larger than a proof in the non-hiding case. We reduce this overhead by using the fact that only certain polynomials reveal information about the witness and necessitate hiding. In particular, only the polynomials $\hat{w}, \hat{z}_A, \hat{z}_B, \hat{z}_C, s, h_1$, and g_1 need hiding commitments. All other polynomials can rely on non-hiding commitments because they can be derived in polynomial-time from the index $\mathbb{i}$. This observation removes a further 1 field element from the proof.

Eliminating unnecessary degree checks. The notion of polynomial commitment scheme that we consider enables each commitment to guarantee a chosen degree bound that is up to the maximum degree bound chosen for the SRS. This flexibility has a cost: ensuring a degree bound strictly less than the maximum degree bound requires two group elements per commitment, corresponding to unshifted and shifted polynomials respectively. When compiling our AHP, we need this feature only when committing to g_1, g_2, g_3 (the exact degree bound matters for soundness) but for all other polynomials it suffices to rely on the maximum degree bound and so for them we omit the shifted polynomials altogether. This increases the soundness error by a negligible amount (which is fine), and lets us reduce argument size by 9 group elements.

Batching pairing equations. We can reduce the cost of the argument verifier by batching pairing equations. Recall that, to verify an evaluation proof with evaluation v and point z, $\mathsf{PC.Check}$ needs to check the pairing equation $e(U - vG - \gamma\bar{v}G, H) = e(\mathsf{w}, \beta H - zH)$. In our compiled zkSNARK, $\mathsf{PC.Check}$ is invoked three times, each with different values of U, w, z, and v. This results in 3 pairing equations. To reduce the number of pairing equations needed down to just one, we use the following reduction that ensures that the $\mathbb{G}_2$ argument to every pairing is constant:

$$\begin{aligned} e(U - vG - \gamma\bar{v}G, H) &= e(\mathsf{w}, \beta H - zH) \\ &= e(\mathsf{w}, \beta H) \cdot e(\mathsf{w}, -zH) \\ &= e(\mathsf{w}, \beta H) \cdot e(-z\mathsf{w}, H) \ . \end{aligned}$$

Hence, we have that

$$e(U - vG - \gamma\bar{v}G + z\mathsf{w}, H) = e(\mathsf{w}, \beta H) \ .$$

Because we have three proofs to check, the verifier has to check three of the above equations. These equations can be batch verified together as follows. The verifier samples a random field element r, and then uses the identity $\prod_i e(G_i, H)^{r^i} = e(\sum_i r^i G_i, H)$ to check the following equation:

$$e(\sum_i r^i (C_{0,i} - v_i G - \gamma\bar{v}_i G + z_i \mathsf{w}_i), H) = e(\sum_i r^i \mathsf{w}_i, \beta H) \ .$$

By properties of random linear combinations, the above equation holds only if each of the individual equations also hold (up to a negligible soundness error). In sum, the verifier only needs to evaluate two pairings.

Opening linear combinations of polynomials. The decision procedure of the AHP verifier checks polynomial equations such as

$$p_1(X) + p_2(X)p_3(X) = p_4(X) \ . \tag{2.11}$$

It does so by querying the polynomials $p_1, \ldots, p_4$ at a random point $z \in \mathbb{F}$, and then checking that the above equation holds with respect to the resulting evaluations $p_1(z), \ldots, p_4(z)$. To enable the compiled SNARK verifier to invoke the AHP decision procedure, the SNARK proof must also contain these evaluations. However, if we instead enable the AHP verifier to query *linear combinations* of polynomial oracles, then one can avoid providing all these evaluations. For example, we can rewrite the check in Equation (2.11) as follows:

$$p_2(z) = v_2 \quad \text{and} \quad p_5(X) := p_1(X) + v_2 p_3(X) - p_4(X) = 0 \ .$$

Then the AHP decision procedure only needs the evaluation $p_2(z)$, which means that the corresponding SNARK proof will contain only 1 field element, instead of 4. For this, we need that the polynomial commitment scheme allows checking evaluations of linear combinations of committed polynomials. The schemes constructed in Section 2.13 have linearly homomorphic commitments, and so support this feature.

Applying this optimization to the equations in our AHP reduces the proof size of the corresponding compiled SNARK by 10 field elements.

2.10 Cryptographic assumptions

We describe the cryptographic assumptions that underlie the constructions of polynomial commitment schemes in this work (Section 2.6.2). In Section 2.10.1 we define *bilinear group samplers*. In Section 2.10.2 we define (a minor variant of) the Strong Diffie–Hellman Assumption. In Section 2.10.3 we define (a minor variant of) the Power Knowledge of Exponent Assumption. In Section 2.10.4 we recall the *Algebraic Group Model*.

2.10.1 Bilinear groups

The cryptographic primitives that we construct in this work rely on cryptographic assumptions about bilinear groups. We formalize these via a *bilinear group sampler*, which is a probabilistic polynomial-time algorithm $\mathsf{SampleGrp}$ that, on input a security parameter λ (represented in unary), outputs a tuple $\langle \mathsf{group} \rangle = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, G, H, e)$ where $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ are groups of a prime order $q \in \mathbb{N}$, G generates $\mathbb{G}_1$, H generates $\mathbb{G}_2$, and $e \colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a (non-degenerate) bilinear map.

2.10.2 Strong Diffie–Hellman

Assumption 1 ([BB04])**.** *The* **Strong Diffie–Hellman (**SDH**) Assumption** *states that for every efficient adversary $\mathcal{A}$ and degree bound $d \in \mathbb{N}$ the following probability is negligible in λ:*

$$\Pr\left[\, C = \tfrac{1}{\beta+c} G \;\middle|\; \begin{array}{r} \langle \mathsf{group} \rangle \leftarrow \mathsf{SampleGrp}(1^\lambda) \\ \beta \leftarrow \mathbb{F}_q \\ \boldsymbol{\Sigma} \leftarrow \{\{\beta^i G\}_{i=0}^{d}, \beta H\} \\ (c, C) \leftarrow \mathcal{A}(\langle \mathsf{group} \rangle, \boldsymbol{\Sigma}) \end{array} \right] .$$

2.10.3 Power knowledge of exponent

The non-hiding variant of our polynomial commitment scheme relies on the PKE assumption below, while the variant relies on the dPKE assumption below.

Assumption 2 ([Gro10])**.** *The* **Power Knowledge of Exponent (**PKE**) Assumption** *states that for every efficient adversary $\mathcal{A}$ and degree bound $d \in \mathbb{N}$ there exists an efficient extractor $\mathcal{E}$ such that for every benign auxiliary input distribution $\mathcal{Z}$ the following probability is negligible in λ:*

$$\Pr\left[\begin{array}{c} G_1 = \alpha G_0 \\ \wedge \\ G_0 \neq \sum_{i=0}^{d} a_i \beta^i G \end{array} \;\middle|\; \begin{array}{r} \langle \mathsf{group} \rangle \leftarrow \mathsf{SampleGrp}(1^\lambda) \\ \mathbb{z} \leftarrow \mathcal{Z}(\langle \mathsf{group} \rangle) \\ \alpha, \beta \leftarrow \mathbb{F}_q \\ \boldsymbol{\Sigma} \leftarrow \{\{\beta^i G, \alpha\beta^i G\}_{i=0}^{d}, \alpha H, \beta H\} \\ (G_0, G_1) \leftarrow \mathcal{A}(\langle \mathsf{group} \rangle, \boldsymbol{\Sigma}, \mathbb{z}) \\ (a_0, \ldots, a_d) \leftarrow \mathcal{E}(\langle \mathsf{group} \rangle, \boldsymbol{\Sigma}, \mathbb{z}) \end{array} \right] .$$

Assumption 3. *The* **duplex Power Knowledge of Exponent Assumption** *(*dPKE*) states that for every efficient adversary $\mathcal{A}$ and degree bound $d \in \mathbb{N}$ there exists an efficient extractor $\mathcal{E}$ such that for every benign auxiliary input distribution $\mathcal{Z}$ the following probability is negligible in λ:*

$$\Pr\left[\begin{array}{c} G_1 = \alpha G_0 \\ \wedge \\ G_0 \neq \sum_{i=0}^{d} a_i \beta^i G + \sum_{i=0}^{d} b_i \gamma \beta^i G \end{array} \;\middle|\; \begin{array}{r} \langle \mathsf{group} \rangle \leftarrow \mathsf{SampleGrp}(1^\lambda) \\ \mathbb{z} \leftarrow \mathcal{Z}(\langle \mathsf{group} \rangle) \\ \alpha, \beta, \gamma \leftarrow \mathbb{F}_q \\ \mathbf{\Sigma} \leftarrow \{\{\beta^i G, \alpha\beta^i G, \gamma\beta^i G, \alpha\gamma\beta^i G\}_{i=0}^{d}, \alpha H, \beta H\} \\ (G_0, G_1) \leftarrow \mathcal{A}(\langle \mathsf{group} \rangle, \mathbf{\Sigma}, \mathbb{z}) \\ (a_0, \ldots, a_d, b_0, \ldots, b_d) \leftarrow \mathcal{E}(\langle \mathsf{group} \rangle, \mathbf{\Sigma}, \mathbb{z}) \end{array}\right].$$

Remark 2.10.1 (benign auxiliary inputs)**.** Extraction with auxiliary input requires that the auxiliary input is sampled from a "benign" distribution, as discussed in [BP15; BCPR16]. In this work we only rely on auxiliary inputs that consist of a prescribed number of random field elements, which are indeed considered benign.

Remark 2.10.2 (asymmetric PKE)**.** The PKE assumption in [Gro10] is stated for *symmetric* bilinear group samplers ($\mathbb{G}_1 = \mathbb{G}_2$). Instead, like many prior works, we consider *asymmetric* bilinear group samplers due to efficiency reasons. Our approach to adapting PKE to the asymmetric setting differs from that taken in prior works such as [GGPR13; DFGK14]. Prior constructions rely on secret powers of β in $\mathbb{G}_2$ for both completeness and security (and in particular incur the costs of many $\mathbb{G}_2$ exponentiations). In contrast, our constructions (of polynomial commitment schemes) do not need secret powers of β in $\mathbb{G}_2$, for either completeness or security, and therefore are not part of the inputs to the adversary. (Also see Section 2.10.5.)

Remark 2.10.3 (prior duplex variants)**.** The dPKE assumption is similar to, but different from, the assumption used in [ZGKPP17b; ZGKPP17a]. Namely, in dPKE the instance contains powers of β with respect to a different generator γG, whereas the assumption in [ZGKPP17b; ZGKPP17a] contains powers of γ with respect to G.

2.10.3.1 Extractability with multiple knowledge commitments

The PKE assumption implies a similar assumption where the adversary may output *multiple* knowledge commitments and its corresponding extractor must extract a linear combination for each knowledge commitment. We call this assumption MPKE, where the letter "M" denotes "multiple". MPKE implies PKE, so the two assumptions are equivalent. We use MPKE to prove extractability of the non-hiding variant our polynomial commitment scheme. In order to prove extractability of the hiding variant we rely on dMPKE, an analogous generalization of dPKE to the case of multiple knowledge commitments.

Assumption 4. *The* MPKE **assumption** *states that for every efficient adversary $\mathcal{A}$ and degree bound $d \in \mathbb{N}$ there exists an efficient extractor $\mathcal{E}$ such that for every* benign *auxiliary input distribution $\mathcal{Z}$,*

the following probability is negligible in λ:

$$\Pr\left[\begin{array}{c} \exists\, i \in [n] \text{ such that} \\ \\ G_{i,1} = \alpha G_{i,0} \\ \wedge \\ G_{i,0} \neq \sum_{j=0}^{d} a_{i,j}\beta^j G \end{array} \middle| \begin{array}{r} \langle \mathsf{group} \rangle \leftarrow \mathsf{SampleGrp}(1^\lambda) \\ \mathbb{z} \leftarrow \mathcal{Z}(\langle \mathsf{group} \rangle) \\ \alpha, \beta \leftarrow \mathbb{F}_q \\ \mathbf{\Sigma} \leftarrow \{\{\beta^i G, \alpha\beta^i G\}_{i=0}^{d}, \beta H, \alpha H\} \\ [(G_{i,0}, G_{i,1})]_{i=1}^{n} \leftarrow \mathcal{A}(\langle \mathsf{group} \rangle, \mathbf{\Sigma}, \mathbb{z}) \\ [[a_{i,j}]_{j=0}^{d}]_{i=1}^{n} \leftarrow \mathcal{E}(\langle \mathsf{group} \rangle, \mathbf{\Sigma}, \mathbb{z}) \end{array}\right].$$

Lemma 2.10.4. *The* PKE *and* MPKE *assumptions are equivalent.*

Proof. That MPKE implies PKE follows because MPKE is a generalization of PKE. For the reverse direction, PKE implies MPKE because a successful adversary $\mathcal{A}$ against MPKE can be used to construct a successful adversary $\mathcal{B}$ against PKE that projects the output of $\mathcal{A}$ to one of the "lucky" entries. In more detail, let $\mathcal{B}_i(\langle \mathsf{group} \rangle, \mathbf{\Sigma}, \mathbb{z})$ be the adversary that returns the i-th commitment output by $\mathcal{A}(\langle \mathsf{group} \rangle, \mathbf{\Sigma}, \mathbb{z})$. Let $\mathcal{E}_{\mathcal{B}_i}$ be any PKE extractor corresponding to $\mathcal{B}_i$. Consider the following MPKE extractor $\mathcal{E}_{\mathcal{A}}$ for $\mathcal{A}$: given $(\langle \mathsf{group} \rangle, \mathbf{\Sigma}, \mathbb{z})$, compute $[a_{i,j}]_{j=0}^{d} \leftarrow \mathcal{E}_{\mathcal{B}_i}(\langle \mathsf{group} \rangle, \mathbf{\Sigma}, \mathbb{z})$ for each $i \in [n]$, and output $[[a_{i,j}]_{j=0}^{d}]_{i=1}^{n}$. Observe that $\mathcal{E}_{\mathcal{A}}$ can fail only if at least one of the PKE extractors $\mathcal{E}_{\mathcal{B}_i}$ fails. Thus, if $\mathcal{E}_{\mathcal{A}}$ fails with non-negligible probability $\mu(\lambda)$, then by averaging at least one $\mathcal{E}_{\mathcal{B}_i}$ fails with non-negligible probability $\mu(\lambda)/n$, contradicting the fact that assumption PKE holds. We conclude that MPKE holds if PKE holds. □

Assumption 5. *The* dMPKE **assumption** *states that for every efficient adversary $\mathcal{A}$ and degree bound $d \in \mathbb{N}$ there exists an efficient extractor $\mathcal{E}$ such that for every* benign *auxiliary input distribution $\mathcal{Z}$, the following probability is negligible in λ:*

$$\Pr\left[\begin{array}{c} \exists\, i \in [n] \text{ such that} \\ \\ G_{i,1} = \alpha G_{i,0} \\ \wedge \\ G_{i,0} \neq \sum_{j=0}^{d} a_{i,j}\beta^j G + \sum_{j=0}^{d} (b_{i,j}\gamma\beta^j) G \end{array} \middle| \begin{array}{r} \langle \mathsf{group} \rangle \leftarrow \mathsf{SampleGrp}(1^\lambda) \\ \mathbb{z} \leftarrow \mathcal{Z}(\langle \mathsf{group} \rangle) \\ \alpha, \beta, \gamma \leftarrow \mathbb{F}_q \\ \mathbf{\Sigma} \leftarrow \{\{\beta^i G, \alpha\beta^i G, \gamma\beta^i G, \alpha\gamma\beta^i G\}_{i=0}^{d}, \alpha H, \beta H\} \\ [(G_{i,0}, G_{i,1})]_{i=1}^{n} \leftarrow \mathcal{A}(\langle \mathsf{group} \rangle, \mathbf{\Sigma}, \mathbb{z}) \\ [[(a_{i,j}, b_{i,j})]_{j=0}^{d}]_{i=1}^{n} \leftarrow \mathcal{E}(\langle \mathsf{group} \rangle, \mathbf{\Sigma}, \mathbb{z}) \end{array}\right].$$

Lemma 2.10.5. *The* dPKE *and* dMPKE *assumptions are equivalent.*

Proof. Follows via straightforward modifications to the proof of Lemma 2.10.4. □

Remark 2.10.6 (dMPKE*)**.** For technical reasons, our proof of hiding (Section 2.11.2.3) for the hiding variant of our polynomial commitment scheme relies on the dMPKE* assumption, which is a variant of dMPKE where γ is sampled not from $\mathbb{F}_q$, but rather from $\mathbb{F}_q^\star$. dMPKE implies dMPKE* via a straightforward reduction: given an adversary $\mathcal{A}$ against dMPKE*, one can construct an adversary $\mathcal{B}$ against dMPKE that, on input $(\langle \mathsf{group} \rangle, \mathbf{\Sigma}, \mathbb{z})$, aborts if $\gamma G = G$, and outputs $\mathcal{A}(\langle \mathsf{group} \rangle, \mathbf{\Sigma}, \mathbb{z})$ otherwise. Because $\gamma = 0$ with probability at most $1/q$, $\mathcal{B}$ and $\mathcal{A}$ (and thus $\mathcal{E}_{\mathcal{B}}$ and $\mathcal{E}_{\mathcal{A}}$) differ in success probability by only $1/q$.

2.10.4 Algebraic group model

In order to achieve additional efficiency, we also construct polynomial commitment schemes in the Algebraic Group Model (AGM) [FKL18], which replaces specific knowledge assumptions (such as Power Knowledge of Exponent assumptions). In the AGM, all algorithms are modeled as *algebraic*, which means that whenever an algorithm outputs a group element G, the algorithm must also output an "explanation" of G in terms of the group elements that it has seen.

Definition 2.10.7 (algebraic algorithm)**.** *Let $\mathbb{G}$ be a cyclic group of prime order q and $\mathcal{A}_{\mathsf{alg}}$ a probabilistic algorithm run on initial inputs including description $\langle \mathsf{group} \rangle$ of $\mathbb{G}$. During its execution $\mathcal{A}_{\mathsf{alg}}$ may interact with oracles or other parties and receive further inputs including obliviously sampled group elements (which it cannot sample directly[12]). Let $\boldsymbol{L} \in \mathbb{G}^n$ be the list of all group elements $\mathcal{A}_{\mathsf{alg}}$ has been given so far such that all other inputs it has received do not depend in any way on group elements[13]. We call $\mathcal{A}_{\mathsf{alg}}$* algebraic *if whenever it outputs a group element $G \in \mathbb{G}$ it also outputs a vector $\boldsymbol{a} = [a_i]_{i=1}^n \in \mathbb{F}_q^n$ such that $G = \sum_{i=1}^n a_i L_i$. The coefficients $\boldsymbol{a}$ are called the "representation" of G with respect to $\boldsymbol{L}$, denoted $G := \langle \boldsymbol{a}, \boldsymbol{L} \rangle$.*

Remark 2.10.8 (AGM vs. GGM)**.** The Algebraic Group Model (AGM) [FKL18] is weaker than the Generic Group Model (GGM) [Sho97; Mau05] but is stronger than the plain model. Indeed, every generic algorithm is an algebraic algorithm [PV05], and so anything proved secure in the AGM is also secure in the GGM. On the other hand, the AGM captures non-generic algorithms that exploit the representation of group elements. For example, index-calculus and some factoring attacks fall outside the class of generic algorithms and apply only over groups in which the elements are represented as integers. Furthermore, there exist (pathological) algebraic-but-not-generic algorithms that can be used to construct schemes that are secure in the GGM, but are insecure in the standard and algebraic group models [Den02]. At present, it is not known if such a scheme could be constructed to illustrate a similar gap between the AGM and the standard model.

To analyze the hardness of an assumption in the GGM one must explicitly augment the model by any functionality offered by the structure of the group, e.g., providing a pairing oracle $\mathcal{A}^{e(\cdot,\cdot)}$. However, in the AGM, the adversary has direct access to e (and thus to its description). Though it is widely believe that e provides no additional information about the elements of $\mathbb{G}$, the AGM captures a hypothetical exploit without needing to explicitly model it and considers the relation between two problems instead of their *individual hardness*. This means that if one can reduce problem A to problem G in the AGM and A is conjectured to remain hard with respect to algebraic algorithms, even when given e, then G also remains hard. No similar statement can be inferred in the GGM.

[12] Outputting obliviously sampled group elements (with unknown representation) is forbidden in the AGM. Instead, $\mathcal{A}_{\mathsf{alg}}$ must obliviously sample elements through an additional oracle $\mathcal{O}()$ such that they are by definition added to the list $\boldsymbol{L}$. Simulating $\mathcal{O}()$ to an algebraic algorithm during a reduction is straightforward and always possible. Integrating the ROM and AGM indeed works for this reason that any outputs from random oracles are added to the list $\boldsymbol{L}$.

[13] The restriction that all inputs to algebraic algorithms that are *not* group elements must not depend on group elements helps to avoid pathological cases. For example, the algorithm that on input "$G\|0$" (which is not a group element), outputs group element G cannot explain G in terms of previously seen group elements.

2.10.5 The effect of powers on security

The SDH assumption and PKE assumption rely (in particular) on the hardness of computing the discrete logarithm β when given the generator $G \in \mathbb{G}_1$ and challenge $\beta G \in \mathbb{G}_1$. In fact, in both cases the adversary is also given elements of the form $\beta^i G$, which can have a small, but noticeable, impact on concrete security. There are generic algorithms [BG04; KKM07; Che10] that, for any power i such that $i \mid (q-1)$ where q is the prime order of $\mathbb{G}_1$, compute the secret β in time $O(\sqrt{q/i} + \sqrt{i})$, improving on the usual $O(\sqrt{q})$-time algorithm. This "polynomial speedup" should be taken into account in practice.

Moreover, while our construction of polynomial commitment schemes in Section 2.11 does not use powers of β in $\mathbb{G}_2$, other schemes that wish to share the same SRS might. Hence it is natural to discuss whether our construction in Section 2.11 remains secure even in the presence of elements of the form $\beta^i H$. Our security reduction to the SDH and PKE assumptions does not rely on the absence of powers of β in $\mathbb{G}_2$, and in particular can be modified in a straightforward way to obtain a security reduction to variants of the SDH and PKE assumptions that additionally give to the adversary the additional elements in $\mathbb{G}_2$. These variants, while similarly plausible assumptions, provide the adversary with a further polynomial speedup that must also be taken into account in practice. Namely, given the generator $G \in \mathbb{G}_1$, challenge βG, and elements of the form $\beta^i G \in \mathbb{G}_1$ and $\beta^j H \in \mathbb{G}_2$, one can use the pairing to compute $e(G, H)^{\beta^{i+j}} = e(\beta^i G, \beta^j H)$. If $i + j \mid q - 1$, then the generic algorithms mentioned above compute β in time $O(\sqrt{q/(i+j)} + \sqrt{i+j})$.

2.11 Polynomial commitments for a single degree bound

We construct (succinct) polynomial commitment schemes that support a single degree bound chosen at setup time. We temporarily restrict our attention to the case where, in the reveal phase, all polynomials are evaluated at the same evaluation point. (We will relax this restriction in Section 2.13.) This section is organized as follows: in Section 2.11.1 we provide formal definitions, in Section 2.11.2 we give a construction in the plain model under knowledge assumptions, and in Section 2.11.3 we give a more efficient construction in the algebraic group model under standard assumptions. In both cases we provide non-hiding and hiding variants.

2.11.1 Definition

A polynomial commitment scheme over a field family $\mathcal{F}$ for **a single degree bound and a single evaluation point** is a tuple of algorithms $\mathsf{PC_s} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Check})$ with the following syntax.

- $\mathsf{PC_s}.\mathsf{Setup}(1^\lambda, D) \to (\mathsf{ck}, \mathsf{rk})$. On input a security parameter λ (in unary), and a maximum degree bound $D \in \mathbb{N}$, $\mathsf{PC_s}.\mathsf{Setup}$ samples a key pair $(\mathsf{ck}, \mathsf{rk})$. The keys contain the description of a finite field $\mathbb{F} \in \mathcal{F}$.
- $\mathsf{PC_s}.\mathsf{Commit}(\mathsf{ck}, \boldsymbol{p}; \boldsymbol{\omega}) \to \boldsymbol{c}$. On input ck and univariate polynomials $\boldsymbol{p} = [p_i]_{i=1}^n$ over the field $\mathbb{F}$ with $\deg(p_i) \leq D$, $\mathsf{PC_s}.\mathsf{Commit}$ outputs commitments $\boldsymbol{c} = [c_i]_{i=1}^n$ to the polynomials $\boldsymbol{p}$. The randomness $\boldsymbol{\omega} = [\omega_i]_{i=1}^n$ is used if the commitments $\boldsymbol{c}$ are meant to be hiding.
- $\mathsf{PC_s}.\mathsf{Open}(\mathsf{ck}, \boldsymbol{p}, z, \xi; \boldsymbol{\omega}) \to \pi$. On input ck, univariate polynomials $\boldsymbol{p} = [p_i]_{i=1}^n$, evaluation point $z \in \mathbb{F}$, and opening challenge ξ, $\mathsf{PC_s}.\mathsf{Open}$ outputs an evaluation proof π. The randomness $\boldsymbol{\omega}$ must equal the one previously used in $\mathsf{PC_s}.\mathsf{Commit}$.
- $\mathsf{PC_s}.\mathsf{Check}(\mathsf{rk}, \boldsymbol{c}, z, \boldsymbol{v}, \pi, \xi) \in \{0, 1\}$. On input rk, commitments $\boldsymbol{c} = [c_i]_{i=1}^n$, evaluation point $z \in \mathbb{F}$, alleged evaluations $\boldsymbol{v} = [v_i]_{i=1}^n$, evaluation proof π, and opening challenge ξ, $\mathsf{PC_s}.\mathsf{Check}$ outputs 1 if π attests that, for each $i \in [n]$, the polynomial committed in c_i has degree at most D and evaluates to v_i at z.

The polynomial commitment scheme satisfies the completeness and extractability properties defined below. The polynomial commitment scheme is (perfectly) hiding if it also satisfies the hiding property defined below.

Definition 2.11.1 (Completeness). *For every maximum degree bound $D \in \mathbb{N}$ and efficient adversary $\mathcal{A}$ it holds that*

$$\Pr\left[\begin{array}{c} \deg(\boldsymbol{p}) \leq D \\ \Downarrow \\ \mathsf{PC_s}.\mathsf{Check}(\mathsf{rk}, \boldsymbol{c}, z, \boldsymbol{v}, \pi, \xi) = 1 \end{array} \middle| \begin{array}{r} (\mathsf{ck}, \mathsf{rk}) \leftarrow \mathsf{PC_s}.\mathsf{Setup}(1^\lambda, D) \\ (\boldsymbol{p}, z, \xi) \leftarrow \mathcal{A}(\mathsf{ck}, \mathsf{rk}) \\ \boldsymbol{c} \leftarrow \mathsf{PC_s}.\mathsf{Commit}(\mathsf{ck}, \boldsymbol{p}) \\ \boldsymbol{v} \leftarrow \boldsymbol{p}(z) \\ \pi \leftarrow \mathsf{PC_s}.\mathsf{Open}(\mathsf{ck}, \boldsymbol{p}, z, \xi) \end{array}\right] = 1 \ .$$

Definition 2.11.2 (Extractability). *For every maximum degree bound $D \in \mathbb{N}$ and efficient adversary $\mathcal{A}$, there exists an efficient extractor $\mathcal{E}$ such that for every round bound $\mathsf{r} \in \mathbb{N}$, efficient public-coin challenger $\mathcal{C}$, efficient query sampler $\mathcal{Q}$, and efficient adversary $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ the following probability is negligibly close to 1:*

$$\Pr\left[\begin{array}{c|r} \mathsf{PC_s.Check}(\mathsf{rk}, \boldsymbol{c}, z, \boldsymbol{v}, \pi, \xi) = 1 & (\mathsf{ck}, \mathsf{rk}) \leftarrow \mathsf{PC_s.Setup}(1^\lambda, D) \\ \Downarrow & \hline \\ \deg(\boldsymbol{p}) \leq D \text{ and } \boldsymbol{v} = \boldsymbol{p}(z) & \begin{array}{r} \textit{For } i = 1, \ldots, \mathsf{r}\textit{:} \\ \rho_i \leftarrow \mathcal{C}(\mathsf{ck}, \mathsf{rk}, i) \\ \boldsymbol{c}_i \leftarrow \mathcal{A}(\mathsf{ck}, \mathsf{rk}, [\rho_j]_{j=1}^i) \\ \boldsymbol{p}_i \leftarrow \mathcal{E}(\mathsf{ck}, \mathsf{rk}, [\rho_j]_{j=1}^i) \end{array} \\ & \hline \\ & \begin{array}{r} Q \leftarrow \mathcal{Q}(\mathsf{ck}, \mathsf{rk}, [\rho_j]_{j=1}^{\mathsf{r}}) \\ (\boldsymbol{v}, \mathsf{st}) \leftarrow \mathcal{B}_1(\mathsf{ck}, \mathsf{rk}, [\rho_j]_{j=1}^{\mathsf{r}}, Q) \\ \textit{Sample opening challenge } \xi \\ \pi \leftarrow \mathcal{B}_2(\mathsf{st}, \xi) \\ \textit{Set } [c_i]_{i=1}^n := [\boldsymbol{c}_i]_{i=1}^{\mathsf{r}},\ [p_i]_{i=1}^n := [\boldsymbol{p}_i]_{i=1}^{\mathsf{r}},\ [d_i]_{i=1}^n := [\boldsymbol{d}_i]_{i=1}^{\mathsf{r}} \\ \textit{Parse } Q \textit{ as } T \times \{z\} \textit{ for some } T \subseteq [n] \textit{ and } z \in \mathbb{F} \\ \textit{Set } \boldsymbol{c} := [c_i]_{i \in T},\ \boldsymbol{p} := [p_i]_{i \in T},\ \boldsymbol{d} := [d_i]_{i \in T} \end{array} \end{array}\right].$$

Definition 2.11.3 (Succinctness). *A polynomial commitment scheme is* **succinct** *if the size of commitments, the size of evaluation proofs, and the time to check an opening are all independent of the degree of the committed polynomials. That is, $|\boldsymbol{c}| = n \cdot \mathrm{poly}(\lambda)$, $|\pi| = \mathrm{poly}(\lambda)$, and $\mathsf{time}(\mathsf{Check}) = n \cdot \mathrm{poly}(\lambda)$.*

Definition 2.11.4 (Hiding). *There exists a polynomial-time simulator $\mathcal{S} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open})$ such that, for every maximum degree bound $D \in \mathbb{N}$, round bound $\mathsf{r} \in \mathbb{N}$, and (even unbounded) non-uniform adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$, the probability that $b = 1$ in the following two experiments is identical.*

$\mathsf{Real}(1^\lambda, D, \mathcal{A})$*:*

1. $(\mathsf{ck}, \mathsf{rk}) \leftarrow \mathsf{PC_s.Setup}(1^\lambda, D)$.
2. *Letting* $\boldsymbol{c}_0 := \bot$*, for* $i = 1, \ldots, \mathsf{r}$*:*
   a) $(\boldsymbol{p}_i, \mathsf{h}_i) \leftarrow \mathcal{A}_1(\mathsf{ck}, \mathsf{rk}, \boldsymbol{c}_0, \boldsymbol{c}_1, \ldots, \boldsymbol{c}_{i-1})$.
   b) *If* $\mathsf{h}_i = 0$*: sample commitment randomness* $\boldsymbol{\omega}_i$.
   c) *If* $\mathsf{h}_i = 1$*: set randomness* $\boldsymbol{\omega}_i$ *to* $\bot$.
   d) $\boldsymbol{c}_i \leftarrow \mathsf{PC_s.Commit}(\mathsf{ck}, \boldsymbol{p}_i; \boldsymbol{\omega}_i)$.
3. $\boldsymbol{c} := [\boldsymbol{c}_i]_{i=1}^{\mathsf{r}}$, $\boldsymbol{p} := [\boldsymbol{p}_i]_{i=1}^{\mathsf{r}}$, $\boldsymbol{\omega} := [\boldsymbol{\omega}_i]_{i=1}^{\mathsf{r}}$.
4. $([Q_j]_{j=1}^\tau, [\xi_j]_{j=1}^\tau, \mathsf{st}) \leftarrow \mathcal{A}_2(\mathsf{ck}, \mathsf{rk}, \boldsymbol{c})$.
5. *For* $j \in [\tau]$*:*
   $\pi_j \leftarrow \mathsf{PC_s.Open}(\mathsf{ck}, \boldsymbol{p}, Q_j, \xi_j; \boldsymbol{\omega})$.
6. $b \leftarrow \mathcal{A}_3(\mathsf{st}, [\pi_j]_{j=1}^\tau)$.

$\mathsf{Ideal}(1^\lambda, D, \mathcal{A})$*:*

1. $(\mathsf{ck}, \mathsf{rk}, \mathsf{trap}) \leftarrow \mathcal{S}.\mathsf{Setup}(1^\lambda, D)$.
2. *Letting* $\boldsymbol{c}_0 := \bot$*, for* $i = 1, \ldots, \mathsf{r}$*:*
   a) $(\boldsymbol{p}_i, \mathsf{h}_i) \leftarrow \mathcal{A}_1(\mathsf{ck}, \mathsf{rk}, \boldsymbol{c}_0, \boldsymbol{c}_1, \ldots, \boldsymbol{c}_{i-1})$.
   b) *If* $\mathsf{h}_i = 0$*: sample randomness* $\boldsymbol{\omega}_i$ *and compute simulated commitments* $\boldsymbol{c}_i \leftarrow \mathcal{S}.\mathsf{Commit}(\mathsf{trap}, |\boldsymbol{p}_i|; \boldsymbol{\omega}_i)$.
   c) *If* $\mathsf{h}_i = 1$*: set* $\boldsymbol{\omega}_i := \bot$ *and compute (real) commitments* $\boldsymbol{c}_i \leftarrow \mathsf{PC_s.Commit}(\mathsf{ck}, \boldsymbol{p}_i; \boldsymbol{\omega}_i)$.
3. $\boldsymbol{c} := [\boldsymbol{c}_i]_{i=1}^{\mathsf{r}}$, $\boldsymbol{p} := [\boldsymbol{p}_i]_{i=1}^{\mathsf{r}}$, $\boldsymbol{\omega} := [\boldsymbol{\omega}_i]_{i=1}^{\mathsf{r}}$.
4. *Zero out hidden polynomials:* $\boldsymbol{p}' := [\mathsf{h}_i \boldsymbol{p}_i]_{i=1}^{\mathsf{r}}$.
5. $([Q_j]_{j=1}^\tau, [\xi_j]_{j=1}^\tau, \mathsf{st}) \leftarrow \mathcal{A}_2(\mathsf{ck}, \mathsf{rk}, \boldsymbol{c})$.
6. *For* $j \in [\tau]$*:*
   $\pi_j \leftarrow \mathcal{S}.\mathsf{Open}(\mathsf{trap}, \boldsymbol{p}', \boldsymbol{p}(Q_j), Q_j, \xi_j; \boldsymbol{\omega})$.
7. $b \leftarrow \mathcal{A}_3(\mathsf{st}, [\pi_j]_{j=1}^\tau)$.

Above we implicitly assume that $\mathcal{A}_1$ outputs $\mathrm{poly}(\lambda)$ polynomials in each round, and that $\mathcal{A}_2$ outputs $\tau = \mathrm{poly}(\lambda)$ query sets Q_j, so that $\mathsf{PC_s.Commit}$, $\mathsf{PC_s.Open}$, $\mathcal{S}.\mathsf{Commit}$, and $\mathcal{S}.\mathsf{Open}$ are all efficient.

2.11.2 In the plain model

We adapt the polynomial commitment scheme in [KZG10] to use "knowledge commitments", and to support commitments to multiple polynomials. We then prove that the resulting scheme satisfies the definitions in Section 2.11.1 under knowledge assumptions. We discuss both non-hiding and hiding variants of the scheme.

2.11.2.1 Construction

We use notation for bilinear groups introduced in Section 2.10.1. The highlighted text below denotes parts of the construction that are not needed if hiding is not desired. We refer to the non-hiding variant as $\mathsf{nhPC_s}$, and to the perfectly hiding variant as $\mathsf{phPC_s}$.

Setup. On input a security parameter λ (in unary), and a maximum degree bound $D \in \mathbb{N}$, $\mathsf{PC_s}.\mathsf{Setup}$ samples a key pair $(\mathsf{ck}, \mathsf{rk})$ as follows. Sample a bilinear group $\langle \mathsf{group} \rangle \leftarrow \mathsf{SampleGrp}(1^\lambda)$, and parse $\langle \mathsf{group} \rangle$ as a tuple $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, G, H, e)$. Sample random elements $\alpha, \beta, \in \mathbb{F}_q$ and $\gamma \in \mathbb{F}_q^\star$. Then compute the vector

$$\boldsymbol{\Sigma} := \begin{pmatrix} G & \beta G & \beta^2 G & \dots & \beta^D G \\ \alpha G & \alpha\beta G & \alpha\beta^2 G & \dots & \alpha\beta^D G \\ \gamma G & \gamma\beta G & \gamma\beta^2 G & \dots & \gamma\beta^D G \\ \alpha\gamma G & \alpha\gamma\beta G & \alpha\gamma\beta^2 G & \dots & \alpha\gamma\beta^D G \end{pmatrix} \in \mathbb{G}_1^{4D+4} \ .$$

Set $\mathsf{ck} := (\langle \mathsf{group} \rangle, \boldsymbol{\Sigma})$ and $\mathsf{rk} := (D, \langle \mathsf{group} \rangle, \gamma G, \alpha H, \beta H)$, and then output the public parameters $(\mathsf{ck}, \mathsf{rk})$. These public parameters will support polynomials over the field $\mathbb{F}_q$ of degree at most D.

Commit. On input ck, univariate polynomials $\boldsymbol{p} = [p_i]_{i=1}^n$ over $\mathbb{F}_q$ with $\deg(\boldsymbol{p}) \leq D$, and randomness $\boldsymbol{\omega} = [\omega_i]_{i=1}^n$, $\mathsf{PC_s}.\mathsf{Commit}$ outputs commitments $\boldsymbol{c} = [c_i]_{i=1}^n$ that are computed as follows. If for any $p_i \in \boldsymbol{p}$, $\deg(p_i) > D$, abort. For each $i \in [n]$, if ω_i is not $\perp$, then interpret the randomness ω_i as the coefficients of a random univariate polynomial $\bar{p}_i$ of degree $\deg(p_i)$. Otherwise, set $\bar{p}_i$ to be the zero polynomial. For each $i \in [n]$, output $c_i := (U_i, V_i) \in \mathbb{G}_1^2$ where

$$U_i := p_i(\beta)G + \gamma\bar{p}_i(\beta)G \qquad V_i := \alpha(p_i(\beta)G + \gamma\bar{p}_i(\beta)G) \ .$$

Note that p_i and $\bar{p}_i$ have degree at most D, and so the above terms are linear combinations of terms in ck.

Open. On input ck, univariate polynomials $\boldsymbol{p} = [p_i]_{i=1}^n$ over $\mathbb{F}_q$, evaluation point $z \in \mathbb{F}_q$, opening challenge $\xi \in \mathbb{F}_q$, and randomness $\boldsymbol{\omega} = [\omega_i]_{i=1}^n$ (the same randomness used for $\mathsf{PC_s}.\mathsf{Commit}$), $\mathsf{PC_s}.\mathsf{Open}$ outputs an evaluation proof $\pi \in \mathbb{G}_1$ that is computed as follows. If for any $p_i \in \boldsymbol{p}$, $\deg(p_i) > D$, abort. For each $i \in [n]$, if ω_i is not $\perp$, then obtain a random univariate polynomial $\bar{p}_i$ of degree $\deg(p_i)$ from ω_i, otherwise set $\bar{p}_i$ to be the zero polynomial. Then compute the linear combination of polynomials $p(X) := \sum_{i=1}^n \xi^i p_i(X)$ and $\bar{p}(X) := \sum_{i=1}^n \xi^i \bar{p}_i(X)$. Compute witness polynomials $w(X) := \frac{p(X)-p(z)}{X-z}$ and $\bar{w}(X) := \frac{\bar{p}(X)-\bar{p}(z)}{X-z}$. Set $\mathsf{w} := w(\beta)G + \gamma\bar{w}(\beta)G \in \mathbb{G}_1$ and $\bar{v} := \bar{p}(z) \in \mathbb{F}_q$. The evaluation proof is $\pi := (\mathsf{w}, \bar{v})$.

Check. On input rk, commitments $\boldsymbol{c} = [c_i]_{i=1}^n$, evaluation point $z \in \mathbb{F}_q$, alleged evaluations $\boldsymbol{v} = [v_i]_{i=1}^n$, evaluation proof $\pi = (\mathsf{w}, \bar{v})$, and opening challenge $\xi \in \mathbb{F}_q$, $\mathsf{PC_s}.\mathsf{Check}$ proceeds as follows. Parse each commitment c_i as a tuple $(U_i, V_i) \in \mathbb{G}_1^2$. Compute the two linear combinations

$$U := \sum_{i=1}^n \xi^i U_i \quad \text{and} \quad V := \sum_{i=1}^n \xi^i V_i \ ,$$

and ensure that the commitment (U, V) is extractable by checking that $e(U, \alpha H) = e(V, H)$. Then compute the linear combination of evaluations $v := \sum_{i=1}^n \xi^i v_i$ and check the evaluation proof via the equality $e(U - vG - \bar{v}\gamma G, H) = e(\mathsf{w}, \beta H - zH)$.

Lemma 2.11.5. *The scheme $\mathsf{PC_s}$ constructed above achieves completeness (Definition 2.11.1).*

Proof. Fix any maximum degree bound D and efficient adversary $\mathcal{A}$. Let $(\mathsf{ck}, \mathsf{rk})$ be any key pair output by the algorithm $\mathsf{PC_s}.\mathsf{Setup}(1^\lambda, D)$ constructed above. The keys contain a description $\langle \mathsf{group} \rangle$ of a bilinear group of some prime order q, which in particular induces a field $\mathbb{F}_q$.

Let $\mathcal{A}(\mathsf{ck}, \mathsf{rk})$ select polynomials $\boldsymbol{p} = [p_i]_{i=1}^n$ over $\mathbb{F}_q$, location $z \in \mathbb{F}_q$, and opening challenge $\xi \in \mathbb{F}_q$. We only need to consider adversaries $\mathcal{A}$ that make choices for which $\deg(\boldsymbol{p}) \leq D$. Now consider commitments $\boldsymbol{c} = [c_i]_{i=1}^n$ and evaluation proof π that are all computed according to the construction above.

We need to show that, for the correct evaluations $\boldsymbol{v} := \boldsymbol{p}(z)$,

$$\mathsf{PC_s}.\mathsf{Check}(\mathsf{rk}, \boldsymbol{c}, z, \boldsymbol{v}, \pi, \xi) = 1 \ .$$

This amounts to arguing that the two pairing equations are satisfied.

For the first pairing equation, note that the pair (U_i, V_i) has the property that the second element is the first element multiplied by the secret scalar α. This is also true about the pair (U, V) obtained by taking the linear combination determined by ξ, as the following computation shows:

$$\begin{aligned} U &= \sum_{i=1}^n \xi^i U_i = \sum_{i=1}^n (\xi^i p_i(\beta) G + \gamma \bar{p}_i(\beta) G) \ , \\ V &= \sum_{i=1}^n \xi^i V_i = \alpha \sum_{i=1}^n (\xi^i p_i(\beta) G + \gamma \bar{p}_i(\beta) G) \ . \end{aligned}$$

We conclude that $V = \alpha U$, and so the check $e(U, \alpha H) = e(V, H)$ passes.

For the second pairing equation, note that in the evaluation proof $\pi = (\mathsf{w}, \bar{v})$, w equals the element $\mathsf{w} := \frac{p(X) - p(z) + \gamma(\bar{p}(X) - \bar{p}(z))}{X - z} G$ where $p(X) := \sum_{i=1}^n \xi^i p_i(X)$ and $\bar{p}(X) := \sum_{i=1}^n \xi^i \bar{p}_i(X)$. Also note that the value v computed by $\mathsf{PC_s}.\mathsf{Check}$ is the evaluation of p at z. Therefore,

$$\begin{aligned} e(U - vG - \gamma \bar{v} G, H) &= e(((p(\beta) - v) + \gamma(\bar{p}(\beta) - \bar{v}))G, H) \\ &= e(\tfrac{p(\beta) - p(z) + \gamma(\bar{p}(\beta) - \bar{p}(z))}{\beta - z} G, (\beta - z) H) \\ &= e((w(\beta) + \gamma \bar{w}(\beta))G, \beta H - zH) \\ &= e(\mathsf{w}, \beta H - zH) \ . \end{aligned}$$

We conclude that the second pairing equation also holds. □

Lemma 2.11.6. *The scheme* PC_s *constructed above achieves succinctness (Definition 2.11.3).*

Proof. For a list of n polynomials, the scheme PC_s requires $2n$ $\mathbb{G}_1$ elements for the commitment and one $\mathbb{G}_1$ element and one $\mathbb{F}_q$ element for the evaluation proof, while the time to check this proof requires two variable-base multi-scalar multiplications of size n and four pairings. □

2.11.2.2 Extractability

Theorem 2.11.7. *If the bilinear group sampler* SampleGrp *satisfies the* SDH *and* dPKE *assumptions,* nhPC_s *and* phPC_s *constructed in Section 2.11.2.1 achieve extractability (Definition 2.11.2).*

First in Lemma 2.11.9 we argue that nhPC_s and phPC_s satisfy *evaluation binding*, a property stating that for any point $z \in \mathbb{F}_q$ and commitments $\boldsymbol{c} = [c_i]_{i=1}^n$, no efficient adversary can produce valid proofs that open $\boldsymbol{c}$ to different lists of values at z. Then in Lemma 2.11.10 we build on this fact to argue that nhPC_s and phPC_s achieve extractability.

Definition 2.11.8. PC_s *satisfies* **evaluation binding** *if for every maximum degree bound* $D \in \mathbb{N}$ *and efficient adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *the following probability is negligible in the security parameter* λ*:*

$$\Pr\left[\begin{array}{c} \boldsymbol{v} \neq \boldsymbol{v'} \\ \wedge \\ \mathsf{PC}_\mathsf{s}.\mathsf{Check}(\mathsf{rk}, \boldsymbol{c}, z, \boldsymbol{v}, \pi, \xi) = 1 \\ \wedge \\ \mathsf{PC}_\mathsf{s}.\mathsf{Check}(\mathsf{rk}, \boldsymbol{c}, z, \boldsymbol{v'}, \pi', \xi) = 1 \end{array} \middle| \begin{array}{r} (\mathsf{ck}, \mathsf{rk}) \leftarrow \mathsf{PC}_\mathsf{s}.\mathsf{Setup}(1^\lambda, D) \\ (\boldsymbol{c}, z, \boldsymbol{v}, \boldsymbol{v'}, \mathsf{st}) \leftarrow \mathcal{A}_1(\mathsf{ck}, \mathsf{rk}) \\ \textit{Sample opening challenge } \xi \\ (\pi, \pi') \leftarrow \mathcal{A}_2(\mathsf{st}, \xi) \end{array}\right].$$

Lemma 2.11.9. *If the bilinear group sampler* SampleGrp *satisfies the* SDH *assumption (Assumption 1),* nhPC_s *and* phPC_s *constructed in Section 2.11.2.1 achieve evaluation binding (Definition 2.11.8).*

Proof. Suppose for contradiction that there exists a maximum degree bound D and an efficient adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ that breaks evaluation binding with non-negligible probability. We show that either $\mathcal{A}$ can be used to break DL with non-negligible probability or that we can use $\mathcal{A}$ to construct an efficient adversary $\mathcal{B}$ that breaks SDH with non-negligible probability. Since the SDH assumption implies the DL assumption, in either case we obtain a contradiction that SDH holds with respect to SampleGrp. We define $\mathcal{B}$ as follows.

$\mathcal{B}(\langle \mathsf{group} \rangle, \boldsymbol{\Sigma})$:
1. Parse $\boldsymbol{\Sigma}$ as $\{\{\beta^i G\}_{i=0}^D, \beta H\}$.
2. Randomly sample $\alpha \leftarrow \mathbb{F}_q, \gamma \in \mathbb{F}_q^\star$ and set
   $\mathsf{ck} := (\langle \mathsf{group} \rangle, \{\beta^i G, \alpha\beta^i G, \gamma\beta^i G, \alpha\gamma\beta^i G\}_{i=0}^D)$,
   $\mathsf{rk} := (\langle \mathsf{group} \rangle, \alpha H, \beta H)$.
3. Compute $\left(\boldsymbol{c}, z, \boldsymbol{v}, \boldsymbol{v'}, \mathsf{st}\right) \leftarrow \mathcal{A}_1(\mathsf{ck}, \mathsf{rk})$.
4. Sample random opening challenge $\xi \in \mathbb{F}_q$.
5. Compute $(\pi, \pi') \leftarrow \mathcal{A}_2(\mathsf{st}, \xi)$.
6. Parse (π, π') as $((\mathsf{w}, \bar{v}), (\mathsf{w}', \bar{v}'))$, $\boldsymbol{v}$ as $[v_i]_{i=1}^n$, and $\boldsymbol{v'}$ as $[v'_i]_{i=1}^n$.
7. Compute $v := \sum_{i=1}^n \xi^i v_i$ and $v' := \sum_{i=1}^n \xi^i v'_i$.
8. If $zG = \beta G$ (i.e., $z = \beta$):
   choose a from $\mathbb{F}_q \setminus \{z\}$, and output $\left(a, \frac{1}{z+a} G\right)$, breaking SDH.
9. Else if $(zG \neq \beta G) \wedge (\mathsf{w} \neq \mathsf{w}')$:
   output $\left(-z, \frac{1}{v'-v+\gamma(\bar{v}'-\bar{v})}(\mathsf{w} - \mathsf{w}')\right)$, breaking SDH.
10. Else abort.

First, we show that if either the predicate in Step 8 or the predicate in Step 9 is satisfied, then $\mathcal{B}$ does in fact break SDH. Next, we show that one of these predicates is satisfied with non-negligible probability whenever $\mathcal{A}$ breaks evaluation binding. We do this by showing that if $\mathcal{B}$ aborts but $\mathcal{A}$ still succeeds, then $\mathcal{A}$ can be used to solve the discrete logarithm problem in SampleGrp with non-negligible probability.

$\mathcal{B}$ succeeds if predicates are satisfied. If $\mathcal{A}$ outputs $z = \beta$, then $\mathcal{B}$ can construct an arbitrary solution to the SDH problem. If on the other hand $(\beta \neq z) \wedge (\mathsf{w} \neq \mathsf{w}')$, then if $\mathcal{A}$ breaks evaluation binding, by construction of PC_s.Check the following equations must hold:

$$e(U - vG - \gamma\bar{v}G, H) = e(\mathsf{w}, \beta H - zH) \ , \tag{2.12}$$
$$e(U - v'G - \gamma\bar{v}'G, H) = e(\mathsf{w}', \beta H - zH) \ . \tag{2.13}$$

Then, $\mathsf{w} \neq \mathsf{w}'$ and $\beta \neq z$ together imply that $v' - v + \gamma(\bar{v}' - \bar{v}) \neq 0$. The above equations can then be rewritten as

$$\frac{1}{v' - v + \gamma(\bar{v}' - \bar{v})}(\mathsf{w} - \mathsf{w}') = \frac{1}{\beta - z} G \ ,$$

making $\left(-z, \frac{1}{v'-v+\gamma(\bar{v}'-\bar{v})}(\mathsf{w} - \mathsf{w}')\right)$ a pair that breaks the SDH assumption.

Probability that predicates are satisfied. We analyze the probability with which $\mathcal{B}$ aborts by considering the probability that the predicates are not satisfied, i.e., $(\beta \neq z) \wedge (\mathsf{w} = \mathsf{w}')$. We break this case down into the following two disjoint subcases:

- **Case 1:** $\bar{v} \neq \bar{v}'$. In this case, Equations (2.12) and (2.13) imply that $v' - v + \gamma(\bar{v}' - \bar{v}) = 0$. We can rewrite this equation to compute the secret discrete logarithm $\gamma = \frac{v - v'}{\bar{v}' - \bar{v}}$.
- **Case 2:** $\bar{v} = \bar{v}'$. In this case, it must hold that $v = v'$. Since $\boldsymbol{v} \neq \boldsymbol{v'}$, this occurs with probability at most $\frac{n}{q}$.

Hence, we conclude that if $(\beta \neq z) \wedge (\mathsf{w} = \mathsf{w}')$ with non-negligible probability and $\mathcal{A}$ still succeeds, then $\mathcal{A}$ can be used to break DL with non-negligible probability, which cannot occur if SDH is hard for SampleGrp.

Thus, if $\mathcal{A}$ succeeds, then with non-negligible probability either $\beta = z$, or $(\beta \neq z) \wedge (\mathsf{w} \neq \mathsf{w}')$, which in turn implies that $\mathcal{B}$ breaks SDH, contradicting our assumption. □

Lemma 2.11.10. *If* PC_s *constructed in Section 2.11.2.1 achieves evaluation binding (Definition 2.11.8), and if the bilinear group sampler* SampleGrp *satisfies the* dPKE *assumption (Assumption 2 and Assumption 3), then* PC_s *achieves extractability (Definition 2.11.2).*

Proof. The dPKE assumption implies the dMPKE* assumption, which considers the case where the adversary outputs multiple knowledge commitments; see Section 2.10.3.1. Below we rely on dMPKE*.

Fix a maximum degree bound D and an efficient adversary $\mathcal{A}$ against extractability. We use $\mathcal{A}$ to define the adversary $\mathcal{D}$ below, which is against dMPKE.

$\mathcal{D}(\langle\mathsf{group}\rangle, \boldsymbol{\Sigma}, \mathbb{z})$:
1. Parse instance $\boldsymbol{\Sigma}$ as $\{\boldsymbol{\Sigma}_{\mathsf{PC}} = \{\beta^i G, \alpha\beta^i G, \gamma\beta^i G, \alpha\gamma\beta^i G\}_{i=0}^{D}, \alpha H, \beta H\}$.
2. Parse the auxiliary input $\mathbb{z}$ as randomness $[\rho_j]_{j=1}^{i}$.
3. Construct $\mathsf{ck} := (\langle\mathsf{group}\rangle, \boldsymbol{\Sigma}_{\mathsf{PC}})$.
4. Construct $\mathsf{rk} := (\langle\mathsf{group}\rangle, \alpha H, \beta H)$.
5. Let $\boldsymbol{c} \leftarrow \mathcal{A}(\mathsf{ck}, \mathsf{rk}, [\rho_j]_{j=1}^{i})$.
6. Output $\boldsymbol{c}$.

By assumption there exists a dMPKE* extractor $\mathcal{E}_\mathcal{D}$ against $\mathcal{D}$ that works with overwhelming probability. We use $\mathcal{E}_\mathcal{D}$ to construct an extractor $\mathcal{E}_\mathcal{A}$ for the polynomial commitment scheme. The rest of this proof will argue that $\mathcal{E}_\mathcal{A}$ defined below succeeds with overwhelming probability. For each round $i \in [\mathsf{r}]$, $\mathcal{E}_\mathcal{A}$ proceeds as follows. We denote by k_i the number of polynomials output by $\mathcal{A}$ in round i.

$\mathcal{E}_\mathcal{A}(\mathsf{ck}, \mathsf{rk}, [\rho_j]_{j=1}^{i})$:
1. Parse ck as $(\langle\mathsf{group}\rangle, \boldsymbol{\Sigma}_{\mathsf{PC}})$ and rk as $(\langle\mathsf{group}\rangle, \alpha H, \beta H)$.
2. Construct dMPKE instance $\boldsymbol{\Sigma} := \{\boldsymbol{\Sigma}_{\mathsf{PC}}, \alpha H, \beta H\}$.
3. Construct auxiliary input $\mathbb{z} := [\rho_j]_{j=1}^{i}$.
4. Run the dMPKE extractor: $([\boldsymbol{a}_j]_{j=1}^{\mathsf{k}_i}, [\boldsymbol{b}_j]_{j=1}^{\mathsf{k}_i}) \leftarrow \mathcal{E}_\mathcal{D}(\langle\mathsf{group}\rangle, \boldsymbol{\Sigma}, \mathbb{z})$.
5. Set $\boldsymbol{X} := (1, X, \ldots, X^D)$.
6. For each j in $[\mathsf{k}_i]$, define polynomials $p_j(X) := \langle \boldsymbol{a}_j, \boldsymbol{X} \rangle \in \mathbb{F}_q[X]$ and $\bar{p}_j(X) := \langle \boldsymbol{b}_j, \boldsymbol{X} \rangle \in \mathbb{F}_q[X]$.
7. For each j in $[\mathsf{k}_i]$, let the randomness ω_j be the coefficients of $\bar{p}_j$.
8. Output the polynomials $\boldsymbol{p} := [p_j]_{j=1}^{\mathsf{k}_i}$ and the randomness $\boldsymbol{\omega} := [\omega_j]_{j=1}^{\mathsf{k}_i}$.

For the purpose of our proof, we additionally let the above extractor output the randomness $\boldsymbol{\omega}$.

Suppose for contradiction that the extractor $\mathcal{E}_\mathcal{A}$ fails with some non-negligible probability, for a choice of round bound $\mathsf{r} \in \mathbb{N}$, efficient public-coin challenger $\mathcal{C}$, efficient query sampler $\mathcal{Q}$, and

efficient adversary $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$. We show this implies that either $\mathcal{D}$ succeeds with non-negligible probability (contradicting our dPKE assumption), or that we can construct an adversary $\mathcal{A}'$ that contradicts Lemma 2.11.9. In more detail, the extractor $\mathcal{E}_{\mathcal{A}}$ may fail due to (at least) one of two reasons.

(1) *Incorrect polynomial or randomness:* there exists an $i \in T$ such that polynomial p_i or random polynomial $\bar{p}_i$ does not match its commitment.

(2) *Incorrect evaluation:* for every $i \in T$, the extracted polynomial p_i and corresponding random polynomial $\bar{p}_i$ match their commitments, but claimed evaluation for one such p_i is incorrect.

If $\mathcal{E}_{\mathcal{A}}$ fails with non-negligible probability, then at least one of these cases occurs with non-negligible probability. We analyze each case, and argue that this cannot be (or else we contradict our assumptions).

(1) Incorrect polynomial or randomness. Informally, this case occurs with negligible probability if the dMPKE* assumption holds for SampleGrp. In more detail, we have to demonstrate that since PC_s.Check accepts, every knowledge commitment in $\boldsymbol{c}$ is "extractable", i.e., for each $c \in \boldsymbol{c}$, $c = (U, V)$ satisfies $\alpha U = V$ with overwhelming probability.

We do this as follows. Since PC_s.Check accepts, we know that $e(U, \alpha H) = e(V, H)$, where $U := \sum_{i \in T} \xi^i U_i$ and $V := \sum_{i \in T} \xi^i V_i$ are linear combinations of the input commitments. In this case, we have that $\alpha U_i \neq V_i$ for some i with probability at most $|T|/q$ over random choice of ξ. Thus with probability $1 - |T|/q$, each knowledge commitment satisfies the equality $V_i = \alpha U_i$, and is thus extractable. In this case we have that

$$U_i \neq \sum_{k=0}^{D} (a_{i,k} \beta^k G + b_{i,k} \gamma \beta^k G) \ ,$$

for some $i \in T$ only if $\mathcal{E}_{\mathcal{D}}$ has failed, and by assumption this only happens with negligible probability.

(2) Incorrect evaluation. We show that this case occurs with negligible probability if evaluation binding holds for PC_s. If $\mathcal{E}_{\mathcal{A}}$ outputs polynomials that do not match the claimed evaluations with non-negligible probability $\mu(\lambda)$, then we can use $(\mathcal{A}, \mathcal{B}_1, \mathcal{B}_2)$, the public-coin challenger $\mathcal{C}$ and the query sampler $\mathcal{Q}$ to construct the following adversary $\mathcal{A}' = (\mathcal{A}'_1, \mathcal{A}'_2)$ that succeeds in breaking evaluation binding (Definition 2.11.8) with the same non-negligible probability $\mu(\lambda)$.

$\mathcal{A}'_1(\mathsf{ck}, \mathsf{rk})$:
1. For $i = 1, \ldots, \mathsf{r}$:
   a) Obtain challenge: $\rho_i \leftarrow \mathcal{C}(\mathsf{ck}, \mathsf{rk}, i)$.
   b) Obtain commitments: $\boldsymbol{c}_i \leftarrow \mathcal{A}(\mathsf{ck}, \mathsf{rk}, [\rho_j]_{j=1}^{i})$.
   c) Extract polynomials and randomness: $(\boldsymbol{p}_i, \boldsymbol{\omega}_i) \leftarrow \mathcal{E}_{\mathcal{A}}(\mathsf{ck}, \mathsf{rk}, [\rho_j]_{j=1}^{i})$.
2. Sample query set: $Q \leftarrow \mathcal{Q}(\mathsf{ck}, \mathsf{rk}, [\rho_j]_{j=1}^{i})$.
3. Set $[c_i]_{i=1}^{n} := [\boldsymbol{c}_i]_{i=1}^{\mathsf{r}}$, $[p_i]_{i=1}^{n} := [\boldsymbol{p}_i]_{i=1}^{\mathsf{r}}$, and $[\omega_i]_{i=1}^{n} := [\boldsymbol{\omega}_i]_{i=1}^{\mathsf{r}}$.
4. Parse Q as $T \times \{z\}$ for some $T \subseteq [n]$ and $z \in \mathbb{F}$.
5. Set $\boldsymbol{c} := [c_i]_{i \in T}$, $\boldsymbol{p} := [p_i]_{i \in T}$, and $\boldsymbol{\omega} := [\omega_i]_{i \in T}$.
6. $(\boldsymbol{v}, \mathsf{st}_{\mathcal{B}}) \leftarrow \mathcal{B}_1(\mathsf{ck}, \mathsf{rk}, [\rho_j]_{j=1}^{\mathsf{k}}, Q)$.
7. Compute alternate evaluations $\boldsymbol{v'} := \boldsymbol{p}(z)$.
8. Set $\mathsf{st} := (\mathsf{ck}, \mathsf{rk}, \boldsymbol{p}, z, \boldsymbol{\omega}, \mathsf{st}_{\mathcal{B}})$.
9. Output $\big(\boldsymbol{c}, z, \boldsymbol{v}, \boldsymbol{v'}, \mathsf{st}\big)$.

$\mathcal{A}'_2(\mathsf{st}, \xi)$:
1. Parse st as $(\mathsf{ck}, \mathsf{rk}, \boldsymbol{p}, z, \boldsymbol{\omega}, \mathsf{st}_{\mathcal{B}})$.
2. Obtain proof of evaluation: $\pi \leftarrow \mathcal{B}_2(\mathsf{st}_{\mathcal{B}}, \xi)$.
3. Compute alternate proof: $\pi' \leftarrow \mathsf{PC_s}.\mathsf{Open}(\mathsf{ck}, \boldsymbol{p}, z, \xi; \boldsymbol{\omega})$.
4. Output (π, π').

Since the extractor successfully extracts each polynomial and the randomness, and since $\mathsf{PC_s}$ satisfies perfect completeness, $\mathcal{A}'_2$ should be able to produce an alternate valid proof π' that is also accepted by $\mathsf{PC_s}.\mathsf{Check}$. Thus, if $\mathcal{A}$ breaks polynomial extractability with non-negligible probability by producing valid proofs for incorrect evaluations, then $\mathcal{A}' = (\mathcal{A}'_1, \mathcal{A}'_2)$ breaks evaluation binding for $\mathsf{PC_s}$ with non-negligible probability, which contradicts our assumption. □

2.11.2.3 Hiding

Theorem 2.11.11. $\mathsf{PC_s}$ *constructed in Section 2.11.2.1 achieves hiding (Definition 2.11.4).*

Proof. We describe a polynomial-time simulator $\mathcal{S}$ such that, for every maximum degree bound D and efficient adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$, the adversary $\mathcal{A}$ cannot distinguish the real world and ideal world experiments.

We leverage the fact that by knowing the "trapdoor" the simulator $\mathcal{S}$ can create the evaluation proof for arbitrary values with respect to the commitment. We build our simulator $\mathcal{S}$ as follows:

$\mathcal{S}.\mathsf{Setup}(1^{\lambda}, D)$:
1. Run $\mathsf{PC_s}.\mathsf{Setup}(1^{\lambda}, D)$, additionally defining $\mathsf{trap} := (\mathsf{ck}, \mathsf{rk}, \beta, \gamma)$.
2. Output $(\mathsf{ck}, \mathsf{rk}, \mathsf{trap})$.

$\mathcal{S}.\mathsf{Commit}(\mathsf{trap}, \mathsf{k}; \boldsymbol{\omega})$:

1. Parse $\boldsymbol{\omega}$ as $[\omega_i]_{i=1}^{\mathsf{k}}$.
2. For $i = 1, \ldots, \mathsf{k}$:
   a) Obtain the random polynomial $\bar{p}_i(X)$ from ω_i.
   b) Compute $U_i := \bar{p}_i(\beta)\gamma G$ and $V_i := \alpha\bar{p}_i(\beta)\gamma G$.
   c) Set $c_i = (U_i, V_i)$.
3. Output $\boldsymbol{c} := [c_i]_{i=1}^{\mathsf{k}}$.

$\mathcal{S}.\mathsf{Open}(\mathsf{trap}, \boldsymbol{p}, \boldsymbol{v}, Q, \xi; \boldsymbol{\omega})$:

1. Parse $\boldsymbol{p} := [p_i]_{i=1}^n, \boldsymbol{v} := [v_i]_{i=1}^n$, and $\boldsymbol{\omega} := [\omega_i]_{i=1}^n$.
2. Parse query set Q as $T \times \{z\}$ for some $T \subseteq [n]$ and $z \in \mathbb{F}_q$.
3. For $i \in T$:
   a) If $\omega_i \neq \perp$:
      i. Compute $(U_i, V_i) \leftarrow \mathcal{S}.\mathsf{Commit}(\mathsf{trap}, 1; \omega_i)$.
      ii. Obtain the random polynomial $\bar{p}_i(X)$ from ω_i.
      iii. Set $\tilde{v}_i := \bar{p}_i(z) - \frac{v_i}{\gamma}$.
   b) Else $\omega_i = \perp$:
      i. Compute $(U_i, V_i) \leftarrow \mathsf{PC}_{\mathsf{s}}.\mathsf{Commit}(\mathsf{ck}, p_i; \perp)$.
      ii. Set $\tilde{v}_i := 0$.
4. Compute $\bar{v} := \sum_{i=1}^n \xi^i \tilde{v}_i$, $v := \sum_{i=1}^n \xi^i v_i$, $U := \sum_{i=1}^n \xi^i U_i$.
5. If $z \neq \beta$:
   Compute $\mathsf{w} := \frac{1}{\beta - z} U - \frac{v - \gamma\bar{v}}{\beta - z} G$.
6. Else $z = \beta$:
   Set $\mathsf{w} := 0G$.
7. Output $\pi := (\mathsf{w}, \bar{v})$.

Clearly, $\mathcal{S}$ is polynomial-time. Associated with each p_i output by $\mathcal{A}$ there is an independently and randomly sampled degree D polynomial $\bar{p}_i$ defined by ω_i. We define a polynomial $\bar{p}'_i$ such that in the real world, $\bar{p}'_i := \bar{p}_i$, whereas in the ideal world, if $\mathsf{h}_i = 0$ (and hence $\omega_i \neq \perp$), then $\bar{p}'_i(X) := \bar{p}_i(X) - \frac{p_i(X)}{\gamma}$, and $\bar{p}'_i = 0$ otherwise. Observe that each $\bar{p}'_i$ is of degree D and is independently and randomly distributed if the corresponding polynomial is required to be hiding. It follows that these polynomials are identically distributed in the two worlds. Furthermore, since $\mathcal{S}.\mathsf{Setup}$ uses $\mathsf{PC}_{\mathsf{s}}.\mathsf{Setup}$ to generate $(\mathsf{ck}, \mathsf{rk})$, we see that $(\mathsf{ck}, \mathsf{rk})$ is also identically distributed.

We claim that for each round $i \in [\mathsf{r}]$, upon fixing $(\mathsf{ck}, \mathsf{rk})$ and $\bar{\boldsymbol{p}}'_i$, the resulting $\boldsymbol{c}_i$ are given by a deterministic function in $\boldsymbol{p}_i(\beta)$ and, after fixing all the $\bar{\boldsymbol{p}}'$, for each query point $[z_j]_{j=1}^{\tau}$ the corresponding proof π_j is given by a deterministic function in $(\boldsymbol{p}(z_j), z_j, \xi_j)$. Since these deterministic functions are parametrized by ck, rk, and the $\bar{\boldsymbol{p}}'$, which we have already shown are identically distributed in the two worlds, it follows that the mappings of these functions will likewise be identically distributed, and thus we claim the two worlds are indistinguishable even by unbounded adversaries.

Abusing notation to express group elements (or vectors thereof) as functions, we claim that for commitments $\boldsymbol{c}_i := (\boldsymbol{U}_i, \boldsymbol{V}_i)$ that $\boldsymbol{U}_i(\boldsymbol{p}_i(\beta)) = \boldsymbol{p}_i(\beta)G + \gamma\bar{\boldsymbol{p}}'_i(\beta)G$ and $\boldsymbol{V}_i = \alpha\boldsymbol{U}_i$. Similarly, we

claim the proof elements $\pi_j := (\bar{v}_j, \mathsf{w}_j)$ that

$$\bar{v}_j\left(z_j, \xi_j\right) = \sum_{i=1}^{n} \xi_j^i \cdot \bar{p}_i'(z_j) \ , \quad \mathsf{w}_j\left(z_j\right) = \begin{cases} \frac{1}{\beta - z_j} U - \frac{v - \gamma \bar{v}_j}{\beta - z_j} G & \text{if } z_j \neq \beta \\ 0 & \text{if } z_j = \beta \end{cases} \ ,$$

where

$$U\left(\xi_j\right) = \sum_{i=1}^{n} \xi_j^i \cdot U_i \ , \quad v\left(\boldsymbol{p}(z_j), \xi_j\right) = \sum_{i=1}^{n} \xi_j^i \cdot p_i(z_j) \ .$$

To conclude the proof it now only remains to be shown that the functions above describe the outputs of PC_s and $\mathcal{S}$. We demonstrate this sequentially below.

Indistinguishability of commitments. In the real world we have

$$U_i := p_i(\beta)G + \gamma \bar{p}_i(\beta)G \ , \quad V_i := \alpha(p_i(\beta) + \gamma \bar{p}_i(\beta))G \ ,$$

where since in this world we have defined $\bar{p}' := \bar{p}$ our claim that $U_i(p_i) = p_i(\beta)G + \gamma \bar{p}_i'(\beta)G$ and $V_i = \alpha U_i$ follows immediately. Now considering the ideal world case, from our pseudocode above we have that

$$U_i := \gamma \bar{p}_i(\beta)G \ , \quad V_i := \alpha U_i \ ,$$

where now we have defined $\bar{p}_i'(z) := \bar{p}_i(z) - \frac{p(z)}{\gamma}$. Plugging this in we have

$$p_i(\beta)G + \gamma \bar{p}_i'(\beta)G = p_i(\beta)G + \gamma \left(\bar{p}_i(\beta) - \frac{p(\beta)}{\gamma}\right) G = \gamma \bar{p}_i(\beta)G \ ,$$

and thus may conclude that commitments are indistinguishable with respect to all adversaries.

Indistinguishability of evaluation proofs. In the real world we have $\bar{v}_j = \sum_{i=1}^{n} \xi_j^i \cdot \bar{p}_i(z_j)$, where since $\bar{\boldsymbol{p}}_i' = \bar{\boldsymbol{p}}_i$ we arrive at the expected function. In the ideal world we have that $\bar{v}_j := \sum_{i=1}^{n} \xi_j^i \cdot \tilde{v}_i$, where in the pseudocode above we see that $\tilde{v}_i = \bar{p}_i'(z_j)$. We conclude that the $\bar{\boldsymbol{v}}$ are indistinguishable with respect to all adversaries.

Finally, we consider the w_j. In the real world we have $\mathsf{w}_j := w(\beta)G + \gamma \bar{w}(\beta)G$, where

$$w(X) := \frac{p(X) - p(z)}{X - z} \ , \quad \bar{w}(X) := \frac{\bar{p}(X) - \bar{p}(z)}{X - z} \ .$$

Plugging these values in we obtain

$$\mathsf{w}_j := \frac{p(\beta) - p(z)}{\beta - z} G + \gamma \frac{\bar{p}(\beta) - \bar{p}(z)}{\beta - z} G = \frac{p(\beta) + \gamma \bar{p}(\beta)}{\beta - z} G - \frac{p(z) + \gamma \bar{p}(z)}{\beta - z} G \quad .$$

Recall our expressions for p and $\bar{p}$ are

$$p(X) := \sum_{i=1}^{n} \xi^i p_i(X) \ , \quad \bar{p}(X) := \sum_{i=1}^{n} \xi^i \bar{p}_i(X) \ .$$

Then we have that

$$\frac{p(\beta)+\gamma\bar{p}(\beta)}{\beta-z}G = \frac{1}{\beta-z}U\ , \quad \frac{p(z)+\gamma\bar{p}(z)}{\beta-z}G = \frac{v-\gamma\bar{v}_j}{\beta-z_j}G\ ,$$

with U and v above defined in terms of a polynomial over the U_i and $p_i(z_j)$, respectively, evaluated at ξ_j. We note that w and $\bar{w}$ are not rational functions because $X-z$ always divides $p(X)-p(z)$ for any univariate polynomial p, and that evaluated at $z=\beta$ they are both 0 rather than undefined. Thus we have shown that in the real world each w_j is defined as promised.

In the ideal world, it is easy to see our expression for w_j already has the expected form, as do the corresponding equations used for computing U and v. We conclude that no adversary can distinguish between the two worlds. □

2.11.3 In the algebraic group model

The constructions in Section 2.11.2 require *two* group elements to commit to a polynomial due to their use of knowledge assumptions. In this section we achieve better efficiency (*one* group element per polynomial) by proving extractability in the AGM. Instead of relying on the PKE extractor to extract polynomials from commitment, we simply use the algebraic adversary's own explanations. This makes the extractability proof straightforward.

We proceed as follows. First, in Section 2.11.3.1, we describe how to modify the constructions in Section 2.11.2.1 to rely on the AGM, and then in Section 2.11.3.2, we demonstrate that these modified constructions achieve extractability against algebraic adversaries.

2.11.3.1 Construction

We use notation for bilinear groups introduced in Section 2.10.1 and notation for algebraic algorithms from Definition 2.10.7. The highlighted text below denotes parts of the construction that are not needed if hiding is not desired. Reusing notation from Section 2.11.2.1, we refer to the non-hiding variant as $\mathsf{nhPC_s}$, and the hiding variant as $\mathsf{phPC_s}$ (perfectly-hiding $\mathsf{PC_s}$). At a high level, the construction follows the blueprint of Section 2.11.2.1 closely, but all the terms including α are never generated during setup (and thus never subsequently used). This is because these are precisely the terms used to prove knowledge when relying on PKE.

Setup. On input a security parameter λ (in unary), and a maximum degree bound $D \in \mathbb{N}$, $\mathsf{PC_s.Setup}$ samples public parameters $(\mathsf{ck}, \mathsf{rk})$ as follows. Sample a bilinear group $\langle \mathsf{group} \rangle \leftarrow \mathsf{SampleGrp}(1^\lambda)$, and parse $\langle \mathsf{group} \rangle$ as a tuple $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, G, H, e)$. Sample random elements $\beta, \gamma \in \mathbb{F}_q$. Then compute the vector

$$\mathbf{\Sigma} := \begin{pmatrix} G & \beta G & \beta^2 G & \dots & \beta^D G \\ \gamma G & \gamma\beta G & \gamma\beta^2 G & \dots & \gamma\beta^D G \end{pmatrix} \in \mathbb{G}_1^{2D+2}\ .$$

Set $\mathsf{ck} := (\langle \mathsf{group} \rangle, \mathbf{\Sigma})$ and $\mathsf{rk} := (D, \langle \mathsf{group} \rangle, \gamma G, \beta H)$, and then output the public parameters $(\mathsf{ck}, \mathsf{rk})$. These public parameters will support polynomials over the field $\mathbb{F}_q$ of degree at most D.

Commit. On input ck, univariate polynomials $\boldsymbol{p} := [p_i]_{i=1}^n$ over $\mathbb{F}_q$, and randomness $\boldsymbol{\omega} := [\omega_i]_{i=1}^n$, $\mathsf{PC_s}.\mathsf{Commit}$ outputs commitments $\boldsymbol{c} := [c_i]_{i=1}^n$ that are computed as follows. If for any $p_i \in \boldsymbol{p}$, $\deg(p_i) > D$, abort. Else, for each $i \in [n]$, if ω_i is not $\perp$, then obtain random univariate polynomial $\bar{p}_i$ of degree $\deg(p_i)$ from ω_i, otherwise $\bar{p}_i$ is set to be a zero polynomial. For each $i \in [n]$, output $c_i := p_i(\beta)G + \gamma\bar{p}_i(\beta)G$. Note that because p_i and $\bar{p}_i$ have degree at most D, the above terms are linear combinations of terms in ck.

Open. On input ck, univariate polynomials $\boldsymbol{p} := [p_i]_{i=1}^n$ over $\mathbb{F}_q$, evaluation point $z \in \mathbb{F}_q$, opening challenge $\xi \in \mathbb{F}_q$, and randomness $\boldsymbol{\omega} := [\omega_i]_{i=1}^n$, which is the same randomness used for $\mathsf{PC_s}.\mathsf{Commit}$, $\mathsf{PC_s}.\mathsf{Open}$ outputs an evaluation proof $\pi \in \mathbb{G}_1$ that is computed as follows. If for any $p_i \in \boldsymbol{p}$, $\deg(p_i) > D$, abort. For each $i \in [n]$, if ω_i is not $\perp$, then obtain random univariate polynomial $\bar{p}_i$ of degree $\deg(p_i)$ from ω_i, otherwise $\bar{p}_i$ is set to be a zero polynomial. Then compute the linear combination of polynomials $p(X) := \sum_{i=1}^n \xi^i p_i(X)$ and $\bar{p}(X) := \sum_{i=1}^n \xi^i \bar{p}_i(X)$. Compute witness polynomials $w(X) := \frac{p(X)-p(z)}{X-z}$ and $\bar{w}(X) := \frac{\bar{p}(X)-\bar{p}(z)}{X-z}$. Set $\mathsf{w} := w(\beta)G + \gamma\bar{w}(\beta)G \in \mathbb{G}_1$ and $\bar{v} := \bar{p}(z) \in \mathbb{F}_q$. The evaluation proof is $\pi := (\mathsf{w}, \bar{v})$.

Check. On input rk, commitments $\boldsymbol{c} := [c_i]_{i=1}^n$, evaluation point $z \in \mathbb{F}_q$, alleged evaluations $\boldsymbol{v} := [v_i]_{i=1}^n$, evaluation proof $\pi := (\mathsf{w}, \bar{v})$, and randomness $\xi \in \mathbb{F}_q$, $\mathsf{PC_s}.\mathsf{Check}$ proceeds as follows. Compute the linear combination $C := \sum_{i=1}^n \xi^i c_i$ Then compute the linear combination of evaluations $v := \sum_{i=1}^n \xi^i v_i$, and check the evaluation proof via the equality $e(C - vG - \gamma\bar{v}G, H) = e(\mathsf{w}, \beta H - zH)$.

Completeness. Completeness can be proved by suitably modifying the completeness proof in Section 2.11.2.1.

Succinctness. The scheme $\mathsf{PC_s}$ constructed in this section requires n $\mathbb{G}_1$ elements to commit to $\boldsymbol{c} = [c_i]_{i=1}^n$, one $\mathbb{G}_1$ and one $\mathbb{F}_q$ element for the evaluation proof, and the time to check this proof of evaluation requires two pairings and one variable-base multi-scalar multiplication of size n.

2.11.3.2 Extractability and hiding

Theorem 2.11.12. *If the bilinear group sampler* $\mathsf{SampleGrp}$ *satisfies the* SDH *assumption against algebraic adversaries (Assumption 1),* $\mathsf{nhPC_s}$ *and* $\mathsf{phPC_s}$ *constructed in Section 2.11.3.1 achieve extractability against algebraic adversaries (Definition 2.11.2).*

To prove this, we rely on the fact that $\mathsf{nhPC_s}$ and $\mathsf{phPC_s}$ satisfy evaluation binding (Definition 2.11.8):

Lemma 2.11.13. *If the bilinear group sampler* $\mathsf{SampleGrp}$ *satisfies the* SDH *assumption (Assumption 1), then* $\mathsf{nhPC_s}$ *and* $\mathsf{phPC_s}$ *constructed in Section 2.11.3.1 achieve evaluation binding (Definition 2.11.8).*

The proof of the above lemma is easily achieved by straightforward modifications to the proof of Lemma 2.11.9.

Lemma 2.11.14. *If the bilinear group sampler* SampleGrp *satisfies the* SDH *assumption against algebraic adversaries (Assumption 1),* $\mathsf{nhPC_s}$ *and* $\mathsf{phPC_s}$ *constructed in Section 2.11.3.1 achieve extractability against algebraic adversaries (Definition 2.11.2).*

Proof. Fix any efficient, algebraic adversary $\mathcal{A}_{\mathsf{alg}}$ and maximum degree bound $D \in \mathbb{N}$. We show how to construct an efficient extractor $\mathcal{E}_{\mathcal{A}_{\mathsf{alg}}}$ for the polynomial commitment scheme that succeeds with overwhelming probability. In each round $i \in [r]$ algorithm $\mathcal{E}_{\mathcal{A}_{\mathsf{alg}}}$ proceeds as follows. We denote by k the number of group elements output by the adversary $\mathcal{A}_{\mathsf{alg}}$.

$\mathcal{E}_{\mathcal{A}_{\mathsf{alg}}}(\mathsf{ck}, \mathsf{rk}; [\rho_j]_{j=1}^{i})$:

1. Parse ck as $(\langle \mathsf{group} \rangle, \boldsymbol{\Sigma})$.
2. Parse $\boldsymbol{\Sigma}$ as $\begin{pmatrix} G & \beta G & \beta^2 G & \dots & \beta^D G \\ \gamma G & \gamma\beta G & \gamma\beta^2 G & \dots & \gamma\beta^D G \end{pmatrix}$.
3. Set $\boldsymbol{\Sigma}_1 := (G, \beta G, \beta^2 G, \dots, \beta^D G)$.
4. Set $\boldsymbol{\Sigma}_2 := (\gamma G, \gamma\beta G, \gamma\beta^2 G, \dots, \gamma\beta^D G)$.
5. Invoke the adversary: $[\langle \boldsymbol{a}_j, \boldsymbol{\Sigma}_1 \rangle + \langle \boldsymbol{b}_j, \boldsymbol{\Sigma}_2 \rangle]_{j=1}^{\mathsf{k}} \leftarrow \mathcal{A}_{\mathsf{alg}}(\mathsf{ck}, \mathsf{rk}; [\rho_j]_{j=1}^{i})$.
6. Set $\boldsymbol{X} := (1, X, \dots, X^D)$.
7. For each j in $[\mathsf{k}]$, define polynomials $p_j(X) := \langle \boldsymbol{a}_j, \boldsymbol{X} \rangle \in \mathbb{F}_q[X]$ and $\bar{p}_j(X) := \langle \boldsymbol{b}_j, \boldsymbol{X} \rangle \in \mathbb{F}_q[X]$.
8. For each j in $[\mathsf{k}]$, let the randomness ω_j be the coefficients of $\bar{p}_j$.
9. Output the polynomials $\boldsymbol{p} = [p_j]_{j=1}^{\mathsf{k}}$ and randomness $\boldsymbol{\omega} := [\omega_j]_{j=1}^{\mathsf{k}}$.

For a given efficient public-coin challenger $\mathcal{C}$, efficient adversary $\mathcal{B} := (\mathcal{B}_1, \mathcal{B}_2)$, efficient query sampler $\mathcal{Q}$, and round bound $\mathsf{r} \in \mathbb{N}$, the extractor $\mathcal{E}_{\mathcal{A}_{\mathsf{alg}}}$ can fail with non-negligible probability only if there exists a polynomial whose claimed evaluation is incorrect. However, because $\mathsf{nhPC_s}$ and $\mathsf{phPC_s}$ satisfy evaluation binding, all evaluations are correct with overwhelming probability. This latter fact follows from a reduction identical to that in the corresponding portion of the extractability proof of Lemma 2.11.10. Hence, $\mathcal{E}_{\mathcal{A}_{\mathsf{alg}}}$ succeeds with overwhelming probability. □

Lemma 2.11.15. $\mathsf{phPC_s}$ *constructed in Section 2.11.3.1 is perfectly hiding (Definition 2.11.4).*

At a high level, one can adapt the proof in Theorem 2.11.11 into a proof of Lemma 2.11.15 by removing from it all terms related to α, as the proof only reasons about these terms for the sake of completeness.

2.12 Polynomial commitments for multiple degree bounds

We construct a polynomial commitment scheme that supports *multiple* degree bounds up to a maximum degree chosen at setup time.

We again temporarily restrict our attention to the case where, in the reveal phase, all polynomials are evaluated at the same evaluation point. (We will relax this restriction in Section 2.13.) We do not provide a standalone definition for the construction that we consider below, because it equals the definition in Section 2.6.1 when restricted to admissible query samplers which output query sets Q consisting of a single evaluation point at which a subset of the polynomials are evaluated (i.e., $Q = T \times \{z\}$ for some $T \subseteq [n]$, and $z \in \mathbb{F}$).[14]

We proceed as follows. First, in Section 2.12.1, we present a construction for the above goal that builds upon ideas in Section 2.11.3.1. Then, in Section 2.12.2, we reduce the hiding and extractability of this construction to the hiding and extractability of a related construction that is simpler to analyze, but is not degree-efficient. This simpler construction might also be of independent interest.

2.12.1 Degree-efficient construction

We demonstrate how to construct a polynomial commitment that supports multiple degree bounds efficiently. Our construction builds upon the construction in Section 2.11.3.1. A polynomial commitment scheme over a field family $\mathcal{F}$ for **multiple degree bounds and a single evaluation point** is a tuple of algorithms $\mathsf{PC_m} = (\mathsf{Setup}, \mathsf{Trim}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Check})$ with the following syntax. Below we use $[[a_i, b_i]]_{i=1}^n$ as a short-hand for the tuple $(a_1, b_1, \ldots, a_n, b_n)$. The highlighted text below denotes the parts of the construction that differ from the construction in Section 2.11.3.1.

Setup. On input a security parameter λ (in unary), and a maximum degree bound $D \in \mathbb{N}$, $\mathsf{PC_m}.\mathsf{Setup}$ samples public parameters pp as follows. Sample a bilinear group $\langle \mathsf{group} \rangle \leftarrow \mathsf{SampleGrp}(1^\lambda)$, and parse $\langle \mathsf{group} \rangle$ as a tuple $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, G, H, e)$. Sample random elements $\beta, \gamma \in \mathbb{F}_q$. Then compute the vector

$$\boldsymbol{\Sigma} := \begin{pmatrix} G & \beta G & \beta^2 G & \ldots & \beta^D G \\ \gamma G & \gamma\beta G & \gamma\beta^2 G & \ldots & \gamma\beta^D G \end{pmatrix} \in \mathbb{G}_1^{2D+2} \ .$$

Set $\mathsf{pp} := (D, \langle \mathsf{group} \rangle, \boldsymbol{\Sigma}, \beta H)$, and then output the public parameters pp. These public parameters will support polynomials over the field $\mathbb{F}_q$ of degree at most D.

Trim. Given oracle access to public parameters pp, and on input a security parameter λ (in unary), and degree bounds $\boldsymbol{d} = [d_i]_{i=1}^n$, $\mathsf{PC_m}.\mathsf{Trim}^{\mathsf{pp}}$ deterministically computes a key pair $(\mathsf{ck}, \mathsf{rk})$ that is specialized to $\boldsymbol{d}$ as follows. Let d be the maximum degree bound in $\boldsymbol{d}$. Then obtain $\boldsymbol{\Sigma}_{\mathsf{ck}}$ from public parameters:

$$\boldsymbol{\Sigma}_{\mathsf{ck}} := \begin{pmatrix} G & \beta G & \ldots & \beta^d G & \beta^{D-d} G & \beta^{D-d+1} G & \ldots & \beta^D G \\ \gamma G & \gamma\beta G & \ldots & \gamma\beta^d G & & & & \end{pmatrix} \in \mathbb{G}_1^{3d+3} \ .$$

[14] Recall from Definition 2.6.5 that an admissible query sampler outputs query sets such that every polynomial is evaluated at least once at a point sampled from a super-polynomially-large subset.

Set $\mathsf{ck} := (\boldsymbol{\Sigma}_{\mathsf{ck}}, \boldsymbol{d})$. Let $\boldsymbol{\Sigma}_{\mathsf{rk}}$ be the set $\{\beta^{D-d_i}G\}_{i\in[n]}$ and $\mathsf{rk} := (D, \langle\mathsf{group}\rangle, \boldsymbol{\Sigma}_{\mathsf{rk}}, \gamma G, \beta H, \boldsymbol{d})$, and then output the key pair $(\mathsf{ck}, \mathsf{rk})$. This key pair is specialized to $\boldsymbol{d}$.

Commit. On input ck, univariate polynomials $\boldsymbol{p} = [p_i]_{i=1}^n$ over the field $\mathbb{F}_q$, degree bounds $\boldsymbol{d} = [d_i]_{i=1}^n$ with $\deg(\boldsymbol{p}) \leq \boldsymbol{d} \leq D$, and randomness $\boldsymbol{\omega} = [[\omega_i, \omega_i']]_{i=1}^n$, $\mathsf{PC}_{\mathsf{m}}.\mathsf{Commit}$ outputs commitments $\boldsymbol{c} = [c_i]_{i=1}^n$ that are computed as follows. Obtain the supported degree bounds $\boldsymbol{d}'$ from ck. If for any $p_i \in \boldsymbol{p}$, $\deg(p_i) > d_i$ or $d_i \notin \boldsymbol{d}'$, abort. For each $i \in [n]$, if ω_i and ω_i' are not $\perp$, then obtain from them random univariate polynomials $\bar{p}_i$ and $\bar{p}_i'$ of degree $\deg(p_i)$; otherwise, set $\bar{p}_i$ and $\bar{p}_i'$ to be the zero polynomial. For each $i \in [n]$, compute $\mathbb{c}_i := p_i(\beta)G + \gamma\bar{p}_i(\beta)G$ and $\mathbb{c}_i' := \beta^{D-d_i}p_i(\beta)G + \gamma\bar{p}_i'(\beta)G$. Finally, set $c_i := (\mathbb{c}_i, \mathbb{c}_i')$, and output $\boldsymbol{c} := [c_i]_{i=1}^n$. Note that because $p_i(X)$, $X^{D-d_i}p_i(X)$, $\bar{p}_i(X)$ and $\bar{p}_i'(X)$ have at most d_i non-zero coefficients, the above terms are linear combinations of terms in ck.

Open. On input ck, univariate polynomials $\boldsymbol{p} = [p_i]_{i=1}^n$ over the field $\mathbb{F}_q$, degree bounds $\boldsymbol{d} = [d_i]_{i=1}^n$, evaluation point $z \in \mathbb{F}_q$, opening challenge ξ, and randomness $\boldsymbol{\omega} = [[\omega_i, \omega_i']]_{i=1}^n$, $\mathsf{PC}_{\mathsf{m}}.\mathsf{Open}$ outputs the evaluation proof π as follows. Obtain the supported degree bounds $\boldsymbol{d}'$ from ck. If for any $p_i \in \boldsymbol{p}$, $\deg(p_i) > d_i$ or $d_i \notin \boldsymbol{d}'$, abort. For each $i \in [n]$, if ω_i and ω_i' are not $\perp$, obtain from them random univariate polynomials $\bar{p}_i$ and $\bar{p}_i'$ of degree $\deg(p_i)$; otherwise, set $\bar{p}_i$ and $\bar{p}_i'$ to be the zero polynomial.

Then, for each $i \in [n]$, define the polynomial $p_i^\star(X) := X^{D-d_i}p_i(X) - X^{D-d_i}p_i(z)$, and compute a witness polynomial $w_i(X) := \frac{p_i(X)-p_i(z)}{X-z}$ for p_i, and a witness polynomial $w_i^\star(X) := X^{D-d_i}w_i(X)$ for $p_i^\star$. Finally, compute the witness polynomial for these $2n$ polynomials as $w := \sum_{i=1}^n \xi^i w_i + \sum_{i=1}^n \xi^{n+i} w_i^\star$.

Next, compute the linear combination of the random polynomials $\bar{p} := \sum_{i=1}^n \xi^i \bar{p}_i$ and $\bar{p}' := \sum_{i=1}^n \xi^{n+i}\bar{p}_i'$, and compute the witness polynomial $\bar{w}(X) := \frac{\bar{p}(X)-\bar{p}(z)+\bar{p}'(X)-\bar{p}'(z)}{X-z}$ for these. Set $\mathsf{w} := w(\beta)G + \gamma\bar{w}(\beta)G \in \mathbb{G}_1$, and $\bar{v} := \bar{p}(z) + \bar{p}'(z) \in \mathbb{F}_q$. The evaluation proof is $\pi := (\mathsf{w}, \bar{v})$.

Check. On input rk, commitments $\boldsymbol{c} = [c_i]_{i=1}^n$, degree bounds $\boldsymbol{d} = [d_i]_{i=1}^n$, evaluation point $z \in \mathbb{F}_q$, alleged evaluations $\boldsymbol{v} = [v_i]_{i=1}^n$, evaluation proof $\pi = (\mathsf{w}, \bar{v})$, and opening challenge ξ, $\mathsf{PC}_{\mathsf{m}}.\mathsf{Check}$ proceeds as follows. Obtain the supported degree bounds $\boldsymbol{d}'$ from rk. If for any $d_i \in \boldsymbol{d}$, $d_i \notin \boldsymbol{d}'$, abort. Parse each commitment c_i as a pair of sub-commitments $(\mathbb{c}_i, \mathbb{c}_i')$, and construct $\mathbb{c}_i^\star := \mathbb{c}_i' - v_i\beta^{D-d_i}G$. Next, compute the two linear combinations

$$C := \sum_{i=1}^n \xi^i \mathbb{c}_i + \sum_{i=1}^n \xi^{n+i}\mathbb{c}_i^\star, \quad v := \sum_{i=1}^n \xi^i v_i,$$

Then check the evaluation proof $\pi = (\mathsf{w}, \bar{v})$ via the equality $e(C - vG - \gamma\bar{v}G, H) = e(\mathsf{w}, \beta H - zH)$.

Lemma 2.12.1. *The scheme PC_{m} constructed above achieves completeness (Definition 2.6.1).*

Proof. Fix any maximum degree bounds D, $\boldsymbol{d} = [d_i]_{i=1}^n$ and efficient adversary $\mathcal{A}$. Let pp be any public parameters output by the algorithm $\mathsf{PC}_{\mathsf{m}}.\mathsf{Setup}(1^\lambda, D)$. Let $\mathcal{A}(\mathsf{pp})$ select polynomials $\boldsymbol{p} = [p_i]_{i=1}^n$ over $\mathbb{F}_q$, degree bounds $\boldsymbol{d} = [d_i]_{i=1}^n$, location $z \in \mathbb{F}_q$, and opening challenge $\xi \in \mathbb{F}_q$. We only need to consider adversaries $\mathcal{A}$ that make choices for which $\deg(\boldsymbol{p}) \leq d_i \leq D$. Let $(\mathsf{ck}, \mathsf{rk})$

be any key pair output by the algorithm $\mathsf{PC}_{\mathsf{m}}.\mathsf{Trim}^{\mathsf{pp}}(1^{\lambda}, \boldsymbol{d})$ constructed above. The keys contain a description $\langle \mathsf{group} \rangle$ of a bilinear group of some prime order q, which in particular induces a field $\mathbb{F}_q$.

Now consider commitments $\boldsymbol{c} = [c_i]_{i=1}^n$ and evaluation proof π that are all computed according to the construction above. We need to show that, for the correct evaluations $\boldsymbol{v} := \boldsymbol{p}(z)$,

$$\mathsf{PC}_{\mathsf{m}}.\mathsf{Check}(\mathsf{rk}, \boldsymbol{c}, \boldsymbol{d}, z, \boldsymbol{v}, \pi, \xi) = 1 \ .$$

This amounts to arguing that the pairing equations are satisfied. For these equations, note that the combined commitment C and evaluation v are computed by $\mathsf{PC}_{\mathsf{m}}.\mathsf{Check}$ as follows:

$$\begin{aligned}
C &= \sum_{i=1}^{n} \xi^i \mathbb{c}_i &&+ \sum_{i=1}^{n} \xi^{n+i} \mathbb{c}_i^{\star} \\
&= \sum_{i=1}^{n} \xi^i (p_i(\beta)G + \gamma \bar{p}_i(\beta)G) &&+ \sum_{i=1}^{n} \xi^{n+i} (\beta^{D-d_i}(p_i(\beta) - p_i(z))G + \gamma \bar{p}_i'(\beta)G) \\
&= \sum_{i=1}^{n} \xi^i (p_i(\beta)G + \gamma \bar{p}_i(\beta)G) &&+ \sum_{i=1}^{n} \xi^{n+i} (p_i^{\star}(\beta)G + \gamma \bar{p}_i'(\beta)G) \ , \\
v &= \sum_{i=1}^{n} \xi^i v_i = \sum_{i=1}^{n} \xi^i p_i(z) \ .
\end{aligned}$$

In the evaluation proof $\pi = (\mathsf{w}, \bar{v})$, we have that

$$\begin{aligned}
\mathsf{w} &= (w(\beta) + \gamma \bar{w}(\beta))G \\
&= \sum_{i=1}^{n} \xi^i w_i(\beta)G + \sum_{i=1}^{n} \xi^{n+i} w_i^{\star}(\beta)G + \sum_{i=1}^{n} \xi^i \bar{w}_i(\beta)\gamma G + \sum_{i=1}^{n} \xi^{n+i} \bar{w}_i'(\beta)\gamma G \\
&= \sum_{i=1}^{n} \frac{\xi^i (p_i(\beta) - p_i(z)) + \xi^{n+i} p_i^{\star}(\beta) + \xi^i (\bar{p}_i(\beta) - \bar{p}_i(z))\gamma + \xi^{n+i} (\bar{p}_i'(\beta) - \bar{p}_i'(z))\gamma}{\beta - z} G
\end{aligned}$$

We also have that the evaluation $\bar{v} = \bar{p}(z) + \bar{p}'(z)$. Therefore,

$$\begin{aligned}
& e(C - vG - \gamma \bar{v} G, H) \\
&= e(\textstyle\sum_{i=1}^{n} (\xi^i ((p_i(\beta) - v_i) + \gamma(\bar{p}_i(\beta) - \bar{p}_i(z))) + \xi^{n+i}(p_i^{\star}(\beta) + \gamma(\bar{p}_i'(\beta) - \bar{p}_i'(z))))G, H) \\
&= e(\tfrac{\sum_{i=1}^{n} \xi^i (p_i(\beta) - p_i(z)) + \xi^{n+i} p_i^{\star}(\beta) + \xi^i (\bar{p}_i(\beta) - \bar{p}_i(z))\gamma + \xi^{n+i}(\bar{p}_i'(\beta) - \bar{p}_i'(z))\gamma}{\beta - z} G, (\beta - z)H) \\
&= e((w(\beta) + \gamma \bar{w}(\beta))G, \beta H - zH) \\
&= e(\mathsf{w}, \beta H - zH) \ .
\end{aligned}$$

We conclude that the pairing equation also holds. □

Lemma 2.12.2. *The scheme* PC_{m} *constructed achieves efficiency, as defined in Definition 2.6.3.*

Proof. PC_m satisfies both efficiency properties:

- *Degree-efficiency:* For a list of n polynomials with degree bounds $\boldsymbol{d} = [d_i]_{i=1}^n$ where $d = \max(\boldsymbol{d})$ is the maximum supported degree bounds for these polynomials, both $\mathsf{PC}_\mathsf{m}.\mathsf{Commit}$ and $\mathsf{PC}_\mathsf{m}.\mathsf{Open}$ only handle polynomials having at most d coefficients, and so the time to commit to the polynomials is the time for $4n$ variable-base multi-scalar multiplications of size at most d, while the time to compute an evaluation proof is the time to compute two polynomial divisions of degree at most d plus the time required for two variable-base multi-scalar multiplications of size at most d.
- *Succinctness:* For a list of n polynomials, the scheme PC_m requires $2n$ $\mathbb{G}_1$ elements for a commitment and one $\mathbb{G}_1$ element and one $\mathbb{F}_q$ element for an evaluation proof, while the time to check this proof requires two variable-base multi-scalar multiplications of size n and two pairings.

□

Extractability and hiding. We reduce the extractability and hiding properties of our construction to those of a simpler-to-analyze construction described below. This latter construction makes black-box use of any PC_s scheme.

2.12.2 Black-box construction

We now provide a simpler construction of PC_m that makes black-box use of PC_s. This construction is *not* degree-efficient (Definition 2.6.3), but is simpler to analyze.

Setup. On input a security parameter λ (in unary), and a maximum degree bound $D \in \mathbb{N}$, $\mathsf{PC}_\mathsf{m}.\mathsf{Setup}$ samples and outputs $\mathsf{pp} := \mathsf{PC}_\mathsf{s}.\mathsf{Setup}(1^\lambda, D)$. The keys contain the description of a finite field $\mathbb{F} \in \mathcal{F}$.

Trim. Given oracle access to public parameters pp, and on input a security parameter λ (in unary), and degree bounds $\boldsymbol{d}$, $\mathsf{PC}_\mathsf{m}.\mathsf{Trim}$ simply parses pp as $(\mathsf{ck}, \mathsf{rk})$ and outputs these.

Commit. On input ck, univariate polynomials $\boldsymbol{p} = [p_i]_{i=1}^n$ over the field $\mathbb{F}$, degree bounds $\boldsymbol{d} = [d_i]_{i=1}^n$ with $\deg(\boldsymbol{p}) \leq \boldsymbol{d} \leq D$, and randomness $\boldsymbol{\omega} = [[\omega_i, \omega_i']]_{i=1}^n$, $\mathsf{PC}_\mathsf{m}.\mathsf{Commit}$ outputs commitments $\boldsymbol{c} = [c_i]_{i=1}^n$ that are computed as follows. First, for each $i \in [n]$, define the *shifted* polynomial $p_i'(X) := X^{D-d_i} p_i(X)$. Next, use PC_s to simultaneously commit to all unshifted and shifted polynomials: $[[\mathbb{c}_i, \mathbb{c}_i']]_{i=1}^n := \mathsf{PC}_\mathsf{s}.\mathsf{Commit}(\mathsf{ck}, [[p_i, p_i']]_{i=1}^n; \boldsymbol{\omega})$. Finally, set $c_i := (\mathbb{c}_i, \mathbb{c}_i')$, and output $\boldsymbol{c} := [c_i]_{i=1}^n$. Note that every polynomial being committed has degree at most D.

Open. On input ck, univariate polynomials $\boldsymbol{p} = [p_i]_{i=1}^n$ over the field $\mathbb{F}$, degree bounds $\boldsymbol{d} = [d_i]_{i=1}^n$, evaluation point $z \in \mathbb{F}$, opening challenge ξ, and randomness $\boldsymbol{\omega} = [[\omega_i, \omega_i']]_{i=1}^n$, $\mathsf{PC}_\mathsf{m}.\mathsf{Open}$ outputs the evaluation proof $\pi := \mathsf{PC}_\mathsf{s}.\mathsf{Open}(\mathsf{ck}, [[p_i, p_i']]_{i=1}^n, z, \xi; \boldsymbol{\omega})$, where each p_i' is the shift of p_i respectively.

Check. On input rk, commitments $\boldsymbol{c} = [c_i]_{i=1}^n$, degree bounds $\boldsymbol{d} = [d_i]_{i=1}^n$, evaluation point $z \in \mathbb{F}$, alleged evaluations $\boldsymbol{v} = [v_i]_{i=1}^n$, evaluation proof π, and opening challenge ξ, $\mathsf{PC}_\mathsf{m}.\mathsf{Check}$ proceeds as follows. Parse each commitment c_i as a pair of sub-commitments $(\mathbb{c}_i, \mathbb{c}_i')$. For each $i \in [n]$, compute the shifted evaluation $v_i' := z^{D-d_i} v_i$. Check that $\mathsf{PC}_\mathsf{s}.\mathsf{Check}(\mathsf{rk}, [[\mathbb{c}_i, \mathbb{c}_i']]_{i=1}^n, z, [[v_i, v_i']]_{i=1}^n, \pi, \xi)$ accepts.

Lemma 2.12.3. *If* PC_s *achieves completeness (Definition 2.11.1) then* PC_m *achieves completeness (Definition 2.6.1).*

Proof. If an adversary $\mathcal{A}(\mathsf{ck}, \mathsf{rk})$ selects polynomials $[p_i]_{i=1}^n$, degree bounds $[d_i]_{i=1}^n$, evaluation point $z \in \mathbb{F}$, and opening challenge ξ such that, for every $i \in [n]$, we have $\deg(p_i) \leq d_i \leq D$, then both the unshifted polynomials $[p_i]_{i=1}^n$ and shifted polynomials $[p'_i]_{i=1}^n$ have degree at most D. Furthermore, because the shifted polynomials are computed as $p'(X) = X^{D-d_i}p(X)$, the shifted evaluations will always match: $v'_i = z^{D-d_i} \cdot v_i$. This means that the completeness of PC_s ensures that the commitments produced via PC_s will pass the tests in $\mathsf{PC}_\mathsf{s}.\mathsf{Check}$. □

Lemma 2.12.4. *If* PC_s *achieves succinctness (Definition 2.11.3) then* PC_m *achieves succinctness (Definition 2.6.3).*

Proof. The commitment of PC_m contains $2n$ PC_s commitments and the evaluation proof contains one PC_s proof. The time to check n evaluations is the same as the time to check $2n$ evaluations in PC_s, plus at most $n \cdot (\log D)$ field operations to compute $\boldsymbol{v}'$. □

Lemma 2.12.5. *If the construction in Section 2.12.2 achieves extractability and hiding when instantiated with* PC_s *from Section 2.11.3, then so does the construction in Section 2.12.1.*

Proof. We show how to reduce the extractability and hiding of the construction in Section 2.12.1 (denoted by PC_m) to that of the foregoing construction (denoted by PC'_m). For simplicity, we consider the case of a single polynomial p with a single degree bound d evaluated at the query point z.

- *Extractability:* Note that PC_m commitments are identical to PC'_m commitments. The same holds for evaluation proofs for unshifted polynomials. Hence, the only difference is in how degree bounds are enforced, and so we focus on this latter aspect.
  Define the polynomials $p_1 := X^{D-d}p(X) - X^{D-d}v$ and $p_2 := X^{D-d}p(X)$. To enforce degree bounds, PC_m provides an evaluation proof for the claim that $p_1(z) = 0$, while PC'_m provides a proof for the claim that $p_2(z) = z^{D-d}v$. However, notice that $p_2(z) = z^{D-d}v$ if and only if $p_2(z) = 0$, and so a PC_m evaluation proof is valid only if the "corresponding" PC'_m proof is also valid.
- *Hiding:* Note that PC_m commitments and evaluation proofs for unshifted polynomials are identical to those for PC'_m, and so we focus on evaluation proofs for shifted polynomials. As seen from the calculation of PC_m, the witness polynomial for p_1 is a shifted version of the witness polynomial for p, and hence the evaluation proof does not reveal any additional information about p.

□

2.12.2.1 Extractability

Theorem 2.12.6. *If* PC_s *achieves extractability (Definition 2.11.2) then* PC_m *achieves extractability (Definition 2.6.2) restricted to query sets* Q *querying a subset of polynomials at the same point (i.e.,* $Q = T \times \{z\}$ *for some* $T \subseteq [n]$*, and* $z \in \mathbb{F}$*).*

Proof. Fix a maximum degree bound D and an efficient adversary $\mathcal{A}$ against PC_m. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ and query sampler $\mathcal{Q}'$ against PC_s. By assumption there exists a PC_s extractor $\mathcal{E}_\mathcal{B}$ against $\mathcal{B}$. We use $\mathcal{E}_\mathcal{B}$ to construct a PC_m extractor $\mathcal{E}_\mathcal{A}$ for $\mathcal{A}$.

$\mathcal{B}(\mathsf{ck}, \mathsf{rk}, [\rho_j]_{j=1}^i)$:	$\mathcal{E}_\mathcal{A}(\mathsf{pp}, [\rho_j]_{j=1}^i)$:
1. Set $\mathsf{pp} := (\mathsf{ck}, \mathsf{rk})$.	1. Parse pp as $(\mathsf{ck}, \mathsf{rk})$.
2. Compute $(\boldsymbol{c}, \boldsymbol{d}) \leftarrow \mathcal{A}(\mathsf{pp}, [\rho_j]_{j=1}^i)$.	2. Compute $\boldsymbol{p} \leftarrow \mathcal{E}_\mathcal{B}(\mathsf{ck}, \mathsf{rk}, [\rho_j]_{j=1}^i)$.
3. Parse $\boldsymbol{c}$ as $[[\mathbb{c}_i, \mathbb{c}_i']]_{i=1}^n$.	3. Parse $\boldsymbol{p}$ as $[[p_i, p_i']]_{i=1}^n$.
4. Output $[[\mathbb{c}_i, \mathbb{c}_i']]_{i=1}^n$.	4. Output $[p_i]_{i=1}^n$.

Suppose for contradiction that the extractor $\mathcal{E}_\mathcal{A}$ fails with non-negligible probability for some choice of round bound $\mathsf{r} \in \mathbb{N}$, efficient public-coin challenger $\mathcal{C}$, efficient query sampler $\mathcal{Q}$, and efficient adversary $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$. This can occur due to one of two reasons.

(1) *Extracted polynomial does not match evaluation:* there exists an extracted polynomial whose claimed evaluation is incorrect.

(2) *Degree bounds are not satisfied:* all extracted polynomials match their claimed evaluations, but there exists a polynomial whose degree differs from the claimed degree.

If $\mathcal{E}_\mathcal{A}$ fails with non-negligible probability, then one of these cases occurs with non-negligible probability. We analyze both cases, and argue that this cannot be.

(1) Extracted polynomial does not match evaluation. Each PC_m commitment c_i in $[c_i]_{i \in T}$ is a pair of PC_s commitments $(\mathbb{c}_i, \mathbb{c}_i')$. Since PC_m.Check invokes PC_s.Check, $[c_i]_{i \in T}$ are accepted by PC_m.Check if and only if $[[\mathbb{c}_i, \mathbb{c}_i']]_{i \in T}$ are accepted by PC_s.Check. Thus, if PC_m.Check accepts but the extractor $\mathcal{E}_\mathcal{A}$ fails with non-negligible probability, then we deduce that PC_s.Check accepts but the extractor $\mathcal{E}_\mathcal{B}$ fails with non-negligible probability against a PC_s query sampler $\mathcal{Q}'$ that obtains Q from $\mathcal{Q}$ and outputs $Q' := \{(2i-1, z), (2i, z) \mid (i, z) \in Q\}$. This contradicts the fact that PC_s achieves extractability. Hence, we conclude that all extracted polynomials match their claimed evaluations with probability negligibly close to 1.

(2) Degree bounds are not satisfied. We first recall how the extractor $\mathcal{E}_\mathcal{A}$ works: it invokes the PC_s extractor to obtain $2n$ polynomials $\boldsymbol{p} := [p_i, p_i']_{i=1}^n$, and outputs $[p_i]_{i=1}^n$. The remaining polynomials $[p_i']_{i=1}^n$ are supposedly "shifted" versions of the output polynomials. It should be the case that for each $i \in T$ it holds that $p_i'(X) = X^{D-d_i} p_i(X)$. To check that this condition is satisfied, PC_m.Check verifies that, for a point $z \in \mathbb{F}$ sampled by the admissible query sampler $\mathcal{Q}$, $v_i' := z^{D-d_i} p_i(z)$ is a valid evaluation for p_i'. The probability that this equation holds but $p_i'(X) \neq X^{D-d_i} p_i(X)$ is negligibly small because $\mathcal{Q}$, being admissible, samples z from a super-polynomially-large subset of $\mathbb{F}$. □

2.12.2.2 Hiding

Theorem 2.12.7. *If PC_s achieves hiding (Definition 2.11.4) then PC_m achieves hiding (Definition 2.6.4) restricted to query sets Q_j querying a subset of polynomials at the same point (i.e., $Q_j = T_j \times \{z_j\}$ for some $T_j \subseteq [n]$, and $z_j \in \mathbb{F}$).*

Proof. Below we construct a simulator $\mathcal{S}_{\mathsf{m}}$ for PC_{m} using a simulator $\mathcal{S}_{\mathsf{s}}$ for PC_{s} as a subroutine.

$\mathcal{S}_{\mathsf{m}}.\mathsf{Setup}(1^\lambda, D)$:
1. Compute $(\mathsf{ck}, \mathsf{rk}, \mathsf{trap}) \leftarrow \mathcal{S}_{\mathsf{s}}.\mathsf{Setup}(1^\lambda, D)$.
2. Output $(\mathsf{pp} := (\mathsf{ck}, \mathsf{rk}), \mathsf{trap})$.

$\mathcal{S}_{\mathsf{m}}.\mathsf{Commit}(\mathsf{trap}, \boldsymbol{d}; \boldsymbol{\omega})$:
1. Ignore the degrees $\boldsymbol{d}$, and parse the randomness $\boldsymbol{\omega}$ as $[[\omega_i, \omega_i']]_{i=1}^{\mathsf{k}}$.
2. Compute $[[\mathbb{c}_i, \mathbb{c}_i']]_{i=1}^{\mathsf{k}} := \mathcal{S}_{\mathsf{s}}.\mathsf{Commit}(\mathsf{trap}, 2\mathsf{k}; [[\omega_i, \omega_i']]_{i=1}^{\mathsf{k}})$.
3. For each $i \in [\mathsf{k}]$, assemble the pair $c_i := (\mathbb{c}_i, \mathbb{c}_i')$ of simulated commitments.
4. Output the simulated commitments $\boldsymbol{c} := [c_i]_{i=1}^n$ for PC_{m}.

$\mathcal{S}_{\mathsf{m}}.\mathsf{Open}(\mathsf{trap}, \boldsymbol{p}, \boldsymbol{v}, \boldsymbol{d}, Q, \xi; \boldsymbol{\omega})$:
1. Parse $\boldsymbol{p}, \boldsymbol{v}, \boldsymbol{d}, \boldsymbol{\omega}$ as $[p_i]_{i=1}^n, [v_i]_{i=1}^n, [d_i]_{i=1}^n, [[\omega_i, \omega_i']]_{i=1}^n$.
2. Parse Q as $T \times \{z\}$ for some $T \subseteq [n]$ and $z \in \mathbb{F}$.
3. For each $i \in T$, set $p_i'(X) := X^{D-d_i} p_i(X)$ and $v_i' := z^{D-d_i} v_i$.
4. Construct PC_{s} query set $Q' := \{(2i-1, z), (2i, z) \mid (i, z) \in Q\}$.
5. Output $\pi \leftarrow \mathcal{S}_{\mathsf{s}}.\mathsf{Open}(\mathsf{trap}, [p_i, p_i']_{i=1}^n, [v_i, v_i']_{i=1}^n, Q', \xi; [[\omega_i, \omega_i']]_{i=1}^n)$.

The simulator $\mathcal{S}_{\mathsf{m}}$ is a simple wrapper around the simulator $\mathcal{S}_{\mathsf{s}}$, and it is straightforward to see that if $\mathcal{S}_{\mathsf{s}}$ simulates correctly for PC_{s} then $\mathcal{S}_{\mathsf{m}}$ simulates correctly for PC_{m}. □

2.13 Polynomial commitments that support different query locations

We construct a polynomial commitment scheme PC that supports different query locations– an instantiation of the primitive defined in Section 2.6.1. The query set Q consists of tuples $(i, z) \in [n] \times \mathbb{F}_q$ of polynomial indices and evaluation points. The construction is again a black-box extension of the polynomial commitment scheme PC_m considered in Section 2.12 which only supports a single query.

2.13.1 Construction

$\mathsf{PC.Setup}$, $\mathsf{PC.Trim}$, and $\mathsf{PC.Commit}$ equal $\mathsf{PC}_\mathsf{m}.\mathsf{Setup}$, $\mathsf{PC}_\mathsf{m}.\mathsf{Trim}$, and $\mathsf{PC}_\mathsf{m}.\mathsf{Commit}$ in Section 2.12, and so below we show how to construct $\mathsf{PC.Open}$ and $\mathsf{PC.Check}$.

Open. On input ck, univariate polynomials $\boldsymbol{p}$ over the field $\mathbb{F}$, degree bounds $\boldsymbol{d}$, the query set Q, opening challenge ξ, and randomness $\boldsymbol{\omega}$ that is the same one used in $\mathsf{PC.Commit}$, $\mathsf{PC.Open}$ proceeds as follows. Suppose there are t different evaluation points $[z_i]_{i=1}^t$ in the query set Q. Divide $\boldsymbol{p}$ into different (possibly overlapping) groups $[\boldsymbol{p}_i]_{i=1}^t$, where every polynomial in $\boldsymbol{p}_i$ is evaluated at point z_i according to Q. Similarly divide degree bounds $\boldsymbol{d}$ and $\boldsymbol{\omega}$ as $[\boldsymbol{d}_i]_{i=1}^t$ and $[\boldsymbol{\omega}_i]_{i=1}^t$ so that $\deg(\boldsymbol{p}_i) \leq \boldsymbol{d}_i$ and $\boldsymbol{\omega}_i$ is the randomness for the polynomial $\boldsymbol{p}_i$. For each group $\boldsymbol{p}_i$, obtain the evaluation proof $\pi_i := \mathsf{PC}_\mathsf{m}.\mathsf{Open}(\mathsf{ck}, \boldsymbol{p}_i, \boldsymbol{d}_i, z_i, \xi; \boldsymbol{\omega}_i)$. Output all the proofs of evaluation $[\pi_i]_{i=1}^t$.

Check. On input rk, commitments $\boldsymbol{c}$, degree bounds $\boldsymbol{d}$, the query set Q, alleged evaluations $\boldsymbol{v}$, evaluation proof π, and opening challenge ξ, $\mathsf{PC.Check}$ proceeds as follows. Suppose there are t different evaluation points $[z_i]_{i=1}^t$ in the query set Q. Parse $\boldsymbol{c}$, $\boldsymbol{d}$, $\boldsymbol{v}$ and π as $[\boldsymbol{c}_i]_{i=1}^t$, $[\boldsymbol{d}_i]_{i=1}^t$, $[\boldsymbol{v}_i]_{i=1}^t$ and $[\pi_i]_{i=1}^t$ so that $\boldsymbol{c}_i$ are the commitments of polynomials $\boldsymbol{p}_i$, where $\deg(\boldsymbol{p}_i) \leq \boldsymbol{d}_i$ and $\boldsymbol{p}_i(z_i)$ is supposed to be $\boldsymbol{v}_i$. For each $i \in [t]$, check that $\mathsf{PC}_\mathsf{m}.\mathsf{Check}(\mathsf{rk}, \boldsymbol{c}_i, \boldsymbol{d}_i, z_i, \boldsymbol{v}_i, \pi_i, \xi)$ accepts.

Completeness. The completeness of PC follows directly from the completeness of PC_m.

Efficiency. PC satisfies both efficiency properties defined in Definition 2.6.3:

- *Degree efficiency:* The degree efficiency of PC follows directly from the degree efficiency of PC_m, because $\mathsf{PC.Commit}$ is the same as $\mathsf{PC}_\mathsf{m}.\mathsf{Commit}$, and $\mathsf{PC.Open}$ invokes $\mathsf{PC}_\mathsf{m}.\mathsf{Open}$ a total of $t = |Q|$.
- *Succinctness:* The succinctness of PC follows directly from the succinctness of PC_m. In particular, a PC commitment equals a PC_m commitment, and is hence of size $\mathrm{poly}(\lambda)$. Similarly, a PC evaluation proof consists of t PC_m evaluation proofs, and is hence of size $t \cdot \mathrm{poly}(\lambda) = \mathrm{poly}(\lambda)$. Finally, $\mathsf{PC.Check}$ invokes $\mathsf{PC}_\mathsf{m}.\mathsf{Check}$ t times such that the i-th invocation is over $|\boldsymbol{c}_i|$ commitments. Because $\mathsf{PC}_\mathsf{m}.\mathsf{Check}$ takes time $n \cdot \mathrm{poly}(\lambda)$ to check n commitments, $\mathsf{PC.Check}$ takes time $\sum_{i=1}^t |\boldsymbol{c}_i| \cdot \mathrm{poly}(\lambda)$.

2.13.2 Extractability

Theorem 2.13.1. *If* PC_{m} *in Section 2.12 achieves extractability (Definition 2.6.2) then* PC *also achieves extractability (Definition 2.6.2).*

Proof. Suppose for contradiction that there exists a maximum degree bound $D \in \mathbb{N}$ and efficient adversary $\mathcal{A}$ against PC such that for some choice of round bound $\mathsf{r} \in \mathbb{N}$, efficient public-coin challenger $\mathcal{C}$, efficient query sampler $\mathcal{Q}$, and efficient adversary $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$, every efficient extractor $\mathcal{E}_{\mathcal{A}}$ fails with non-negligible probability $\mu(\lambda)$.

Then, we show how to use these to break extractability for PC_{m} by constructing adversary $\mathcal{A}'$, query sampler $\mathcal{Q}'$, and adversary $\mathcal{B}'$ as follows. We define $\mathcal{A}'$ to equal $\mathcal{A}$, and construct $\mathcal{Q}'$ and $\mathcal{B}'$ below.

$\mathcal{Q}'(\mathsf{pp}, [\rho_j]_{j=1}^{\mathsf{r}})$:
1. Obtain query set $Q \leftarrow \mathcal{Q}(\mathsf{pp}, [\rho_j]_{j=1}^{\mathsf{r}})$.
2. Parse Q as $\cup_{j \in [t]} T_j \times \{z_j\}$, for some $T_j \subseteq [n]$ and $z_j \in \mathbb{F}$, where each z_j is distinct.
3. Uniformly sample $k \in [t]$.
4. Output $Q' := T_k \times \{z_k\}$.

$\mathcal{B}'_1(\mathsf{pp}, [\rho_j]_{j=1}^{\mathsf{r}}, Q)$:
1. Parse Q as $\{(i, z) \mid i \in [n]\}$ for some $z \in \mathbb{F}$.
2. Obtain query set $Q_{\mathsf{PC}} \leftarrow \mathcal{Q}(\mathsf{pp}, [\rho_j]_{j=1}^{\mathsf{r}})$.
3. Parse Q_{PC} as $\cup_{j \in [t]} T_j \times \{z_j\}$, for some $T_j \subseteq [n]$ and $z_j \in \mathbb{F}$, where each z_j is distinct.
4. Check that for some $k \in [t]$, $Q = Q_k$.
5. Obtain $(\boldsymbol{v}, \mathsf{st}_{\mathsf{PC}}) \leftarrow \mathcal{B}_1(\mathsf{pp}, [\rho_j]_{j=1}^{\mathsf{r}}, Q_{\mathsf{PC}})$.
6. Parse $\boldsymbol{v}$ as $[\boldsymbol{v}_i]_{i=1}^{t}$ similarly to above.
7. Output $(\boldsymbol{v}_k, \mathsf{st} := (\mathsf{st}_{\mathsf{PC}}, k))$.

$\mathcal{B}'_2(\mathsf{st}, \xi)$:
1. Parse st as $(\mathsf{st}_{\mathsf{PC}}, k)$.
2. Obtain proof $\pi \leftarrow \mathcal{B}_2(\mathsf{st}_{\mathsf{PC}}, \xi)$.
3. Parse π as $[\pi_j]_{j=1}^{t}$.
4. Output π_k .

Now, by assumption, there exists an extractor $\mathcal{E}_{\mathcal{A}'}$ for PC_{m} that succeeds in extracting against any choice of $\mathcal{Q}'$ and $\mathcal{B}'$. In particular, it succeeds against $\mathcal{Q}'$ and $\mathcal{B}'$ constructed above. Because the extractor $\mathcal{E}_{\mathcal{A}} := \mathcal{E}_{\mathcal{A}'}$ fails only if $\mathcal{E}_{\mathcal{A}'}$ fails, we need only to analyze the probability with which this latter algorithm fails. We know that whenever $\mathcal{E}_{\mathcal{A}}$ fails, $\mathsf{PC.Check}$ accepts, but there exists $k \in [t]$ such that

$$\deg(\boldsymbol{p}_k) \not\leq \boldsymbol{d}_k \quad \vee \quad \boldsymbol{v}_k \neq \boldsymbol{p}_k(z_k) \ .$$

By construction of $\mathsf{PC}.\mathsf{Check}$, this means that the k-th invocation of $\mathsf{PC}_\mathsf{m}.\mathsf{Check}$ succeeds, but the corresponding polynomials are either of incorrect degree or have incorrect claimed evaluations. Because $\mathcal{Q}'$ selects this index k with probability $1/t$, $\mathcal{A}'$ and $\mathcal{B}'$ break extractability of PC_m with non-negligible probability $\mu(\lambda)/t$, thus contradicting our assumption. □

2.13.3 Hiding

Theorem 2.13.2. *If* PC_m *in Section 2.12 achieves hiding (Definition 2.6.4) then* PC *also achieves hiding (Definition 2.6.4).*

Proof. We achieve this by constructing a simulator $\mathcal{S}_{\mathsf{PC}}$ for PC using the PC_m simulator $\mathcal{S}_\mathsf{m}$. $\mathcal{S}_{\mathsf{PC}}.\mathsf{Setup}$, $\mathcal{S}_{\mathsf{PC}}.\mathsf{Trim}$, and $\mathcal{S}_{\mathsf{PC}}.\mathsf{Commit}$ are the same as $\mathcal{S}_\mathsf{m}.\mathsf{Setup}$, $\mathcal{S}_\mathsf{m}.\mathsf{Trim}$, and $\mathcal{S}_\mathsf{m}.\mathsf{Commit}$, and so we focus on constructing $\mathcal{S}_{\mathsf{PC}}.\mathsf{Open}$.

$\mathcal{S}_{\mathsf{PC}}.\mathsf{Open}(\mathsf{trap}, \boldsymbol{p}, \boldsymbol{v}, \boldsymbol{d}, Q_i, \xi_i; \boldsymbol{\omega})$:

1. Parse Q as $\cup_{i\in[t]} T_j \times \{z_i\}$, for some $T_i \subseteq [n]$ and $z_i \in \mathbb{F}$, where each z_i is distinct.
2. Divide $\boldsymbol{p}$, $\boldsymbol{d}$, and $\boldsymbol{\omega}$ into $[\boldsymbol{p}_i]_{i=1}^t$, $[\boldsymbol{d}_i]_{i=1}^t$, and $[\boldsymbol{\omega}_i]_{i=1}^t$ so that $\boldsymbol{p}_i = [p_j]_{j\in T_i}$, $\boldsymbol{d}_i = [d_j]_{j\in T_i}$, and $\boldsymbol{\omega}_i = [\omega_j]_{j\in T_i}$.
3. For each i in $1, \ldots, t$, compute evaluation proof $\pi_i \leftarrow \mathcal{S}_\mathsf{m}.\mathsf{Open}(\mathsf{trap}, \boldsymbol{p}_i, \boldsymbol{v}_i, Q_i := T_i \times \{z_i\}, \xi; \boldsymbol{\omega}_i)$.
4. Output $\pi_i := [\pi_i]_{i=1}^t$.

The simulator $\mathcal{S}_{\mathsf{PC}}$ is a simple wrapper around the simulator $\mathcal{S}_\mathsf{m}$. Since $\mathcal{S}_\mathsf{m}$ achieves perfect hiding, so does $\mathcal{S}_{\mathsf{PC}}$ regardless of τ, the number of query sets Q_i, or t_i, the respective number of distinct points in each query set. □

Chapter 3

Proof-carrying Data without Succinct Arguments

Proof-carrying data (PCD) is a powerful cryptographic primitive that enables mutually distrustful parties to perform distributed computations that run indefinitely. Known approaches to construct PCD are based on succinct non-interactive arguments of knowledge (SNARKs) that have a succinct verifier or a succinct accumulation scheme.

In this chapter, we show how to obtain PCD *without relying on SNARKs*: we construct a PCD scheme given any non-interactive argument of knowledge (e.g., with linear-size arguments) that has a *split accumulation scheme*, which is a weak form of accumulation that we introduce.

Moreover, we construct a transparent non-interactive argument of knowledge for R1CS whose split accumulation is verifiable via a (small) *constant number of group and field operations*. Our construction is proved secure in the random oracle model based on the hardness of discrete logarithms, and it leads, via the random oracle heuristic and our result above, to concrete efficiency improvements for PCD.

Along the way, we construct a split accumulation scheme for Hadamard products under Pedersen commitments and for a simple polynomial commitment scheme based on Pedersen commitments.

Our results are supported by a modular and efficient implementation.

This work was previously published in [BCLMS21].

3.1 Introduction

Proof-carrying data (PCD) [CT10] is a powerful cryptographic primitive that enables mutually distrustful parties to perform distributed computations that run indefinitely, while ensuring that the correctness of every intermediate state of the computation can be verified efficiently. A special case of PCD is *incrementally-verifiable computation* (IVC) [Val08]. PCD has found applications in enforcing language semantics [CTV13], verifiable MapReduce computations [CTV15], image authentication [NT16], blockchains [Mina; KB20; BMRS20; CCDW20], and others. Given the theoretical and practical relevance of PCD, it is an important research question to build efficient PCD schemes from minimal cryptographic assumptions.

PCD from succinct verification. The canonical construction of PCD is via *recursive composition* of succinct non-interactive arguments (SNARGs) [BCCT13; BCTV14; COS20]. Informally, a proof that the computation was executed correctly for t steps consists of a proof of the claim "the t-th step of the computation was executed correctly, and there exists a proof that the computation was executed correctly for $t - 1$ steps". The latter part of the claim is expressed using the SNARG verifier itself. This construction yields secure PCD (with IVC as a special case) provided the SNARG satisfies an adaptive knowledge soundness property (i.e., is a SNARK). Efficiency requires the SNARK to have sublinear-time verification, achievable via SNARKs for machine computations [BCCT13] or preprocessing SNARKs for circuit computations [BCTV14; COS20].

Requiring sublinear-time verification, however, significantly restricts the choice of SNARK, which limits what is achievable for PCD. These restrictions have practical implications: the concrete efficiency of recursion is limited by the use of expensive curves for pairing-based SNARKs [BCTV14] or heavy use of cryptographic hash functions for hash-based SNARKs [COS20].

PCD from accumulation. Recently, [BCMS20] gave an alternative construction of PCD using SNARKs that have succinct *accumulation schemes*; this developed and formalized a novel approach for recursion sketched in [BGH19]. Informally, rather than being required to have sublinear-time verification, the SNARK is required to be accompanied by a cryptographic primitive that enables "postponing" the verification of SNARK proofs by way of an accumulator that is updated at each recursion step. The main efficiency requirement on the accumulation scheme is that the accumulation procedure must be succinctly verifiable, and in particular the accumulator itself must be succinct.

Requiring a SNARK to have a succinct accumulation scheme is a weaker condition than requiring it to have sublinear-time verification. This has enabled constructing PCD from SNARKs that do *not* have sublinear-time verification [BCMS20], which in turn led to PCD constructions from assumptions and with efficiency properties that were not previously achieved. Practitioners have exploited this freedom to design implementations of recursive composition with improved practical efficiency [Halo20; Pickles20].

Our motivation. The motivation of this work is twofold. First, can PCD be built from a weaker primitive than SNARKs with succinct accumulation schemes? If so, can we leverage this to obtain PCD constructions with improved *concrete* efficiency?

3.1.1 Contributions

We make theory and systems contributions that advance the state of the art for PCD: (1) We introduce *split accumulation schemes for relations*, a cryptographic primitive that relaxes prior notions of accumulation. (2) We obtain PCD from any non-interactive argument of knowledge that satisfies this weaker notion of accumulation; surprisingly, this allows for arguments with no succinctness whatsoever. (3) We construct a non-interactive argument of knowledge based on discrete logarithms (and random oracles) whose accumulation verifier has constant size (improving over the logarithmic-size verifier of prior accumulation schemes in this setting). (4) We implement and evaluate constructions from this work and from [BCMS20].

We elaborate on each of these contributions next.

(1) Split accumulation for relations. Recall from [BCMS20] that an accumulation scheme for a predicate $\Phi\colon X \to \{0,1\}$ enables proving/verifying that each input in an infinite stream $\mathsf{q}_1, \mathsf{q}_2, \ldots$ satisfies the predicate Φ, by augmenting the stream with *accumulators*. Informally, for each i, the prover produces a new accumulator acc_{i+1} from the input q_i and the old accumulator acc_i; the verifier can check that the triple $(\mathsf{q}_i, \mathsf{acc}_i, \mathsf{acc}_{i+1})$ is a valid accumulation step, much more efficiently than running Φ on q_i. At any time, the decider can validate acc_{i+1}, which establishes that for all $j \leq i$ it was the case that $\Phi(\mathsf{q}_j) = 1$. The accumulator size (and hence the running time of the three algorithms) cannot grow in the number of accumulation steps.

We extend this notion in two orthogonal ways. First we consider relations $\Phi\colon X \times W \to \{0,1\}$ and now for a stream of instances $\mathsf{qx}_1, \mathsf{qx}_2, \ldots$ the goal is to establish that there exist witnesses $\mathsf{qw}_1, \mathsf{qw}_2, \ldots$ such that $\Phi(\mathsf{qx}_i, \mathsf{qw}_i) = 1$ for each i. Second, we consider accumulators acc_i that are split into an instance part $\mathsf{acc}_i.\mathbb{x}$ and a witness part $\mathsf{acc}_i.\mathbb{w}$ with the restriction that the accumulation verifier only gets to see the instance part (and possibly an auxiliary accumulation proof pf). We refer to this notion as *split accumulation for relations*, and refer to (for contrast) the notion from [BCMS20] as *atomic accumulation for languages*.

The purpose of these extensions is to enable us to consider accumulation schemes in which predicate witnesses and accumulator witnesses are large while still requiring the accumulation verifier to be succinct (it receives short predicate instances and accumulator instances but not large witnesses). We will see that such accumulation schemes are both simpler and cheaper, while still being useful for primitives such as PCD.

See Section 3.2.1 for more on atomic vs. split accumulation, and Section 3.4 for formal definitions.

(2) PCD via split accumulation. A non-interactive argument has a split accumulation scheme if the relation corresponding to its verifier has a split accumulation scheme (we make this precise later). We show that any non-interactive argument of knowledge (NARK) having a split accumulation scheme where the *accumulation verifier* is sublinear can be used to build a proof-carrying data (PCD) scheme, *even if the NARK does not have sublinear argument size*. This significantly broadens the class of non-interactive arguments from which PCD can be built, and is the first result to obtain PCD from non-interactive arguments that need not be succinct. Similarly to [BCMS20], if the NARK and accumulation scheme are post-quantum secure, so is the PCD scheme. (It remains an open question whether there are non-trivial post-quantum instantiations of these.)

Theorem 3 (informal)**.** *There is an efficient transformation that compiles any NARK with a split accumulation scheme into a PCD scheme. If the NARK and its split accumulation scheme are zero knowledge, then the PCD scheme is also zero knowledge. Additionally, if the NARK and its accumulation scheme are post-quantum secure then the PCD scheme is also post-quantum secure.*

Similarly to all PCD results known to date, the above theorem holds in a model where all parties have access to a common reference string, *but no oracles*. (The construction makes non-black-box use of the accumulation scheme verifier, and the theorem does not carry over to the random oracle model.)

A corollary of Theorem 3 is that any NARK with a split accumulation scheme can be "bootstrapped" into a SNARK for machine computations. (PCD implies IVC and, further assuming collision-resistant hashing, also efficient SNARKs for machine computations [BCCT13].) This is surprising: an argument with decidedly weak efficiency properties implies an argument with succinct proofs and succinct verification!

See Section 3.2.2 for a summary of the ideas behind Theorem 3, and Section 3.5 for technical details.

(3) NARK with split accumulation based on DL. Theorem 3 motivates the question of whether we can leverage the weaker condition on the argument system to improve the efficiency of PCD. Our focus is on minimizing the cost of the accumulation verifier for the argument system, because it is the only component that is not used as a black box, and thus typically determines concrete efficiency. Towards this end, we present a (zero knowledge) NARK with (zero knowledge) split accumulation based on discrete logarithms, with a *constant-size* accumulation verifier; the NARK has a transparent (public-coin) setup.

Theorem 4 (informal)**.** *In the random oracle model and assuming the hardness of the discrete logarithm problem, there exists a transparent (zero knowledge) NARK for R1CS and a corresponding (zero knowledge) split accumulation scheme with the following efficiency:*

NARK			***split accumulation scheme***			
prover time	*verifier time*	*argument size*	*prover time*	*verifier time*	*decider time*	*accumulator size*
$O(\mathsf{M})\ \mathbb{G}$	$O(\mathsf{M})\ \mathbb{G}$	$O(1)\ \mathbb{G}$	$O(\mathsf{M})\ \mathbb{G}$	$O(1)\ \mathbb{G}$	$O(\mathsf{M})\ \mathbb{G}$	$\lvert\mathsf{acc}.\mathbb{x}\rvert = O(1)\ \mathbb{G} + O(1)\ \mathbb{F}$
$O(\mathsf{M})\ \mathbb{F}$	$O(\mathsf{M})\ \mathbb{F}$	$O(\mathsf{M})\ \mathbb{F}$	$O(\mathsf{M})\ \mathbb{F}$	$O(1)\ \mathbb{F}$	$O(\mathsf{M})\ \mathbb{F}$	$\lvert\mathsf{acc}.\mathbb{w}\rvert = O(\mathsf{M})\ \mathbb{F}$

Above, M *denotes the number of constraints in the R1CS instance,* $\mathbb{G}$ *denotes group scalar multiplications or group elements, and* $\mathbb{F}$ *denotes field operations or field elements.*

The NARK construction from Theorem 4 is particularly simple: it is obtained by applying the Fiat–Shamir transformation to a sigma protocol for R1CS based on Pedersen commitments (and linear argument size). The only "special" feature about the construction is that, as we prove, it has a very efficient split accumulation scheme for the relation corresponding to its verifier. By heuristically instantiating the random oracle, we can apply Theorem 3 (and [BCCT13]) to obtain a SNARK for machines from this modest starting point.

We find it informative to compare Theorem 4 and SNARKs with atomic accumulation based on discrete logarithms [BCMS20]:

- the SNARK's argument size is $O(\log \mathsf{M})$ group elements, *much less* than the NARK's $O(\mathsf{M})$ field elements;
- the SNARK's accumulator verifier uses $O(\log \mathsf{M})$ group scalar multiplications and field operations, *much more* than the NARK's $O(1)$ group scalar multiplications and field operations.

Therefore Theorem 4 offers a tradeoff that minimizes the cost of the accumulator at the expense of argument size. (As we shall see later, this tradeoff has concrete efficiency advantages.)

Our focus on argument systems based on discrete logarithms is motivated by the fact that they can be instantiated based on efficient curves suitable for recursion: the Tweedle [BGH19] or Pasta [Hop20] curve cycles, which follow the curve cycle technique for efficient recursion [BCTV14]. (In fact, as our construction does not rely on any number-theoretic properties of $|\mathbb{G}|$, we could even use the $(\texttt{secp256k1}, \texttt{secq256k1})$ cycle, where `secp256k1` is the curve used in Bitcoin.) This focus on discrete logarithms is a choice made for this work, and we believe that our ideas can lead to efficiency improvements to recursion in other settings (e.g., pairing-based and hash-based arguments) and leave these to future work.

See Section 3.2.3 for a summary of the ideas behind Theorem 3, and Section 3.8 for technical details.

(4) Split accumulation for common predicates. We obtain split accumulation schemes with constant-size accumulation verifiers for common predicates: (i) *Hadamard products (and more generally any bilinear function) under Pedersen commitments* (see Section 3.2.5 for a summary and Section 3.7 for details); (ii) *polynomial evaluations under Pedersen commitments* (see Section 3.2.6 for a summary and Section 3.11 for technical details). Split accumulation for Hadamard products is a building block that we use to prove Theorem 3.

(5) Implementation and evaluation. We contribute a set of Rust libraries that realize PCD via accumulation via modular combinations of interchangeable components: (a) generic interfaces for atomic and split accumulation; (b) generic construction of PCD from arguments with atomic and split accumulation; (c) split accumulation for our zkNARK for R1CS; (d) split accumulation for Hadamard products under Pedersen commitments; (e) split accumulation for polynomial evaluations under Pedersen commitments; (f) atomic accumulation for polynomial commitments based on inner product arguments and pairings from [BCMS20]; (g) constraints for all the foregoing accumulation verifiers. Practitioners interested in PCD will find these libraries useful for prototyping and comparing different types of recursion (and, e.g., may help decide if current systems based on atomic recursion [Halo20; Pickles20] are better off via split recursion or not).

We additionally conduct experiments to evaluate our implementation. Our experiments focus on determining the *recursion threshold*, which informally is the number of constraints that need to be proved at each step of the recursion. Our evaluation demonstrates that, over curves from the popular "Pasta" cycle [Hop20], the recursion threshold for split accumulation of our NARK for R1CS is as low as $52{,}000$ constraints, which is at least $8.5\times$ cheaper than the cost of IVC constructed from atomic accumulation for discrete-logarithm-based protocols [BCMS20]. In fact, the recursion threshold is even lower than that for IVC constructed from prior state-of-the-art pairing-friendly SNARKs [Gro16]. While this comes at the expense of much larger proof sizes, this overhead is attractive for notable applications (e.g., incrementally-verifiable ledgers).

See Section 3.9 and Section 3.10 for more details on our implementation and evaluation, respectively.

Remark 3.1.1 (concurrent work)**.** A concurrent work [BDFG21] studies similar questions as this work. Below we summarize the similarities and the differences between the two works.

Similarities. Both works are motivated by the goal of reducing the cost of recursive arguments. The main object of study in [BDFG21] is additive polynomial commitment schemes (PC schemes), for which [BDFG21] considers different types of *aggregation schemes*: (1) *public* aggregation in [BDFG21] is closely related to atomic accumulation specialized to PC schemes from a prior work [BCMS20]; and (2) *private* aggregation in [BDFG21] is closely related to split accumulation specialized to PC schemes from this work. Moreover, the private aggregation scheme for additive PC schemes in [BDFG21] is similar to our split accumulation scheme for Pedersen PC schemes (overviewed in Section 3.2.6 and detailed in Section 3.11). The protocols differ in how efficiency depends on the n claims to aggregate/accumulate: the verifier in [BDFG21] uses $n + 1$ group scalar multiplications while ours uses $2n$. (Informally, [BDFG21] first randomly combines claims and then evaluates at a random point, while we first evaluate at a random point and then randomly combine claims.)

Differences. The two works develop distinct, and complementary, directions.

The focus of [BDFG21] is to design protocols for any additive PC scheme (and, even more generally, any PC scheme with a linear combination scheme), including the aforementioned private aggregation protocol and a compiler that endows a given PC scheme with zero knowledge.

In contrast, our focus is to formulate a definition of split accumulation for general relation predicates that (a) we demonstrate suffices to construct PCD, and (b) in the random oracle model, we can also demonstrably achieve via a split accumulation scheme based on Pedersen commitments. We emphasize that our definitions are materially different from the case of atomic accumulation in [BCMS20], and necessitate careful consideration of technicalities such as the flavor of adaptive knowledge soundness, which algorithms can be allowed to query oracles, and so on. Hence, we cannot simply rely on the existing foundations for atomic accumulation of [BCMS20] in order to infer the correct definitions and security reductions for split accumulation. Overall, our theoretical work enables us to achieve the first construction of PCD without succinct arguments, and also to obtain a novel NARK for R1CS with a constant-size accumulation verifier.

We stress that the treatment of accumulation at a higher level of abstraction than for PC schemes is essential to prove theorems about PCD. In particular, contrary to what is claimed as a theorem in [BDFG21], it is *not* known how to build PCD from a PC scheme with an aggregation/accumulation scheme in any model without making additional heuristic assumptions. This is because obtaining a NARK from a PC scheme using known techniques requires the use of a random oracle, which we do not know how to accumulate. In contrast, we construct PCD in the standard model starting directly from an aggregation/accumulation scheme *for a NARK*, and *no additional assumptions*. Separately, the security of our accumulation scheme for a NARK in the standard model *is* an assumption, which is conjectured based on a security proof in the ROM.

Another major difference is that we additionally contribute a comprehensive and modular implementation of protocols from [BCMS20] and this work, and conduct an evaluation for the

discrete logarithm setting. This supports the asymptotic improvements with measured improvements in concrete efficiency.

3.2 Techniques

We summarize the main ideas behind our results. In Section 3.2.1 we discuss our new notion of split accumulation for relation predicates, and compare it with the notion of atomic accumulation for language predicates from [BCMS20]. In Section 3.2.2 we discuss the proof of Theorem 3. In Section 3.2.3 we discuss the proof of Theorem 4; for this we rely on a new result about split accumulation for Hadamard products, which we discuss in Section 3.2.5. Then, in Section 3.2.6, we discuss our split accumulation for a Pedersen-based polynomial commitment, which can act as a drop-in replacement for polynomial commitments used in prior SNARKs, such as those of [BGH19]. Finally, in Section 3.2.7 we elaborate on our implementation and evaluation. Figure 3.1 illustrates the relation between our results. The rest of the chapter contains technical details, and we provide pointers to relevant sections along the way.

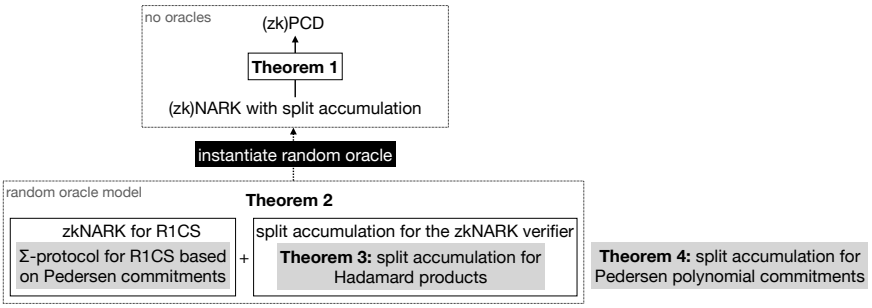

Figure 3.1: Diagram showing the relation between our results. Gray boxes within a result are notable subroutines.

3.2.1 Accumulation: atomic vs split

We review the notion of accumulation from [BCMS20], which we refer to as *atomic accumulation*, and then describe the weaker notion that we introduce, which we call *split accumulation*.

Atomic accumulation for languages. An *accumulation scheme for a language predicate* $\Phi\colon X \to \{0,1\}$ is a tuple of algorithms $(\mathrm{P}, \mathrm{V}, \mathrm{D})$, known as the prover, verifier, and decider, that enable proving/verifying statements of the form $\Phi(\mathsf{q}_1) \wedge \Phi(\mathsf{q}_2) \wedge \cdots$ more efficiently than running the predicate Φ on each input.

This is done as follows. Starting from an initial ("empty") accumulator acc_1, the prover is used to accumulate the first input q_1 to produce a new accumulator $\mathsf{acc}_2 \leftarrow \mathrm{P}(\mathsf{q}_1, \mathsf{acc}_1)$; then the prover is used again to accumulate the second input q_2 to produce a new accumulator $\mathsf{acc}_3 \leftarrow \mathrm{P}(\mathsf{q}_2, \mathsf{acc}_2)$; and so on.

Each accumulator produced so far enables efficient verification of the predicate on all inputs that went into the accumulator. For example, to establish that $\Phi(\mathsf{q}_1) \wedge \cdots \wedge \Phi(\mathsf{q}_T) = 1$ it suffices to check that:

- the verifier accepts each accumulation step: $\mathrm{V}(\mathsf{q}_1, \mathsf{acc}_1, \mathsf{acc}_2) = 1$, $\mathrm{V}(\mathsf{q}_2, \mathsf{acc}_2, \mathsf{acc}_3) = 1$, and so on; and
- the decider accepts the final accumulator: $\mathrm{D}(\mathsf{acc}_T) = 1$.

Qualitatively, this replaces the naive cost $T \cdot |\Phi|$ with the new cost $T \cdot |\mathrm{V}| + |\mathrm{D}|$. This is beneficial when the verifier is much cheaper than checking the predicate directly and the decider is not much costlier than checking the predicate directly. Crucially, the verifier and decider costs (and, in particular, the accumulator size) should not grow with the number T of accumulation steps (which need not be known in advance).

The properties of an accumulation scheme are summarized in the following informal definition, which additionally includes an accumulation proof used to check an accumulation step (but is not passed on).

Definition 3.2.1 (informal)**.** *An* **accumulation scheme** *for a predicate* $\Phi\colon X \to \{0,1\}$ *consists of a triple of algorithms* $(\mathrm{P}, \mathrm{V}, \mathrm{D})$*, known as the prover, verifier, and decider, that satisfies the following properties.*

- Completeness: *For every accumulator* acc *and predicate input* $\mathsf{q} \in X$*, if* $\mathrm{D}(\mathsf{acc}) = 1$ *and* $\Phi(\mathsf{q}) = 1$*, then for* $(\mathsf{acc}^\star, \mathsf{pf}^\star) \leftarrow \mathrm{P}(\mathsf{acc}, \mathsf{q})$ *it holds that* $\mathrm{V}(\mathsf{q}, \mathsf{acc}, \mathsf{acc}^\star, \mathsf{pf}^\star) = 1$ *and* $\mathrm{D}(\mathsf{acc}^\star) = 1$.
- Soundness: *For every efficiently-generated old accumulator* acc*, predicate input* $\mathsf{q} \in X$*, new accumulator* $\mathsf{acc}^\star$*, and accumulation proof* $\mathsf{pf}^\star$*, if* $\mathrm{D}(\mathsf{acc}^\star) = 1$ *and* $\mathrm{V}(\mathsf{q}, \mathsf{acc}, \mathsf{acc}^\star, \mathsf{pf}^\star) = 1$ *then, with all but negligible probability,* $\Phi(\mathsf{q}) = 1$ *and* $\mathrm{D}(\mathsf{acc}) = 1$.

The above definition omits many details, such as the ability to accumulate multiple accumulators $[\mathsf{acc}_j]_{j=1}^m$ and multiple predicate inputs $[\mathsf{q}_i]_{i=1}^n$ in one step, the optional property of zero knowledge (enabled by the accumulation proof $\mathsf{pf}^\star$), the fact that $\mathrm{P}, \mathrm{V}, \mathrm{D}$ should receive keys $\mathsf{apk}, \mathsf{avk}, \mathsf{dk}$ generated by an indexer algorithm that receives the specification of Φ, and others. We refer the reader to [BCMS20] for more details.

The aspect that we wish to highlight here is the following: in order for the verifier to be much cheaper than the predicate ($|\mathrm{V}| \ll |\Phi|$) it must be that the accumulator itself is much smaller than the predicate ($|\mathsf{acc}| \ll |\Phi|$) because the verifier receives the accumulator as input. (And if the accumulator is accompanied by a validity proof pf then this proof must also be small.)

We refer to this setting as *atomic accumulation* because the entirety of the accumulator is treated as one short monolithic string. In contrast, in this work we consider a relaxation where this is not the case, and will enable us to obtain new instantiations that lead to new theoretical and practical results.

Split accumulation for relations. We propose a relaxed notion of accumulation: a *split accumulation scheme for a relation predicate* $\Phi\colon X \times W \to \{0,1\}$ is again a tuple of algorithms $(\mathrm{P}, \mathrm{V}, \mathrm{D})$ as before. Split accumulation differs from atomic accumulation in that: (a) an input to Φ consists of a short instance part qx and a (possibly) long witness part qw; (b) an accumulator acc is split into a short instance part $\mathsf{acc}.\mathbb{x}$ and a (possibly) long witness part $\mathsf{acc}.\mathbb{w}$; (c) the verifier only needs the short parts of inputs and accumulators to verify an accumulation step, along with a short validity proof instead of the long witness parts.

As before, the prover is used to accumulate a predicate input $\mathsf{q}_i = (\mathsf{qx}_i, \mathsf{qw}_i)$ into a prior accumulator acc_i to obtain a new accumulator and validity proof $(\mathsf{acc}_{i+1}, \mathsf{pf}_{i+1}) \leftarrow \mathrm{P}(\mathsf{q}_i, \mathsf{acc}_i)$. Different from before, however, we wish to establish that given instances $\mathsf{qx}_1, \ldots, \mathsf{qx}_T$ there exist (more precisely, a party knows) witnesses $\mathsf{qw}_1, \ldots, \mathsf{qw}_T$ such that $\Phi(\mathsf{qx}_1, \mathsf{qw}_1) \wedge \cdots \wedge \Phi(\mathsf{qx}_T, \mathsf{qw}_T) = 1$. For this it suffices to check that:

- the verifier accepts each accumulation step given the short instance: $\mathrm{V}(\mathsf{qx}_1, \mathsf{acc}_1.\mathbb{x}, \mathsf{acc}_2.\mathbb{x}, \mathsf{pf}_2) = 1$, $\mathrm{V}(\mathsf{qx}_2, \mathsf{acc}_2.\mathbb{x}, \mathsf{acc}_3.\mathbb{x}, \mathsf{pf}_3) = 1$, and so on; and
- the decider accepts the final accumulator (made of both the instance and witness): $\mathrm{D}(\mathsf{acc}_T) = 1$.

Again the naive cost $T \cdot |\Phi|$ is replaced with the new cost $T \cdot |\mathrm{V}| + |\mathrm{D}|$, but now it could be that an accumulator is, e.g., as large as $|\Phi|$; we only need the *instance part* of the accumulator (and predicate inputs) to be short.

The security property of a split accumulation scheme involves an extractor that outputs a long witness part from a short instance part and proof, and is reminiscent of the knowledge soundness of a succinct non-interactive argument. Turning this high level description into a working definition requires some care, however, and we view this as a contribution of this work.[1] Informally the security definition could be summarized as follows.

Definition 3.2.2 (informal)**.** *A* **split accumulation scheme** *for a predicate* $\Phi\colon X \times W \to \{0,1\}$ *consists of a triple of algorithms* $(\mathrm{P}, \mathrm{V}, \mathrm{D})$ *that satisfies the following properties.*

- Completeness: *For every accumulator* acc *and predicate input* $\mathsf{q} = (\mathsf{qx}, \mathsf{qw}) \in X \times W$*, if* $\mathrm{D}(\mathsf{acc}) = 1$ *and* $\Phi(\mathsf{q}) = 1$*, then for* $(\mathsf{acc}^\star, \mathsf{pf}^\star) \leftarrow \mathrm{P}(\mathsf{q}, \mathsf{acc})$ *it holds that* $\mathrm{V}(\mathsf{qx}, \mathsf{acc}.\mathbb{x}, \mathsf{acc}^\star.\mathbb{x}, \mathsf{pf}^\star) = 1$ *and* $\mathrm{D}(\mathsf{acc}^\star) = 1$.
- Knowledge: *For every efficiently-generated old accumulator instance* $\mathsf{acc}.\mathbb{x}$*, old input instance* qx*, accumulation proof* $\mathsf{pf}^\star$*, and new accumulator* $\mathsf{acc}^\star$*, if* $\mathrm{D}(\mathsf{acc}^\star) = 1$ *and* $\mathrm{V}(\mathsf{qx}, \mathsf{acc}.\mathbb{x}, \mathsf{acc}^\star.\mathbb{x}, \mathsf{pf}^\star) = 1$ *then, with all but negligible probability, an efficient extractor can find an old accumulator witness* $\mathsf{acc}.\mathbb{w}$ *and predicate witness* qw *such that* $\Phi(\mathsf{qx}, \mathsf{qw}) = 1$ *and* $\mathrm{D}((\mathsf{acc}.\mathbb{x}, \mathsf{acc}.\mathbb{w})) = 1$.

One can verify that split accumulation is indeed a relaxation of atomic accumulation: any atomic accumulation scheme is (trivially) a split accumulation scheme with empty witnesses. Crucially, however, a split accumulation scheme alleviates a major restriction of atomic accumulation, namely, that accumulators and predicate inputs have to be short.

See Section 3.4 for formal definitions for split accumulation.[2]

Next, in Section 3.2.2 we show that split accumulation suffices for recursive composition (which has surprising theoretical consequences) and then in Section 3.2.3 we present a NARK with split accumulation scheme based on discrete logarithms.

3.2.2 PCD from split accumulation

We summarize the main ideas behind Theorem 3, which obtains proof-carrying data (PCD) from any NARK that has a split accumulation scheme. To ease exposition, in this summary we focus on

[1] By "working definition" we mean a definition that we can provably fulfill under concrete hardness assumptions in the random oracle model, and, separately, that provably suffices for recursive composition in the plain model without random oracles.

[2] The definitions in Section 3.4 are stated for the ROM, and one can obtain the definitions for the standard model (no ROM) by simply omitting the random oracle. Jumping ahead, the definitions in the standard model are those that we use for constructing PCD, while the definitions in the ROM are those that we prove are satisfied by our constructions of accumulation schemes.

IVC, which can be viewed as the special case where a circuit F is repeatedly applied. That is, we wish to incrementally prove a claim of the form "$F^T(z_0) = z_T$" where F^T denotes F composed with itself T times.

Prior work: recursion via atomic accumulation. Our starting point is a theorem from [BCMS20] that obtains PCD from any SNARK that has an atomic accumulation scheme. The IVC construction implied by that theorem is roughly follows.

- The *IVC prover* receives a previous instance z_i, proof π_i, and accumulator acc_i; accumulates (z_i, π_i) with acc_i to obtain a new accumulator acc_{i+1} and accumulation proof pf_{i+1}; and generates a SNARK proof π_{i+1} of the following claim expressed as a circuit R (see Fig. 3.2, middle box): "$z_{i+1} = F(z_i)$, and there exist a SNARK proof π_i, accumulator acc_i, and accumulation proof pf_{i+1} such that the accumulation verifier accepts $((z_i, \pi_i), \mathsf{acc}_i, \mathsf{acc}_{i+1}, \mathsf{pf}_{i+1})$". The IVC proof for z_{i+1} is $(\pi_{i+1}, \mathsf{acc}_{i+1})$.
- The *IVC verifier* validates an IVC proof (π_i, acc_i) for z_i by running the SNARK verifier on the instance (z_i, acc_i) and proof π_i, and running the accumulation scheme decider on the accumulator acc_i.

In each iteration we maintain the invariant that if acc_i is a valid accumulator (according to the decider) and π_i is a valid SNARK proof, then the computation is correct up to the i-th step.

Note that while it would suffice to prove that "$z_{i+1} = F(z_i)$, π_i is a valid SNARK proof, and acc_i is a valid accumulator", we cannot afford to do so. Indeed: (i) proving that π_i is a valid proof requires proving a statement about the argument verifier, which may not be sublinear; and (ii) proving that acc_i is a valid accumulator requires proving a statement about the decider, which may not be sublinear. Instead of proving this claim directly, we "defer" it by having the prover accumulate (z_i, π_i) into acc_i to obtain a new accumulator acc_{i+1}. The soundness property of the accumulation scheme ensures that if acc_{i+1} is valid and the accumulation verifier accepts $((z_i, \pi_i), \mathsf{acc}_i, \mathsf{acc}_{i+1}, \mathsf{pf}_{i+1})$, then π_i is a valid SNARK proof and acc_i is a valid accumulator. Thus all that remains to maintain the invariant is for the prover to prove that the accumulation verifier accepts; this is possible provided that the *accumulation verifier* is sublinear.

Our construction: recursion via split accumulation. Our construction naturally extends the above idea to the setting of NARKs with split accumulation schemes. Indeed, the only difference to the above construction is that the proof π_{i+1} generated by the IVC prover is for the statement "$z_{i+1} = F(z_i)$, and there exist a NARK proof *instance* $\pi_i.\mathbb{x}$, an accumulator *instance* $\mathsf{acc}_i.\mathbb{x}$, and an accumulation proof pf_{i+1} such that the accumulation verifier accepts $((z_i, \pi_i.\mathbb{x}), \mathsf{acc}_i.\mathbb{x}, \mathsf{acc}_{i+1}.\mathbb{x}, \mathsf{pf}_{i+1})$", and accordingly the IVC verifier runs the NARK verifier on $((z_i, \mathsf{acc}_i.\mathbb{x}), \pi_i)$ (in addition to running the accumulation scheme decider on the accumulator acc_i). This is illustrated in Fig. 3.2 (lower box). Note that the circuit R itself is unchanged from the atomic case; the difference is in whether we pass the *entire* proof and accumulators or just the $\mathbb{x}$ part.

Proving that this relaxation yields a secure construction is more complex. Similar to prior work, the proof of security proceeds via a recursive extraction argument, as we explain next.

For an atomic accumulation scheme ([BCMS20]), one maintains the following extraction invariant: the i-th extractor outputs $(z_i, \pi_i, \mathsf{acc}_i)$ such that π_i is valid according to the SNARK, acc_i

is valid according to the decider, and $F^{T-i}(z_i) = z_T$. The T-th "extractor" is simply the malicious prover, and we can obtain the i-th extractor by applying the knowledge guarantee of the SNARK to the $(i+1)$-th extractor. That the invariant is maintained is implied by the soundness guarantee of the atomic accumulation scheme.

For a split accumulation scheme, we want to maintain the same extraction invariant; however, the extractor for the NARK will only yield $(z_i, \pi_i.\mathbb{x}, \mathsf{acc}_i.\mathbb{x})$, and not the corresponding witnesses. This is where we make use of the extraction property of the split accumulation scheme itself. Specifically, we interleave the knowledge guarantees of the NARK and accumulation scheme as follows: the i-th NARK extractor is obtained from the $(i+1)$-th accumulation extractor using the knowledge guarantee of the NARK, and the i-th accumulation extractor is obtained from the i-th NARK extractor using the knowledge guarantee of the accumulation scheme. We take the malicious prover to be the T-th accumulation extractor.

From sketch to proof. In Section 3.5, we give the formal details of our construction and a proof of correctness. In particular, we show how to construct PCD, a more general primitive than IVC. In the PCD setting, rather than each computation step having a single input z_i, it receives m inputs from different nodes. Proving correctness hence requires proving that *all* of these inputs were computed correctly. For our construction, this entails checking m proofs and m accumulators. To do this, we extend the definition of an accumulation scheme to allow accumulating multiple instance-proof pairs and multiple "old" accumulators.

We also note that the application to PCD leads to other definitional considerations, which are similar to those that have appeared in previous works [COS20; BCMS20]. In particular, the knowledge soundness guarantee for both the NARK *and* the accumulation scheme should be of the stronger "multi-instance witness-extended emulation with auxiliary input and output" type used in previous work. Additionally, the underlying construction of split accumulation achieves only expected polynomial-time extraction (in the ROM), and so the recursive extraction technique requires that we are able to extract from expected-time adversaries.

Remark 3.2.3 (knowledge soundness for PCD vs. IVC)**.** The proof of security for PCD extracts a transcript *one full layer at a time*. Since a layer consists of many nodes, each with an *independently-generated* proof and accumulator, a standard "single-instance" extraction guarantee is insufficient in general. However, in the special case of IVC, every layer consists of exactly one node, and so single-instance extraction does suffice.

Remark 3.2.4 (flavors of PCD)**.** The recent advances in PCD from accumulation achieve weaker efficiency guarantees than PCD from succinct verification, and formally these results are incomparable. (Starting from weaker assumptions they obtain weaker conclusions.) The essential feature that all these works achieve is that the efficiency of PCD algorithms is independent of the number of nodes in the PCD computation, which is how PCD is defined (see Section 3.3.2). That said, prior work on PCD from succinct verification [BCCT13; BCTV14; COS20] additionally guarantees that verifying a PCD proof is sublinear in a node's computation; and prior work on PCD from atomic accumulation [BCMS20] merely ensures that a PCD proof has size (but not necessarily verification

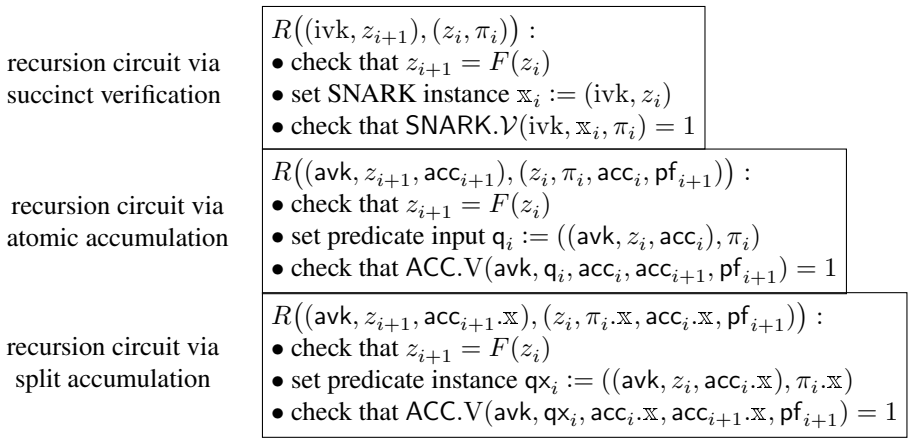

Figure 3.2: Comparison of circuits used to realize recursion with different techniques.

time) that is sublinear in a node's computation. The PCD scheme obtained in this work does not have these additional features: a PCD proof has size that is linear in a node's computation.

3.2.3 NARK with split accumulation based on DL

We summarize the main ideas behind Theorem 4, which provides, in the discrete logarithm setting with random oracles, a (zero knowledge) NARK for R1CS that has a (zero knowledge) split accumulation scheme whose accumulation verifier has constant size (more precisely, performs a constant number of group scalar multiplications, field operations, and random oracle calls).

Recall that R1CS is a standard generalization of arithmetic circuit satisfiability where the "circuit description" is given by coefficient matrices, as specified below. ("$\circ$" denotes the entry-wise product.)

Definition 3.2.5 (R1CS problem)**.** *Given a finite field $\mathbb{F}$, coefficient matrices $A, B, C \in \mathbb{F}^{\mathsf{M} \times \mathsf{N}}$, and an instance vector $x \in \mathbb{F}^{\mathsf{n}}$, is there a witness vector $w \in \mathbb{F}^{\mathsf{N}-\mathsf{n}}$ such that $Az \circ Bz = Cz$ for $z := (x, w) \in \mathbb{F}^{\mathsf{N}}$?*

We explain our construction incrementally. In Section 3.2.3.1 we begin by describing a NARK for R1CS that is *not* zero knowledge, and a "basic" split accumulation scheme for it that is also not zero knowledge. In Section 3.2.3.2 we show how to extend the NARK and its split accumulation scheme to both be zero knowledge. In Section 3.2.3.3 we explain why the accumulation scheme described so far is limited to the special case of 1 old accumulator and 1 predicate input (which suffices for IVC), and sketch how to obtain accumulation for m old accumulators and n predicate inputs (which is required for PCD); this motivates the problem of accumulating Hadamard products, which we subsequently address in Section 3.2.5.

We highlight here that both the NARK and the accumulation scheme are particularly simple compared to other protocols in the SNARK literature (especially with regard to constructions that enable recursion!), and view this as a significant advantage for potential deployments of these ideas in the real world.

3.2.3.1 Without zero knowledge

Let $\mathsf{ck} = (G_1, \ldots, G_{\mathsf{M}}) \in \mathbb{G}^{\mathsf{M}}$ be a commitment key for the Pedersen commitment scheme with message space $\mathbb{F}^{\mathsf{M}}$, and let $\mathsf{Commit}(\mathsf{ck}, a) := \sum_{i \in [\mathsf{M}]} a_i \cdot G_i$ denote its commitment function. Consider the following non-interactive argument for R1CS:

$$
\begin{array}{ll}
\mathcal{P}\big(\mathsf{ck}, (A,B,C), x, w\big) & \\
z := (x, w) \in \mathbb{F}^{\mathsf{N}} & \\
z_{_A} := Az \in \mathbb{F}^{\mathsf{M}} \quad C_{_A} := \mathsf{Commit}(\mathsf{ck}, z_{_A}) \in \mathbb{G} & \\
z_{_B} := Bz \in \mathbb{F}^{\mathsf{M}} \quad C_{_B} := \mathsf{Commit}(\mathsf{ck}, z_{_B}) \in \mathbb{G} & \\
z_{_C} := Cz \in \mathbb{F}^{\mathsf{M}} \quad C_{_C} := \mathsf{Commit}(\mathsf{ck}, z_{_C}) \in \mathbb{G} &
\end{array}
\quad \xrightarrow{\quad C_{_A}, C_{_B}, C_{_C}, w \quad} \quad
\begin{array}{l}
\mathcal{V}\big(\mathsf{ck}, (A,B,C), x\big) \\
z := (x, w) \\
z_{_A} := Az \quad C_{_A} \stackrel{?}{=} \mathsf{Commit}(\mathsf{ck}, z_{_A}) \\
z_{_B} := Bz \quad C_{_B} \stackrel{?}{=} \mathsf{Commit}(\mathsf{ck}, z_{_B}) \\
z_{_C} := Cz \quad C_{_C} \stackrel{?}{=} \mathsf{Commit}(\mathsf{ck}, z_{_C}) \\
\qquad\qquad C_{_C} \stackrel{?}{=} \mathsf{Commit}(\mathsf{ck}, z_{_A} \circ z_{_B})
\end{array}
$$

The NARK's security follows from the binding property of Pedersen commitments. (At this point we are not using any homomorphic properties, but we will in the accumulation scheme.) Moreover, denoting by $\mathsf{K} = \Omega(\mathsf{M})$ the number of non-zero entries in the coefficient matrices, the NARK's efficiency is as follows:

NARK prover time	NARK verifier time	NARK argument size
$O(\mathsf{M})\ \mathbb{G}$	$O(\mathsf{M})\ \mathbb{G}$	$O(1)\ \mathbb{G}$
$O(\mathsf{K})\ \mathbb{F}$	$O(\mathsf{K})\ \mathbb{F}$	$O(\mathsf{N})\ \mathbb{F}$

The NARK may superficially appear useless because it has linear argument size and is not zero knowledge. Nevertheless, we can obtain an efficient split accumulation scheme for it, as we describe next.[3]

The predicate to be accumulated is the NARK verifier with a suitable split between predicate instance and predicate witness: Φ takes as input a predicate instance $\mathsf{qx} = (x, C_{_A}, C_{_B}, C_{_C})$ and a predicate witness $\mathsf{qw} = w$, and then runs the NARK verifier with R1CS instance x and proof $\pi = (C_{_A}, C_{_B}, C_{_C}, w)$.[4]

An accumulator acc is split into an accumulator instance $\mathsf{acc}.\mathbb{x} = (x, C_{_A}, C_{_B}, C_{_C}, C_{\circ}) \in \mathbb{F}^{\mathsf{n}} \times \mathbb{G}^4$ and an accumulator witness $\mathsf{acc}.\mathbb{w} = w \in \mathbb{F}^{\mathsf{N}-\mathsf{n}}$. The accumulation decider D validates a split accumulator $\mathsf{acc} = (\mathsf{acc}.\mathbb{x}, \mathsf{acc}.\mathbb{w})$ as follows: set $z := (x, w) \in \mathbb{F}^{\mathsf{N}}$; compute the vectors $z_{_A} := Az$, $z_{_B} := Bz$, and $z_{_C} := Cz$; and check that the following conditions hold:

$$C_{_A} \stackrel{?}{=} \mathsf{Commit}(\mathsf{ck}, z_{_A}),\ C_{_B} \stackrel{?}{=} \mathsf{Commit}(\mathsf{ck}, z_{_B}),\ C_{_C} \stackrel{?}{=} \mathsf{Commit}(\mathsf{ck}, z_{_C}),\ C_{\circ} \stackrel{?}{=} \mathsf{Commit}(\mathsf{ck}, z_{_A} \circ z_{_B}).$$

[3] We could even "re-arrange" computation between the NARK and the accumulation scheme, and simplify the NARK further to be the NP decider (the verifier receives just the witness w and checks that the R1CS condition holds). We do not do so because this does not lead to any savings in the accumulation verifier (the main efficiency metric of interest) and also because the current presentation more naturally leads to the zero knowledge variant described in Section 3.2.3.2. (We note that the foregoing rearrangement is a general transformation that does not preserve zero knowledge or succinctness of the given NARK.)

[4] For now we view the commitment key ck and coefficient matrices A, B, C as hardcoded in the accumulation predicate Φ; our definitions later handle this more precisely.

Note that the accumulation decider D is similar, *but not equal*, to the NARK verifier.

We are left to describe the accumulation prover and accumulation verifier. Both have access to a random oracle ρ. For adaptive security, queries to the random oracle should include a hash τ of the coefficient matrices A, B, C and instance size n, which can be precomputed in an offline phase. (Formally, this is done via the *indexer* algorithm of the accumulation scheme, which receives the coefficient matrices and instance size, performs all one-time computations such as deriving τ, and produces an accumulator proving key apk, an accumulator verification key avk, and a decision key dk for P, V, and D respectively.)

The intuition for accumulation is to set the new accumulator to be a random linear combination of the old accumulator and predicate input, and use the accumulation proof to collect cross terms that arise from the Hadamard product (a bilinear, not linear, operation). This naturally leads to the following simple construction.

$\mathrm{P}^{\rho_{\mathsf{AS}}}(\mathsf{acc}, (\mathsf{qx}, \mathsf{qw}))$:
1. $z_A := A \cdot (\mathsf{qx}.x, \mathsf{qw}.w)$, $z_B := B \cdot (\mathsf{qx}.x, \mathsf{qw}.w)$.
2. $z'_A := A \cdot (\mathsf{acc}.\mathbb{x}.x, \mathsf{acc}.\mathbb{w}.w)$, $z'_B := B \cdot (\mathsf{acc}.\mathbb{x}.x, \mathsf{acc}.\mathbb{w}.w)$.
3. $\mathsf{pf} := \mathsf{Commit}(\mathsf{ck}, z_A \circ z'_B + z'_A \circ z_B)$.
4. $\beta := \rho_{\mathsf{AS}}(\tau, \mathsf{acc}.\mathbb{x}, \mathsf{qx}, \mathsf{pf})$.
5. $\mathsf{acc}^\star.\mathbb{x}.x := \mathsf{acc}.\mathbb{x}.x + \beta \cdot \mathsf{qx}.x$.
6. $\mathsf{acc}^\star.\mathbb{x}.C_A := \mathsf{acc}.\mathbb{x}.C_A + \beta \cdot \mathsf{qx}.C_A$.
7. $\mathsf{acc}^\star.\mathbb{x}.C_B := \mathsf{acc}.\mathbb{x}.C_B + \beta \cdot \mathsf{qx}.C_B$.
8. $\mathsf{acc}^\star.\mathbb{x}.C_C := \mathsf{acc}.\mathbb{x}.C_C + \beta \cdot \mathsf{qx}.C_C$.
9. $\mathsf{acc}^\star.\mathbb{x}.C_\circ := \mathsf{acc}.\mathbb{x}.C_\circ + \beta \cdot \mathsf{pf} + \beta^2 \cdot \mathsf{qx}.C_C$.
10. $\mathsf{acc}^\star.\mathbb{w}.w := \mathsf{acc}.\mathbb{w}.w + \beta \cdot \mathsf{qw}.w$.
11. Output $(\mathsf{acc}^\star, \mathsf{pf})$.

$\mathrm{V}^{\rho_{\mathsf{AS}}}(\mathsf{acc}.\mathbb{x}, \mathsf{qx}, \mathsf{acc}^\star.\mathbb{x}, \mathsf{pf})$:
1. $\beta := \rho_{\mathsf{AS}}(\tau, \mathsf{acc}.\mathbb{x}, \mathsf{qx}, \mathsf{pf})$.
2. $\mathsf{acc}^\star.\mathbb{x}.x \stackrel{?}{=} \mathsf{acc}.\mathbb{x}.x + \beta \cdot \mathsf{qx}.x$.
3. $\mathsf{acc}^\star.\mathbb{x}.C_A \stackrel{?}{=} \mathsf{acc}.\mathbb{x}.C_A + \beta \cdot \mathsf{qx}.C_A$.
4. $\mathsf{acc}^\star.\mathbb{x}.C_B \stackrel{?}{=} \mathsf{acc}.\mathbb{x}.C_B + \beta \cdot \mathsf{qx}.C_B$.
5. $\mathsf{acc}^\star.\mathbb{x}.C_C \stackrel{?}{=} \mathsf{acc}.\mathbb{x}.C_C + \beta \cdot \mathsf{qx}.C_C$.
6. $\mathsf{acc}^\star.\mathbb{x}.C_\circ \stackrel{?}{=} \mathsf{acc}.\mathbb{x}.C_\circ + \beta \cdot \mathsf{pf} + \beta^2 \cdot \mathsf{qx}.C_C$.

The efficiency of the split accumulation scheme can be summarized by the following table:

accumulation prover time	accumulation verifier time	decider time	accumulator size
$O(\mathsf{M})\ \mathbb{G}$	$4\ \mathbb{G}$ [5]	$O(\mathsf{M})\ \mathbb{G}$	$\lvert\mathsf{acc}.\mathbb{x}\rvert = 4\ \mathbb{G} + \mathsf{n}\ \mathbb{F}$
$O(\mathsf{K})\ \mathbb{F}$	$O(\mathsf{n})\ \mathbb{F}$	$O(\mathsf{K})\ \mathbb{F}$	$\lvert\mathsf{acc}.\mathbb{w}\rvert = (\mathsf{N} - \mathsf{n})\ \mathbb{F}$
1 RO	1 RO	—	—

The key efficiency feature is that the accumulation verifier only performs 1 call to the random oracle, a constant number of group scalar multiplications, and field operations. (More precisely, the verifier makes n field operations, but this does not grow with circuit size and, more fundamentally, is inevitable because the accumulation verifier must receive the R1CS instance $x \in \mathbb{F}^{\mathsf{n}}$ as input.)

3.2.3.2 With zero knowledge

We explain how to add zero knowledge to the approach described in the previous section.

[5] The verifier performs 4 group scalar multiplication by computing $\beta \cdot \mathsf{qx}.C_C$ and then $\beta \cdot \mathsf{pf} + \beta^2 \cdot \mathsf{qx}.C_C = \beta \cdot (\mathsf{pf} + \beta \cdot \mathsf{qx}.C_C)$ via another group scalar multiplication. Further it is possible to combine C_A and C_B in one commitment in both the NARK and the accumulation scheme. This reduces the group scalar multiplications in the verifier to 3, and the accumulator size to $3\ \mathbb{G} + \mathsf{n}\ \mathbb{F}$.

First, we extend the NARK to additionally achieve zero knowledge. For this we construct a sigma protocol for R1CS based on Pedersen commitments, which is summarized in Figure 3.3; then we apply the Fiat–Shamir transformation to it to obtain a corresponding zkNARK for R1CS. Here the commitment key for the Pedersen commitment is $\mathsf{ck} := (G_1, \ldots, G_{\mathsf{M}}, H) \in \mathbb{G}^{\mathsf{M}+1}$, as we need a spare group element for the commitment randomness. The blue text in the figure represents the "diff" compared to the non-zero-knowledge version, and indeed if all such text were removed the protocol would collapse to the previous one.

Second, we extend the split accumulation scheme to accumulate the modified protocol for R1CS. Again the predicate being accumulated is the NARK verifier but now since the NARK verifier has changed so does the predicate. A zkNARK proof π now can be viewed as a pair (π_1, π_2) denoting the prover's commitment and response in the sigma protocol. Then the predicate Φ takes as input a predicate instance $\mathsf{qx} = (x, \pi_1) \in \mathbb{F}^{\mathsf{n}} \times \mathbb{G}^8$ and a predicate witness $\mathsf{qw} = \pi_2 \in \mathbb{F}^{\mathsf{N}-\mathsf{n}+4}$, and then runs the NARK verifier with R1CS instance x and proof $\pi = (\pi_1, \pi_2)$.

An accumulator acc is split into an accumulator instance $\mathsf{acc}.\mathbb{x} = (x, C_A, C_B, C_C, C_\circ) \in \mathbb{F}^{\mathsf{n}} \times \mathbb{G}^4$ (the same as before) and an accumulator witness $\mathsf{acc}.\mathbb{w} = (w, \sigma_A, \sigma_B, \sigma_C, \sigma_\circ) \in \mathbb{F}^{\mathsf{N}-\mathsf{n}+4}$. The decider is essentially the same as in Section 3.2.3.1, except that now the four commitments are computed using the corresponding randomness in $\mathsf{acc}.\mathbb{w}$.

The accumulation prover and accumulation verifier can be extended, in a straightforward way, to support the new zkSNARK protocol; we provide these in Figure 3.4, with text in blue to denote the "diff" to accumulate the zero knowledge features of the NARK and with text in red to denote the features to make accumulation itself zero knowledge. There we use ρ_{NARK} to denote the oracle used for the zkNARK for R1CS, which is obtained via the Fiat–Shamir transformation applied to a sigma protocol (as mentioned above); for adaptive security, the Fiat–Shamir query includes, in addition to π_1, a hash $\tau := \rho_{\mathsf{NARK}}(A, B, C, \mathsf{n})$ of the coefficient matrices and the R1CS input $x \in \mathbb{F}^{\mathsf{n}}$ (this means that the Fiat–Shamir query equals $(\tau, \mathsf{qx}) = (\tau, x, \pi_1)$).

Note that now the accumulation prover and accumulation verifier are each making 2 calls to the random oracle, rather than 1 as before, because they have to additionally compute the sigma protocol's challenge.

3.2.3.3 Towards general accumulation

The accumulation schemes described in Sections 3.2.3.1 and 3.2.3.2 are limited to a special case, which we could call the "IVC setting", where accumulation involves 1 old accumulator and 1 predicate input. However, the definition of accumulation requires supporting m old accumulators $[\mathsf{acc}_j]_{j=1}^m = [(\mathsf{acc}_j.\mathbb{x}, \mathsf{acc}_j.\mathbb{w})]_{j=1}^m$ and n predicate inputs $[(\mathsf{qx}_i, \mathsf{qw}_i)]_{i=1}^n$, for any m and n. (E.g., to construct PCD we set both m and n equal to the "arity" of the compliance predicate.) How can we extend the ideas described so far to this more general case?

The zkNARK verifier performs two types of computations: linear checks and a Hadamard product check. We describe how to accumulate each of these in the general case.

- *Linear checks.* A split accumulator $\mathsf{acc} = (\mathsf{acc}.\mathbb{x}, \mathsf{acc}.\mathbb{w})$ in Section 3.2.3.2 included sub-accumulators for different linear checks: x, C_A, C_B, C_C in $\mathsf{acc}.\mathbb{x}$ and $w, \sigma_A, \sigma_B, \sigma_C$ in $\mathsf{acc}.\mathbb{w}$.

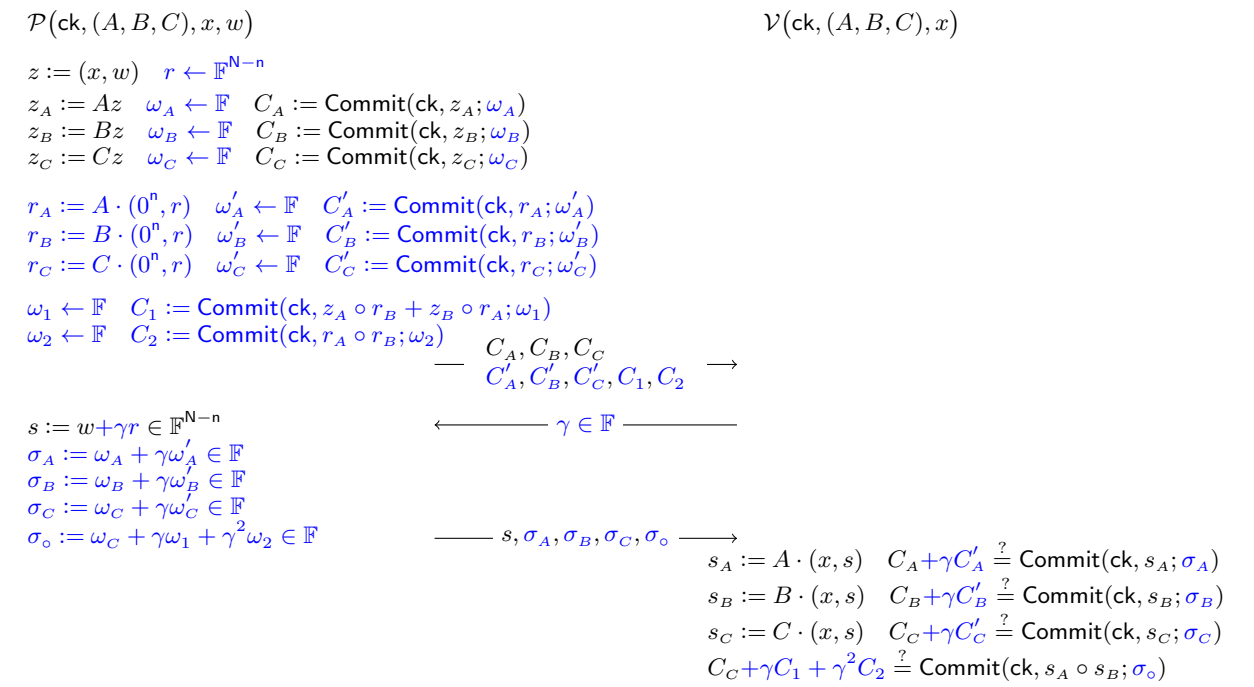

Figure 3.3: The sigma protocol for R1CS that underlies the zkNARK for R1CS.

$\mathrm{P}^{\rho_{\mathsf{AS}}}((\mathsf{qx},\mathsf{qw}),\mathsf{acc})$:
1. $z_A := A \cdot (\mathsf{qx}.x, \mathsf{qw}.s)$, $z_B := B \cdot (\mathsf{qx}.x, \mathsf{qw}.s)$.
2. $z'_A := A \cdot (\mathsf{acc}.\mathbb{x}.x, \mathsf{acc}.\mathbb{w}.s)$, $z'_B := B \cdot (\mathsf{acc}.\mathbb{x}.x, \mathsf{acc}.\mathbb{w}.s)$.
3. Sample $x^\star \leftarrow \mathbb{F}^{\mathsf{n}}$ and $s^\star \leftarrow \mathbb{F}^{\mathsf{N}-\mathsf{n}}$ and $\omega_2^\star \leftarrow \mathbb{F}$.
4. $s_A^\star := A \cdot (x^\star, s^\star)$, $s_B^\star := B \cdot (x^\star, s^\star)$, $s_C^\star := C \cdot (x^\star, s^\star)$.
5. $C_A^\star := \mathsf{Commit}(\mathsf{ck}, s_A^\star; \omega_A^\star)$ for $\omega_A^\star \leftarrow \mathbb{F}$.
6. $C_B^\star := \mathsf{Commit}(\mathsf{ck}, s_B^\star; \omega_B^\star)$ for $\omega_B^\star \leftarrow \mathbb{F}$.
7. $C_C^\star := \mathsf{Commit}(\mathsf{ck}, s_C^\star; \omega_C^\star)$ for $\omega_C^\star \leftarrow \mathbb{F}$.
8. $\mathsf{pf}_1 := \mathsf{Commit}(\mathsf{ck}, z_A \circ s_B^\star + s_A^\star \circ z_B; 0)$.
9. $\mathsf{pf}_2 := \mathsf{Commit}(\mathsf{ck}, s_A^\star \circ s_B^\star + z_A \circ z'_B + z'_A \circ z_B; \omega_2^\star)$.
10. $\mathsf{pf}_3 := \mathsf{Commit}(\mathsf{ck}, s_A^\star \circ z'_B + z'_A \circ s_B^\star; 0)$.
11. $\mathsf{pf} := (x^\star, C_A^\star, C_B^\star, C_C^\star, \mathsf{pf}_1, \mathsf{pf}_2, \mathsf{pf}_3)$.
12. $\beta := \rho_{\mathsf{AS}}(\tau, \mathsf{acc}.\mathbb{x}, \mathsf{qx}, \mathsf{pf})$.
13. Compute $\gamma := \rho_{\mathsf{NARK}}(\tau, \mathsf{qx})$.
14. $\mathsf{acc}^\star.\mathbb{x}.x := \mathsf{acc}.\mathbb{x}.x + \beta \cdot x^\star + \beta^2 \cdot \mathsf{qx}.x$.
15. $\mathsf{acc}^\star.\mathbb{x}.C_A := \mathsf{acc}.\mathbb{x}.C_A + \beta \cdot C_A^\star + \beta^2 \cdot (\mathsf{qx}.C_A + \gamma \cdot \mathsf{qx}.C'_A)$.
16. $\mathsf{acc}^\star.\mathbb{x}.C_B := \mathsf{acc}.\mathbb{x}.C_B + \beta \cdot C_B^\star + \beta^2 \cdot (\mathsf{qx}.C_B + \gamma \cdot \mathsf{qx}.C'_B)$.
17. $\mathsf{acc}^\star.\mathbb{x}.C_C := \mathsf{acc}.\mathbb{x}.C_C + \beta \cdot C_C^\star + \beta^2 \cdot (\mathsf{qx}.C_C + \gamma \cdot \mathsf{qx}.C'_C)$.
18. $\mathsf{acc}^\star.\mathbb{x}.C_\circ := \mathsf{acc}.\mathbb{x}.C_\circ + \beta \cdot \mathsf{pf}_1 + \beta^2 \cdot \mathsf{pf}_2 + \beta^3 \cdot \mathsf{pf}_3 + \beta^4 \cdot (\mathsf{qx}.C_C + \gamma \cdot C_1 + \gamma^2 \cdot C_2)$.
19. $\mathsf{acc}^\star.\mathbb{w}.s := \mathsf{acc}.\mathbb{w}.s + \beta \cdot s^\star + \beta^2 \cdot \mathsf{qw}.s$.
20. $\mathsf{acc}^\star.\mathbb{w}.\sigma_A := \mathsf{acc}.\mathbb{w}.\sigma_A + \beta \cdot \omega_A^\star + \beta^2 \cdot \mathsf{qw}.\sigma_A$.
21. $\mathsf{acc}^\star.\mathbb{w}.\sigma_B := \mathsf{acc}.\mathbb{w}.\sigma_B + \beta \cdot \omega_B^\star + \beta^2 \cdot \mathsf{qw}.\sigma_B$.
22. $\mathsf{acc}^\star.\mathbb{w}.\sigma_C := \mathsf{acc}.\mathbb{w}.\sigma_C + \beta \cdot \omega_C^\star + \beta^2 \cdot \mathsf{qw}.\sigma_C$.
23. $\mathsf{acc}^\star.\mathbb{w}.\sigma_\circ := \mathsf{acc}.\mathbb{w}.\sigma_\circ + \beta^2 \cdot \omega_2^\star + \beta^4 \cdot \mathsf{qw}.\sigma_\circ$.
24. Output $(\mathsf{acc}^\star, \mathsf{pf})$.

$\mathrm{V}^{\rho_{\mathsf{AS}}}(\mathsf{qx}, \mathsf{acc}.\mathbb{x}, \mathsf{acc}^\star.\mathbb{x}, \mathsf{pf})$:
1. $\beta := \rho_{\mathsf{AS}}(\tau, \mathsf{acc}.\mathbb{x}, \mathsf{qx}, \mathsf{pf})$.
2. $\gamma := \rho_{\mathsf{NARK}}(\tau, \mathsf{qx})$.
3. $\mathsf{acc}^\star.\mathbb{x}.x \stackrel{?}{=} \mathsf{acc}.\mathbb{x}.x + \beta x^\star + \beta^2 \cdot \mathsf{qx}.x$.
4. $\mathsf{acc}^\star.\mathbb{x}.C_A \stackrel{?}{=} \mathsf{acc}.\mathbb{x}.C_A + \beta C_A^\star + \beta^2 \cdot (\mathsf{qx}.C_A + \gamma \cdot \mathsf{qx}.C'_A)$.
5. $\mathsf{acc}^\star.\mathbb{x}.C_B \stackrel{?}{=} \mathsf{acc}.\mathbb{x}.C_B + \beta C_B^\star + \beta^2 \cdot (\mathsf{qx}.C_B + \gamma \cdot \mathsf{qx}.C'_B)$.
6. $\mathsf{acc}^\star.\mathbb{x}.C_C \stackrel{?}{=} \mathsf{acc}.\mathbb{x}.C_C + \beta C_C^\star + \beta^2 \cdot (\mathsf{qx}.C_C + \gamma \cdot \mathsf{qx}.C'_C)$.
7. $\mathsf{acc}^\star.\mathbb{x}.C_\circ \stackrel{?}{=} \mathsf{acc}.\mathbb{x}.C_\circ + \beta \cdot \mathsf{pf}_1 + \beta^2 \cdot \mathsf{pf}_2 + \beta^3 \cdot \mathsf{pf}_3 + \beta^4 \cdot (\mathsf{qx}.C_C + \gamma \cdot C_1 + \gamma^2 \cdot C_2)$.

Figure 3.4: Accumulation prover and accumulation verifier for the zkNARK for R1CS.

We can keep these components and simply use more random coefficients or, as we do, further powers of the element β. For example, in the accumulation prover P a computation such as $\mathsf{acc}^\star.\mathbb{x}.x := \mathsf{acc}.\mathbb{x}.x + \beta \cdot \mathsf{qx}.x$ is replaced by a computation such as $\mathsf{acc}^\star.\mathbb{x}.x := \sum_{j=1}^{m} \beta^{j-1} \cdot \mathsf{acc}_j.\mathbb{x}.x + \sum_{i=1}^{n} \beta^{m+j-1} \cdot \mathsf{qx}_i.x$.

- *Hadamard product check.* A split accumulator $\mathsf{acc} = (\mathsf{acc}.\mathbb{x}, \mathsf{acc}.\mathbb{w})$ in Section 3.2.3.2 also included a sub-accumulator for the Hadamard product check: $C_\circ$ in $\mathsf{acc}.\mathbb{x}$ and $\sigma_\circ$ in $\mathsf{acc}.\mathbb{w}$. Because a Hadamard product is a *bi*linear operation, combining two Hadamard products via a random coefficient led to a quadratic polynomial whose coefficients include the two original Hadamard products and a cross term. This is indeed why we stored the cross term in the accumulation proof pf. However, if we consider the cross terms that arise from combining more than two Hadamard products (i.e., when $m + n > 2$) then the corresponding polynomials do not lend themselves to accumulation because the original Hadamard products appear together with other cross terms. To handle this issue, we introduce in Section 3.2.5 a new subroutine that accumulates Hadamard products via an additional round of interaction.

We work out, and prove secure, the above ideas in full generality in Section 3.8.

3.2.4 On proving knowledge soundness

In order to construct accumulation schemes that fulfill the type of knowledge soundness that we ultimately need for PCD (see Section 3.2.2), we formulate a new *expected-time forking lemma in the random oracle model*, which is informally stated below. In our setting, $(\mathsf{q}, \mathfrak{b}, \mathsf{o}) \in L$ if $\mathsf{o} = ([\mathsf{qx}_i]_{i=1}^n, \mathsf{acc}, \mathsf{pf})$ is such that $\mathrm{D}(\mathsf{acc}) = 1$ and, given that $\rho(\mathsf{q}) = \mathfrak{b}$, the accumulation verifier accepts: $\mathrm{V}^\rho([\mathsf{qx}_i]_{i=1}^n, \mathsf{acc}.\mathbb{x}, \mathsf{pf}) = 1$.

Lemma 1 (informal)**.** *Let L be an efficiently recognizable set. There exists an algorithm* Fork *such that for every expected polynomial time algorithm A and integer $N \in \mathbb{N}$ the following holds. With all but negligible probability over the choice of random oracle ρ, randomness r of A, and randomness of* Fork*, if $A^\rho(r)$ outputs a tuple $(\mathsf{q}, \mathfrak{b}, \mathsf{o}) \in L$ with $\rho(\mathsf{q}) = \mathfrak{b}$, then $\mathsf{Fork}^{A,\rho}(1^N, \mathsf{q}, \mathfrak{b}, \mathsf{o}, r)$ outputs $[(\mathfrak{b}_j, \mathsf{o}_j)]_{j=1}^N$ such that $\mathfrak{b}_1, \ldots, \mathfrak{b}_N$ are pairwise distinct and for each $j \in [N]$ it holds that $(\mathsf{q}, \mathfrak{b}_j, \mathsf{o}_j) \in L$.*

This forking lemma differs from prior forking lemmas in three significant ways. First, it is in the random oracle model rather than the interactive setting (unlike [BCCGP16]). Second, we can obtain any polynomial number of accepting transcripts in expected polynomial time with only negligible loss in success probability (unlike forking lemmas for signature schemes, which typically extract two transcripts in strict polynomial time [BN06]). Finally, it holds even if the adversary itself runs in expected (as opposed to strict) polynomial time. This is important for our application to PCD where the extractor in one recursive step becomes the adversary in the next. This last feature requires some care, since the running time of the adversary, and in particular the length of its random tape, may not be bounded. For more details, see Section 3.6.2.

Moreover, in our security proofs we at times additionally rely on an expected-time variant of the *zero-finding game lemma* from [BCMS20] to show that if a particular polynomial equation holds at a point obtained from the random oracle via a "commitment" to the equation, then it must with overwhelming probability be a polynomial identity. For more details, see Section 3.11.2.

3.2.5 Split accumulation for Hadamard products

We construct a split accumulation scheme for a predicate Φ_{HP} that considers the Hadamard product of committed vectors. For a commitment key ck for messages in $\mathbb{F}^\ell$, the predicate Φ_{HP} takes as input a predicate instance $\mathsf{qx} = (C_1, C_2, C_3) \in \mathbb{G}^3$ consisting of three Pedersen commitments, a predicate witness $\mathsf{qw} = (a, b, \omega_1, \omega_2, \omega_3)$ consisting of two vectors $a, b \in \mathbb{F}^\ell$ and three opening randomness elements $\omega_1, \omega_2, \omega_3 \in \mathbb{F}$, and checks that $C_1 = \mathsf{CM.Commit}(\mathsf{ck}, a; \omega_1)$, $C_2 = \mathsf{CM.Commit}(\mathsf{ck}, b; \omega_2)$, and $C_3 = \mathsf{CM.Commit}(\mathsf{ck}, a \circ b; \omega_3)$. In other words, C_3 is a commitment to the Hadamard product of the vectors committed in C_1 and C_2.

Theorem 5 (informal)**.** *The Hadamard product predicate Φ_{HP} has a split accumulation scheme $\mathsf{AS}_{\mathsf{HP}}$ that is secure in the random oracle model (and assuming the hardness of the discrete logarithm problem) where verifying accumulation requires 5 group scalar multiplications and $O(1)$ field operations per claim, and results in an accumulator whose instance part is 3 group elements and witness part is $O(\ell)$ field elements. Moreover, the accumulation scheme can be made zero knowledge at a sub-constant overhead per claim.*

We formalize and prove this theorem in Section 3.7. Below we summarize the ideas behind this result. Our construction directly extends to accumulate any bilinear function (see Remark 3.2.6).

A bivariate identity. The accumulation scheme is based on a bivariate polynomial identity, and is the result of turning a public-coin two-round reduction into a non-interactive scheme by using the random oracle. Given n pairs of vectors $[(a_i, b_i)]_{i=1}^n$, consider the following two polynomials with coefficients in $\mathbb{F}^\ell$:

$$a(X, Y) := \sum_{i=1}^n X^{i-1} Y^{i-1} a_i \quad \text{and} \quad b(X) := \sum_{i=1}^n X^{n-i} b_i \ .$$

The Hadamard product of the two polynomials can be written as

$$a(X, Y) \circ b(X) = \sum_{i=1}^{2n-1} X^{i-1} t_i(Y) \quad \text{where} \quad t_n(Y) = \sum_{i=1}^n Y^{i-1} a_i \circ b_i \ .$$

The expression of the coefficient polynomials $\{t_i(Y)\}_{i \neq n}$ is not important; instead, the important aspect here is that a coefficient polynomial, namely $t_n(Y)$, includes the Hadamard products of all n pairs of vectors as different coefficients. This identity is the starting point of the accumulation scheme, which informally evaluates this expression at random points to reduce the n Hadamard products to 1 Hadamard product. Similar ideas are used to reduce several Hadamard products to a single inner product in [BCCGP16; BBBPWM18].

Batching Hadamard products. We describe a public-coin two-round reduction from n Hadamard product claims to 1 Hadamard product claim. The verifier receives n predicate instances

$[\mathsf{qx}_i]_{i=1}^n = [(C_{1,i}, C_{2,i}, C_{3,i})]_{i=1}^n$ each consisting of three Pedersen commitments, and the prover receives corresponding predicate witnesses $[\mathsf{qw}_i]_{i=1}^n = [(a_i, b_i, \omega_{1,i}, \omega_{2,i}, \omega_{3,i})]_{i=1}^n$ containing the corresponding openings.

- The verifier sends a first challenge $\mu \in \mathbb{F}$.
- The prover computes the product polynomial $a(X, \mu) \circ b(X) = \sum_{i=1}^{2n-1} X^{i-1} t_i(\mu) \in \mathbb{F}^{\ell}[X]$; for each $i \in [2n-1] \setminus \{n\}$, computes the commitment $C_{t,i} := \mathsf{CM.Commit}(\mathsf{ck}, t_i; 0) \in \mathbb{G}$; and sends to the verifier an accumulation proof $\mathsf{pf} := [C_{t,i}, C_{t,n+i}]_{i=1}^{n-1}$.
- The verifier sends a second challenge $\nu \in \mathbb{F}$.
- The verifier computes and outputs a new predicate instance $\mathsf{qx} = (C_1, C_2, C_3)$:

$$\begin{aligned} C_1 &= \textstyle\sum_{i=1}^{n} \nu^{i-1} \mu^{i-1} C_{1,i} \ , \\ C_2 &= \textstyle\sum_{i=1}^{n} \nu^{n-i} C_{2,i} \ , \\ C_3 &= \textstyle\sum_{i=1}^{n-1} \nu^{i-1} C_{t,i} + \nu^{n-1} \sum_{i=1}^{n} \mu^{i-1} C_{3,i} + \sum_{i=1}^{n-1} \nu^{n+i-1} C_{t,n+i} \ . \end{aligned}$$

- The prover computes and outputs a corresponding predicate witness $\mathsf{qw} = (a, b, \omega_1, \omega_2, \omega_3)$:

$$\begin{aligned} a &:= \textstyle\sum_{i=1}^{n} \nu^{i-1} \mu^{i-1} a_i & \omega_1 &:= \textstyle\sum_{i=1}^{n} \nu^{i-1} \mu^{i-1} \omega_{1,i} \ , \\ b &:= \textstyle\sum_{i=1}^{n} \nu^{n-i} b_i & \omega_2 &:= \textstyle\sum_{i=1}^{n} \nu^{n-i} \omega_{2,i} \ , \\ & & \omega_3 &:= \nu^{n-1} \textstyle\sum_{i=1}^{n} \mu^{i-1} \omega_{3,i} \ . \end{aligned}$$

Observe that the new predicate instance $\mathsf{qx} = (C_1, C_2, C_3)$ consists of commitments to $a(\nu, \mu)$, $b(\nu)$, and $a(\nu, \mu) \circ b(\nu)$, respectively, and the predicate witness $\mathsf{qw} = (a, b, \omega_1, \omega_2, \omega_3)$ consists of corresponding opening information. The properties of low-degree polynomials imply that if any of the n claims is incorrect (there is $i \in [n]$ such that $\Phi_{\mathsf{HP}}(\mathsf{qx}_i, \mathsf{qw}_i) = 0$) then, with high probability, so is the output claim ($\Phi_{\mathsf{HP}}(\mathsf{qx}, \mathsf{qw}) = 0$).

Split accumulation. The batching protocol described above yields a split accumulation scheme for Φ_{HP} in the random oracle model. An accumulator acc has the same form as a predicate input $(\mathsf{qx}, \mathsf{qw})$: $\mathsf{acc}.\mathbb{x}$ has the same form as a predicate instance qx, and $\mathsf{acc}.\mathbb{w}$ has the same form as a predicate witness qw. The accumulation decider D simply equals Φ_{HP} (this is well-defined due to the prior sentence). The accumulation prover and accumulation verifier are as follows.

- The accumulation prover P runs the interactive reduction by relying on the random oracle to generate the random verifier messages (i.e., it applies the Fiat–Shamir transformation to the reduction), in order to produce an accumulation proof pf as well as an accumulator $\mathsf{acc} = (\mathsf{qx}, \mathsf{qw})$ whose instance part is computed like the verifier of the reduction and witness part is computed like the prover of the reduction.

- The accumulation verifier V re-derives the challenges using the random oracle, and checks that qx was correctly derived from $[\mathsf{qx}_i]_{i=1}^n$ (also via the help of the accumulation proof pf).

The construction described above is not zero knowledge. One way to achieve zero knowledge is for the accumulation prover to sample a random predicate input that satisfies the predicate,

accumulate it, and include it as part of the accumulation proof pf. In our construction (detailed in Section 3.7), we opt for a more efficient solution, leveraging the fact that we are not actually interested in accumulating the random predicate input.

Efficiency. The efficiency claimed in Theorem 5 is evident from the construction. The (short) instance part of an accumulator consists of 3 group elements, while the (long) witness part of an accumulator consists of $O(\ell)$ field elements. The accumulator verifier V performs 2 random oracle calls, 5 group scalar multiplication, and $O(1)$ field operations per accumulated claim.

Security. Given an adversary that produces a list of Hadamard product claims $[\mathsf{qx}_i]_{i=1}^n = [(C_{1,i}, C_{2,i}, C_{3,i})]_{i=1}^n$, a single Hadamard product claim $\mathsf{qx} = (C_1, C_2, C_3)$ and corresponding witness $\mathsf{qw} = (a, b, \omega_1, \omega_2, \omega_3)$, and an accumulation proof pf that makes the accumulation verifier accept, we need to extract witnesses $[\mathsf{qw}_i]_{i=1}^n = [(a_i, b_i, \omega_{1,i}, \omega_{2,i}, \omega_{3,i})]_{i=1}^n$ for the instances $[\mathsf{qx}_i]_{i=1}^n$. Our security proof (in Section 3.7.2) works in the random oracle model, assuming hardness of the discrete logarithm problem.

In the proof we apply our expected-time forking lemma *twice* (see Section 3.2.4 for a discussion of this lemma and Section 3.6.2 for details including a corollary that summarizes its double invocation). This lets us construct a two-level tree of transcripts with branching factor n on the first challenge μ and branching factor $2n-1$ on the second challenge ν. Given such a transcript tree, the extractor works as follows:

1. Using the transcripts corresponding to challenges $\{(\mu_1, \nu_{1,k})\}_{k\in[n]}$ we extract ℓ-element vectors $[a_i]_{i=1}^n, [b_i]_{i=1}^n$ and field elements $[\omega_{1,i}]_{i=1}^n, [\omega_{2,i}]_{i=1}^n$ such that $[a_i]_{i=1}^n$ and $[b_i]_{i=1}^n$ are committed in $[C_{1,i}]_{i=1}^n$ and $[C_{2,i}]_{i=1}^n$ under randomness $[\omega_{1,i}]_{i=1}^n$ and $[\omega_{2,i}]_{i=1}^n$, respectively.
2. Define $a(X,Y) := \sum_{i=1}^n X^{i-1}Y^{i-1}a_i \in \mathbb{F}^\ell[X,Y]$ and $b(X) := \sum_{i=1}^n X^{n-i}b_i \in \mathbb{F}^\ell[X]$, using the vectors extracted above; then let $t_i(Y)$ be the coefficient of X^{i-1} in $a(X,Y) \circ b(X)$. For each $j \in [n]$, using the transcripts corresponding to challenges $\{(\mu_j, \nu_{j,k})\}_{k\in[2n-1]}$, we extract field elements $[\tau_i^{(j)}]_{i=1}^{2n-1}$ such that $t_n(\mu_j)$ is committed in $\sum_{i=1}^{n-1}\mu_j^{i-1}C_{3,i}$ under randomness $\tau_n^{(j)}$ and $[t_i(\mu_j), t_{n+i}(\mu_j)]_{i=1}^{n-1}$ are committed in $\mathsf{pf}^{(j)} := [C_{t,i}^{(j)}, C_{t,n+i}^{(j)}]_{i=1}^{n-1}$ under randomness $[\tau_i^{(j)}, \tau_{n+i}^{(j)}]_{i=1}^{n-1}$ respectively.
3. Compute the solution $[\omega_{3,i}]_{i=1}^n$ to the linear system $\{\tau_n^{(j)} = \sum_{i=1}^{n-1}\mu_j^{i-1}\omega_{3,i}\}_{j\in[n]}$. Together with the relation $\{t_n(\mu_j) = \sum_{i=1}^{n-1}\mu_j^{i-1}a_i \circ b_i\}_{j\in[n]}$, we deduce that $C_{3,i}$ is a commitment to $a_i \circ b_i$ under randomness $\omega_{3,i}$ for all $i \in [n]$.
4. For each $i \in [n]$, output $\mathsf{qw}_i := (a_i, b_i, \omega_{1,i}, \omega_{2,i}, \omega_{3,i})$.

Remark 3.2.6 (extension to any bilinear operation)**.** The ideas described above extend, in a straightforward way, to accumulating *any bilinear operation* of committed vectors. Let $f\colon \mathbb{F}^\ell \times \mathbb{F}^\ell \to \mathbb{F}^m$ be a bilinear operation, i.e., such that: (a) $f(a+a', b) = f(a,b)+f(a',b)$; (b) $f(a, b+b') = f(a,b)+f(a,b')$; (c) $\alpha \cdot f(a,b) = f(\alpha a, b) = f(a, \alpha b)$. Let Φ_f be the predicate that takes as input a predicate instance $\mathsf{qx} = (C_1, C_2, C_3) \in \mathbb{G}^3$ consisting of three Pedersen commitments, a predicate witness $\mathsf{qw} = (a, b, \omega_1, \omega_2, \omega_3)$ consisting of two vectors $a, b \in \mathbb{F}^\ell$ and three opening randomness elements $\omega_1, \omega_2, \omega_3 \in \mathbb{F}$, and checks that $C_1 = \mathsf{CM.Commit}(\mathsf{ck}_\ell, a; \omega_1)$, $C_2 = \mathsf{CM.Commit}(\mathsf{ck}_\ell, b; \omega_2)$, and $C_3 = \mathsf{CM.Commit}(\mathsf{ck}_m, f(a,b); \omega_3)$. The Hadamard product $\circ\colon \mathbb{F}^\ell \times \mathbb{F}^\ell \to \mathbb{F}^\ell$ is a bilinear opera-

tion, as is the scalar product $\langle \cdot, \cdot \rangle \colon \mathbb{F}^\ell \times \mathbb{F}^\ell \to \mathbb{F}$. Our accumulation scheme for Hadamard products works the same way, mutatis mutandis, for a general bilinear map f.

3.2.6 Split accumulation for Pedersen polynomial commitments

We construct an efficient split accumulation scheme $\mathsf{AS}_{\mathsf{PC}}$ for a predicate Φ_{PC} that checks a polynomial evaluation claim for a "trivial" polynomial commitment scheme $\mathsf{PC}_{\mathsf{Ped}}$ based on Pedersen commitments (see Fig. 3.5). In more detail, for a Pedersen commitment key ck for messages in $\mathbb{F}^{d+1}$, the predicate Φ_{PC} takes as input a predicate instance $\mathsf{qx} = (C, z, v) \in \mathbb{G} \times \mathbb{F} \times \mathbb{F}$ and a predicate witness $\mathsf{qw} = p \in \mathbb{F}^{\leq d}[X]$, and checks that $C = \mathsf{CM.Commit}(\mathsf{ck}, p)$, $p(z) = v$, and $\deg(p) \leq d$. In other words, the predicate Φ_{PC} checks that the polynomial p of degree at most d committed in C evaluates to v at z.

- *Setup:* On input $\lambda, D \in \mathbb{N}$, output $\mathsf{pp}_{\mathsf{CM}} \leftarrow \mathsf{CM.Setup}(1^\lambda, D+1)$.
- *Trim:* On input $\mathsf{pp}_{\mathsf{CM}}$ and $d \in \mathbb{N}$, check that $d \leq D$, set $\mathsf{ck} := \mathsf{CM.Trim}(\mathsf{pp}_{\mathsf{CM}}, d+1)$, and output $(\mathsf{ck}, \mathsf{rk} := \mathsf{ck})$.
- *Commit:* On input ck and $p \in \mathbb{F}[X]$ of degree at most $|\mathsf{ck}| - 1$, output $C \leftarrow \mathsf{CM.Commit}(\mathsf{ck}, p)$.
- *Open:* On input (ck, p, C, z), output $\pi := p$.
- *Check:* On input $(\mathsf{rk}, (C, z, v), \pi = p)$, check that $C = \mathsf{CM.Commit}(\mathsf{rk}, p)$, $p(z) = v$, and $\deg(p) < |\mathsf{rk}|$.

Completeness of $\mathsf{PC}_{\mathsf{Ped}}$ follows from that of CM, while extractability follows from the binding property of CM.

Figure 3.5: $\mathsf{PC}_{\mathsf{Ped}}$ is a trivial polynomial commitment scheme based on the Pedersen commitment scheme CM.

Theorem 6 (informal)**.** *The* **(Pedersen) polynomial commitment predicate** Φ_{PC} *has a split accumulation scheme* $\mathsf{AS}_{\mathsf{PC}}$ *that is secure in the random oracle model (and assuming the hardness of the discrete logarithm problem). Verifying accumulation requires* 2 *group scalar multiplications and* $O(1)$ *field additions/multiplications per claim, and results in an accumulator whose instance part is* 1 *group element and* 2 *field elements and whose witness part is* d *field elements. (See Table 3.1.)*

One can use $\mathsf{AS}_{\mathsf{PC}}$ to obtain a split accumulation scheme for a different NARK; see Remark 3.2.7 for details.

In Table 3.1 we compare the efficiency of our split accumulation scheme $\mathsf{AS}_{\mathsf{PC}}$ for the predicate Φ_{PC} with the efficiency of the atomic accumulation scheme $\mathsf{AS}_{\mathsf{IPA}}$ [BCMS20] for the equivalent predicate defined by the check algorithm of the (succinct) PC scheme $\mathsf{PC}_{\mathsf{IPA}}$ based on the inner-product argument on cyclic groups [BCCGP16; BBBPWM18; WTSTW18]. The takeaway is that the accumulation verifier for $\mathsf{AS}_{\mathsf{PC}}$ is significantly cheaper than the accumulation verifier for $\mathsf{AS}_{\mathsf{IPA}}$.

Technical details are in Section 3.11; in the rest of this section we sketch the ideas behind Theorem 6.

First we describe a simple public-coin interactive reduction for combining two or more evaluation claims into a single evaluation claim, and then explain how this interactive reduction gives rise to the

scheme	type	assumption	prover (per claim)	verifier (per claim)	decider	accumulator size: instance	accumulator size: witness
$\mathsf{AS}_{\mathsf{IPA}}$ [BCMS20]	atomic	DLOG + RO †	$O(\log d)$ $\mathbb{G}$ $O(d)$ $\mathbb{F}$ [$+O(d)$ $\mathbb{G}$ per acc.]	$O(\log d)$ $\mathbb{G}$ $O(\log d)$ $\mathbb{F}$ $O(\log d)$ RO	$O(d)$ $\mathbb{G}$ $O(d)$ $\mathbb{F}$	1 $\mathbb{G}$ $O(\log d)$ $\mathbb{F}$	0
$\mathsf{AS}_{\mathsf{PC}}$ [this work]	split	DLOG + RO	$O(d)$ $\mathbb{G}$ $O(d)$ $\mathbb{F}$	2 $\mathbb{G}$ $O(1)$ $\mathbb{F}$ 2 RO	$O(d)$ $\mathbb{G}$ $O(d)$ $\mathbb{F}$	1 $\mathbb{G}$ 2 $\mathbb{F}$	d $\mathbb{F}$

Table 3.1: Efficiency comparison between the atomic accumulation scheme $\mathsf{AS}_{\mathsf{IPA}}$ for $\mathsf{PC}_{\mathsf{IPA}}$ in [BCMS20] and the split accumulation scheme $\mathsf{AS}_{\mathsf{PC}}$ for $\mathsf{PC}_{\mathsf{Ped}}$ in this work. Above $\mathbb{G}$ denotes group scalar multiplications or group elements, and $\mathbb{F}$ denotes field operations or field elements. (†: $\mathsf{AS}_{\mathsf{IPA}}$ relies on knowledge soundness of $\mathsf{PC}_{\mathsf{IPA}}$, which results from applying the Fiat–Shamir transformation to a logarithmic-round protocol. The security of this protocol has only been proven via a superpolynomial-time extractor [BMMTV21] or in the algebraic group model [GT21].)

split accumulation scheme. We prove security in the random oracle model, using an expected-time extractor.

Batching evaluation claims. First consider two evaluation claims (C_1, z, v_1) and (C_2, z, v_2) for the *same* evaluation point z (and degree d). We can use a random challenge $\alpha \in \mathbb{F}$ to combine these claims into one claim (C', z, v') where $C' := C_1 + \alpha C_2$ and $v' := v_1 + \alpha v_2$. If either of the original claims does not hold then, with high probability over the choice of α, neither does the new claim. This idea extends to any number of claims for the same evaluation point, by taking $C' := \sum_i \alpha^i C_i$ and $v' := \sum_i \alpha^i v_i$.

Next consider two evaluation claims (C_1, z_1, v_1) and (C_2, z_2, v_2) at (possibly) different evaluation points z_1 and z_2. We explain how these can be combined into four claims all at the *same* point. Below we use the fact that $p(z) = v$ if and only if there exists a polynomial $w(X)$ such that $p(X) = w(X) \cdot (X - z) + v$.

Let $p_1(X)$ and $p_2(X)$ be the polynomials "inside" C_1 and C_2, respectively, that are known to the prover.

1. The prover computes the witness polynomials $w_1 := \frac{p_1(X) - v_1}{X - z_1}$ and $w_2 := \frac{p_2(X) - v_2}{X - z_2}$ and sends the commitments $W_1 := \mathsf{Commit}(w_1)$ and $W_2 := \mathsf{Commit}(w_2)$.

2. The verifier sends a random evaluation point $z^* \in \mathbb{F}$.

3. The prover computes and sends the evaluations $y_1 := p_1(z^*)$, $y_2 := p_2(z^*)$, $y_1' := w_1(z^*)$, and $y_2' := w_2(z^*)$.

4. The verifier checks the relation between each witness polynomial and the original polynomial at the random evaluation point z^*:

$$y_1 = y_1' \cdot (z^* - z_1) + y_1' \quad \text{and} \quad y_2 = y_2' \cdot (z^* - z_2) + y_2' \ .$$

Next, the verifier outputs four evaluation claims for $p_1(z^*) = y_1, p_2(z^*) = y_2, w_1(z^*) = y'_1, w_2(z^*) = y'_2$:

$$(C_1, z^*, y_1)\ ,\ (C_2, z^*, y_2)\ ,\ (W_1, z^*, y'_1)\ ,\ (W_2, z^*, y'_2)\ .$$

More generally, we can reduce m evaluation claims at m points to $2m$ evaluation claims all at the same point.

By combining the two techniques, one obtains a public-coin interactive reduction from any number of evaluation claims (regardless of evaluation points) to a single evaluation claim.

Split accumulation. The batching protocol described above yields a split accumulation scheme for Φ_{PC} in the random oracle model. An accumulator acc has the same form as a predicate input: the instance part is an evaluation claim and the witness part is a polynomial. Next we describe the algorithms of the accumulation scheme.

- The accumulation prover P runs the interactive reduction by relying on the random oracle to generate the random verifier messages (i.e., it applies the Fiat–Shamir transformation to the reduction), in order to combine the instance parts of old accumulators and inputs to obtain the instance part of a new accumulator. Then P also combines the committed polynomials using the same linear combinations in order to derive the new committed polynomial, which is the witness part of the new accumulator. The accumulation proof pf consists of the messages to the verifier in the reduction, which includes the commitments to the witness polynomials W_i and the evaluations y_i, y'_i at z^* of p_i, w_i (that is, $\mathsf{pf} := [(W_i, y_i, y'_i)]_{i=1}^n$).
- The accumulation verifier V checks that the challenges were correctly computed from the random oracle, and performs the checks of the reduction (the claims were correctly combined and that the proper relation between each y_i, y'_i, z_i, z^* holds).
- The accumulation decider D reads the accumulator in its entirety and checks that the polynomial (the witness part) satisfies the evaluation claim (the instance part). (Here the random oracle is not used.)

Efficiency. The efficiency claimed in Theorem 6 (and Table 3.1) is evident from the construction. The accumulation prover P computes $n + m$ commitments to polynomials when combining n old accumulators and m predicate inputs (all polynomials are for degree at most d). The (short) instance part of an accumulator consists of 1 group element and 2 field elements, while the (long) witness part of an accumulator consists of $O(d)$ field elements. The accumulator decider D computes 1 commitment (and 1 polynomial evaluation at 1 point) in order to validate an accumulator. Finally, the cost of running the accumulator verifier V is dominated by $2(n + m)$ scalar multiplication of the linear commitments.

Security. Given an adversary that produces evaluation claims $[\mathsf{qx}_i]_{i=1}^n = [(C_i, z_i, v_i)]_{i=1}^n$, a single claim $\mathsf{qx} = (C, z, v)$ and polynomial $\mathsf{qw} = s(X)$ with $s(z^*) = v$ to which C is a commitment, and accumulation proof pf that makes the accumulation verifier accept, we need to extract polynomials $[\mathsf{qw}_i]_{i=1}^n = [p_i(X)]_{i=1}^n$ with $p_i(z_i) = v_i$ to which C_i is a commitment. Our security proof (in

Section 3.11.3.1) works in the random oracle model, assuming hardness of the discrete logarithm problem.

In the proof, we apply our expected-time forking lemma (see Sections 3.2.4 and 3.6.2) to obtain $2n$ polynomials $[s^{(j)}]_{j=1}^{2n}$ for the same evaluation point z^* but distinct challenges α_j, where n is the number of evaluation claims. The checks in the reduction procedure imply that $s^{(j)}(X) = \sum_{i=1}^{n} \alpha_j^i p_i(X) + \sum_{i=1}^{n} \alpha_j^{n+i} w_i(X)$, where $w_i(X)$ is the witness corresponding to $p_i(X)$; hence we can recover the $p_i(X), w_i(X)$ by solving a linear system (given by the Vandermonde matrix in the challenges $[\alpha_j]_{j=1}^{2n}$). We then use an expected-time variant of the zero-finding game lemma from [BCMS20] (see Section 3.11.2) to show that if a particular polynomial equation on $p_i(X), w_i(X)$ holds at the point z^* obtained from the random oracle, it must with overwhelming probability be an identity. Applying this to the equation induced by the reduction shows that, with high probability, each extracted polynomial p_i satisfies the corresponding evaluation claim (C_i, z_i, v_i).

Remark 3.2.7 (from $\mathsf{PC}_{\mathsf{Ped}}$ to an accumulatable NARK)**.** If one replaced the (succinct) polynomial commitment scheme that underlies the preprocessing zkSNARK in [CHMMVW20] with the aforementioned (non-succinct) trivial Pedersen polynomial commitment scheme then (after some adjustments and using our Theorem 6) one would obtain a zkNARK for R1CS with a split accumulation scheme whose accumulation verifier *is* of constant size but other asymptotics would be worse compared to Theorem 4.

First, the cryptographic costs and the quasilinear costs of the NARK and accumulation scheme would also grow in the number K of non-zero entries in the coefficient matrices, which can be much larger than M and N (asymptotically and concretely). Second, the NARK prover would additionally use a quasilinear number of field operations due to FFTs. Finally, in addition to poorer asymptotics, this approach would lead to a concretely more expensive accumulation verifier and overall a more complex protocol. Nevertheless, one *can* design a concretely efficient zkNARK for R1CS based on the Pedersen PC scheme and our accumulation scheme for it. This naturally leads to an alternative construction to the one in Section 3.2.3 (which is instead based on accumulation of Hadamard products), and would lead to a slightly more expensive prover (which now would use FFTs) and a slightly cheaper accumulation verifier (a smaller number of group scalar multiplications). We leave this as an exercise for the interested reader.

3.2.7 Implementation and evaluation

We elaborate on our implementation and evaluation of accumulation schemes and their application to PCD.

The case for a PCD framework. Different PCD constructions offer different trade-offs. The tradeoffs are both about asymptotics (see Remark 3.2.4) and about practical concerns, as we review below.

- *PCD from sublinear verification* [BCCT13; BCTV14; COS20] is typically instantiated via preprocessing SNARKs based on pairings.[6] This route offers excellent verifier time (a few

[6] Instantiations based on hashes are also possible [COS20] but are (post-quantum and) less efficient.

milliseconds regardless of the computation at a PCD node), but requires a private-coin setup (which complicates deployment) and cycles of pairing-friendly elliptic curves (which are costly in terms of group arithmetic and size).
- *PCD from atomic accumulation* [BCMS20] can, e.g., be instantiated via SNARKs based on cyclic groups [BGH19]. This route offers a transparent setup (easy to deploy) and logarithmic-size arguments (a few kilobytes even for large computations), using cycles of standard elliptic curves (more efficient than their pairing-friendly counterparts). On the other hand, this route yields linear verification times (expensive for large computations) and logarithmic costs for accumulation (increasing the cost of recursion).
- *PCD from split accumulation* (this work) can, e.g., be instantiated via NARKs based on cyclic groups. This route still offers a transparent setup and allows using cycles of standard elliptic curves. Moreover, it offers constant costs for accumulation, but at the expense of argument size, which is now linear.

It would be desirable to have a *single framework that supports different PCD constructions via a modular composition of simpler building blocks*. Such a framework would enable a number of desirable features: (a) ease of replacing older building blocks with new ones; (b) ease of prototyping different PCD constructions for different applications (which may have different needs), thereby enabling practitioners to make informed choices about which PCD construction is best for them; (c) simpler and more efficient auditing of complex cryptographic systems with many intermixed layers. (Realizing even a single PCD construction is a substantial implementation task.); and (d) separation of "application" logic from the underlying recursion via a common PCD interface. Together, these features would enable further industrial deployment of PCD, as well as making future research and comparisons simpler.

Implementation (Section 3.9). The above considerations motivated our implementation efforts for PCD. Our code base has two main parts, one for realizing accumulation schemes and another for realizing PCD from accumulation (the latter is integrated with PCD from succinct verification under a unified PCD interface).

- *Framework for accumulation.* We designed a modular framework for (atomic and split) accumulation schemes, and use it to implement, under a common interface, several accumulation schemes: (a) the atomic accumulation scheme $\mathsf{AS}_{\mathsf{AGM}}$ in [BCMS20] for the PC scheme $\mathsf{PC}_{\mathsf{AGM}}$; (b) the atomic accumulation scheme $\mathsf{AS}_{\mathsf{IPA}}$ in [BCMS20] for the PC scheme $\mathsf{PC}_{\mathsf{IPA}}$; (c) the split accumulation scheme $\mathsf{AS}_{\mathsf{PC}}$ in this work for the PC scheme $\mathsf{PC}_{\mathsf{Ped}}$; (d) the split accumulation scheme $\mathsf{AS}_{\mathsf{HP}}$ in this work for the Hadamard product predicate Φ_{HP}; (e) the split accumulation scheme for our NARK for R1CS. Our framework also provides a generic method for defining R1CS constraints for the verifiers of these accumulation schemes; we leverage this to implement R1CS constraints for all of these accumulation schemes.

- *PCD from accumulation.* We use the foregoing framework to implement a generic construction of PCD from accumulation. We support the PCD construction of [BCMS20] (which uses atomic accumulation) and the PCD construction in this work (which uses split accumulation). Our code

builds on, and extends, an existing PCD library.[7] Our implementation is modular: it takes as ingredients an implementation of any NARK, an implementation of any accumulation scheme for that NARK, and constraints for the accumulation verifier, and produces a concrete PCD construction. This allows us, for example, to obtain a PCD instantiation based on our NARK for R1CS and its split accumulation scheme.

Evaluation for DL setting (Section 3.10). When realizing PCD in practice the main goal is to "minimize the cost of recursion", that is, to minimize the number of constraints that need to be recursively proved in each PCD step (excluding the constraints for the application) without hurting other parameters too much (prover time, argument size, and so on). We evaluate our implementation with respect to this goal, with a focus on understanding the trade-offs between atomic and split accumulation *in the discrete logarithm setting*.

The DL setting is of particular interest to practitioners, as it leads to systems with a transparent (public-coin) setup that can be based on efficient cycles of (standard) elliptic curves [BGH19; Hop20]; indeed, some projects are developing real-world systems that use PCD in the DL setting [Halo20; Pickles20]. The main drawback of the DL setting is that verification time (and sometimes argument size) is linear in a PCD node's computation. This inefficiency is, however, tolerable if a PCD node's computation is not too large, as is the case in the aforementioned projects. (Especially so when taking into account the disadvantages of PCD based on pairings, which involves relying on a private-coin setup and more expensive curve cycles.)

We evaluate our implementation to answer two questions: (a) how efficient is recursion with split accumulation for our simple zkNARK for R1CS? (b) what is the constraint cost of split accumulation for $\mathsf{PC}_{\mathsf{Ped}}$ compared to atomic accumulation for $\mathsf{PC}_{\mathsf{IPA}}$? All our experiments are performed over the 255-bit Pallas curve in the Pasta cycle of curves [Hop20], which is used by real-world deployments.

- *Split accumulation for R1CS.* Our evaluation demonstrates that the cost of recursion for IVC with our split accumulation scheme for the simple NARK for R1CS is low, both with zero knowledge ($\sim 99 \times 10^3$ constraints) and without ($\sim 52 \times 10^3$ constraints). In fact, this cost is even lower than the cost of IVC based on highly efficient pairing-based circuit-specific SNARKs. Furthermore, like in the pairing-based case, this cost does not grow with the size of computation being checked. This is much better than prior constructions of IVC based on atomic accumulation for $\mathsf{PC}_{\mathsf{IPA}}$ in the DL setting, as we will see next.

- *Comparison of accumulation for PC schemes.* Several (S)NARKs are built from PC schemes, and the primary cost of recursion for these is determined by the cost of accumulation for the PC scheme. In light of this we compare the costs of two accumulation schemes:
  - the atomic accumulation scheme for the PC scheme $\mathsf{PC}_{\mathsf{IPA}}$ [BCMS20];
  - the split accumulation scheme for $\mathsf{PC}_{\mathsf{Ped}}$ (Section 3.11).

  Our evaluation demonstrates that the constraint cost of the $\mathsf{AS}_{\mathsf{PC}}$ accumulation verifier is 8 to 20 times cheaper than that of the $\mathsf{AS}_{\mathsf{IPA}}$ accumulation verifier.

[7] `https://github.com/arkworks-rs/pcd`

We note that the cost of all the aforementioned accumulation schemes is dominated by the cost of many common subcomponents, and so improvements in these subcomponents will preserve the relative cost. For example, applying existing techniques [Halo20; Pickles20] for optimizing the constraint cost of elliptic curve scalar multiplications should benefit all our schemes in a similar way.

3.3 Preliminaries

Indexed relations. An *indexed relation* $\mathcal{R}$ is a set of triples $(\mathbb{i}, \mathbb{x}, \mathbb{w})$ where $\mathbb{i}$ is the index, $\mathbb{x}$ is the instance, and $\mathbb{w}$ is the witness; the corresponding *indexed language* $\mathcal{L}(\mathcal{R})$ is the set of pairs $(\mathbb{i}, \mathbb{x})$ for which there exists a witness $\mathbb{w}$ such that $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \in \mathcal{R}$. For example, the indexed relation of satisfiable boolean circuits consists of triples where $\mathbb{i}$ is the description of a boolean circuit, $\mathbb{x}$ is a partial assignment to its input wires, and $\mathbb{w}$ is an assignment to the remaining wires that makes the boolean circuit output 0.

Security parameters. For simplicity of notation, we assume that all public parameters have length at least λ, so that algorithms which receive such parameters can run in time $\mathrm{poly}(\lambda)$.

Random oracles. We denote by $\mathcal{U}(\lambda)$ the set of all functions that map $\{0,1\}^*$ to $\{0,1\}^\lambda$. We denote by $\mathcal{U}(*)$ the set $\bigcup_{\lambda\in\mathbb{N}} \mathcal{U}(\lambda)$. A *random oracle* with security parameter λ is a function $\rho\colon \{0,1\}^* \to \{0,1\}^\lambda$ sampled uniformly at random from $\mathcal{U}(\lambda)$.

Adversaries. All the definitions in this work should be taken to refer to non-uniform adversaries. An adversary (or extractor) running in *expected polynomial time* is then a Turing machine provided with a *polynomial-size* non-uniform advice string and access to an infinite random tape, whose expected running time for all choices of advice is polynomial. We sometimes write $(\mathsf{o}; r) \leftarrow A(x)$ when A is an expected polynomial-time algorithm, where o is A's output and r is the randomness used by A (i.e., up to the rightmost position of the head on the randomness tape). We also write $(\mathsf{o}, r') \leftarrow A(x; r)$, where r is a string of finite length: this denotes executing A with an infinite random tape with prefix r and r' is the randomness used by A (and in particular its prefix is consistent with r). Finally, we write $\mathsf{o} \leftarrow A(x; \sigma)$ where $\sigma \in \{0,1\}^*$ is an infinite string representing the entire random tape.

3.3.1 Non-interactive arguments in the ROM

A tuple of algorithms $\mathsf{ARG} = (\mathcal{G}, \mathcal{I}, \mathcal{P}, \mathcal{V})$ is a (preprocessing) *non-interactive argument* in the random oracle model (ROM) for an indexed relation family $\{\mathcal{R}_{\mathsf{pp}}\}_{\mathsf{pp}}$ if the following properties hold.

- **Completeness.** For every adversary $\mathcal{A}$,

$$\Pr\left[\begin{array}{c|r} (\mathbb{i}, \mathbb{x}, \mathbb{w}) \notin \mathcal{R}_{\mathsf{pp}} & \rho \leftarrow \mathcal{U}(\lambda) \\ \vee & \mathsf{pp} \leftarrow \mathcal{G}^\rho(1^\lambda) \\ \mathcal{V}^\rho(\mathrm{ivk}, \mathbb{x}, \pi) = 1 & (\mathbb{i}, \mathbb{x}, \mathbb{w}) \leftarrow \mathcal{A}^\rho(\mathsf{pp}) \\ & (\mathrm{ipk}, \mathrm{ivk}) \leftarrow \mathcal{I}^\rho(\mathsf{pp}, \mathbb{i}) \\ & \pi \leftarrow \mathcal{P}^\rho(\mathrm{ipk}, \mathbb{x}, \mathbb{w}) \end{array}\right] = 1 \ .$$

- **Soundness.** For every polynomial-size adversary $\tilde{\mathcal{P}}$,

$$\Pr\left[\begin{array}{c|r} (\mathbb{i}, \mathbb{x}) \notin \mathcal{L}(\mathcal{R}_{\mathsf{pp}}) & \rho \leftarrow \mathcal{U}(\lambda) \\ \wedge & \mathsf{pp} \leftarrow \mathcal{G}^\rho(1^\lambda) \\ \mathcal{V}^\rho(\mathrm{ivk}, \mathbb{x}, \pi) = 1 & (\mathbb{i}, \mathbb{x}, \pi) \leftarrow \tilde{\mathcal{P}}^\rho(\mathsf{pp}) \\ & (\mathrm{ipk}, \mathrm{ivk}) \leftarrow \mathcal{I}^\rho(\mathsf{pp}, \mathbb{i}) \end{array}\right] \leq \mathrm{negl}(\lambda) \ .$$

Completeness allows $(\mathbb{i}, \mathbb{x}, \mathbb{w})$ to depend adversarially on the random oracle ρ and public parameters pp; and soundness allows $(\mathbb{i}, \mathbb{x})$ to depend adversarially on the random oracle ρ and public parameters pp.

Our PCD construction makes use of the stronger property of *knowledge soundness*, and optionally also the property of (statistical) *zero knowledge*. We define both of these properties below.

We refer to an argument with knowledge soundness as a NARK (non-interactive argument of knowledge) whereas an argument that just satisfies soundness is a NARG.

Knowledge soundness. $\mathsf{ARG} = (\mathcal{G}, \mathcal{I}, \mathcal{P}, \mathcal{V})$ has *knowledge soundness* (with respect to auxiliary input distribution $\mathcal{D}$) if for every expected polynomial time adversary $\tilde{\mathcal{P}}$ there exists an expected polynomial time extractor $\mathcal{E}$ such that for every set Z,

$$\Pr\left[\begin{array}{c|r} (\mathsf{pp}, \mathsf{ai}, \vec{\mathbb{i}}, \vec{\mathbb{x}}, \mathsf{ao}) \in Z & \rho \leftarrow \mathcal{U}(\lambda) \\ \wedge \forall\, j \in [\ell]\,, (\mathbb{i}_j, \mathbb{x}_j, \mathbb{w}_j) \in \mathcal{R}_{\mathsf{pp}} & \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda) \\ & \mathsf{ai} \leftarrow \mathcal{D}(\mathsf{pp}) \\ & (\vec{\mathbb{i}}, \vec{\mathbb{x}}, \vec{\mathbb{w}}, \mathsf{ao}) \leftarrow \mathcal{E}_{\tilde{\mathcal{P}}}(\mathsf{pp}, \mathsf{ai}) \end{array}\right]$$

$$\geq \Pr\left[\begin{array}{c|r} (\mathsf{pp}, \mathsf{ai}, \vec{\mathbb{i}}, \vec{\mathbb{x}}, \mathsf{ao}) \in Z & \rho \leftarrow \mathcal{U}(\lambda) \\ \wedge \forall\, j \in [\ell]\,, \mathcal{V}^\rho(\mathrm{ivk}_j, \mathbb{x}_j, \pi_j) = 1 & \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda) \\ & \mathsf{ai} \leftarrow \mathcal{D}(\mathsf{pp}) \\ & (\vec{\mathbb{i}}, \vec{\mathbb{x}}, \vec{\pi}, \mathsf{ao}) \leftarrow \tilde{\mathcal{P}}^\rho(\mathsf{pp}, \mathsf{ai}) \\ & \forall\, j \in [\ell]\,,\ (\mathrm{ipk}_j, \mathrm{ivk}_j) \leftarrow \mathcal{I}^\rho(\mathsf{pp}, \mathbb{i}_j) \end{array}\right] - \mathrm{negl}(\lambda)\ .$$

Remark 3.3.1. The definition of knowledge soundness that we use is stronger than usual, to prove post-quantum security in Theorem 3.5.3. This stronger definition is similar to *witness-extended emulation* [Lin03].

Zero knowledge. $\mathsf{ARG} = (\mathcal{G}, \mathcal{I}, \mathcal{P}, \mathcal{V})$ has (statistical) zero knowledge if there exists a probabilistic polynomial-time simulator $\mathcal{S}$ such that for every honest adversary $\mathcal{A}$ (on input pp it only outputs triples in the indexed relation $\mathcal{R}_{\mathsf{pp}}$) the distributions below are statistically close:

$$\left\{(\rho, \mathsf{pp}, \mathbb{i}, \mathbb{x}, \pi) \;\middle|\; \begin{array}{r} \rho \leftarrow \mathcal{U}(\lambda) \\ \mathsf{pp} \leftarrow \mathcal{G}^\rho(1^\lambda) \\ (\mathbb{i}, \mathbb{x}, \mathbb{w}) \leftarrow \mathcal{A}^\rho(\mathsf{pp}) \\ (\mathrm{ipk}, \mathrm{ivk}) \leftarrow \mathcal{I}^\rho(\mathsf{pp}, \mathbb{i}) \\ \pi \leftarrow \mathcal{P}^\rho(\mathrm{ipk}, \mathbb{x}, \mathbb{w}) \end{array}\right\} \quad \text{and} \quad \left\{(\rho[\mu], \mathsf{pp}, \mathbb{i}, \mathbb{x}, \pi) \;\middle|\; \begin{array}{r} \rho \leftarrow \mathcal{U}(\lambda) \\ (\mathsf{pp}, \tau) \leftarrow \mathcal{S}^\rho(1^\lambda) \\ (\mathbb{i}, \mathbb{x}, \mathbb{w}) \leftarrow \mathcal{A}^\rho(\mathsf{pp}) \\ (\pi, \mu) \leftarrow \mathcal{S}^\rho(\tau, \mathbb{i}, \mathbb{x}) \end{array}\right\}\ .$$

Above, $\rho[\mu]$ is the function that, on input x, equals $\mu(x)$ if μ is defined on x, or $\rho(x)$ otherwise. This definition uses explicitly-programmable random oracles [BR93]. (Non-interactive zero knowledge with non-programmable random oracles is impossible for non-trivial languages [Pas03; BCS16].)

3.3.2 Proof-carrying data

A triple of algorithms $\mathsf{PCD} = (\mathbb{G}, \mathbb{I}, \mathbb{P}, \mathbb{V})$ is a (preprocessing) *proof-carrying data scheme* (PCD scheme) for a class of compliance predicates F if the properties below hold.

Definition 3.3.2. *A* **transcript** T *is a directed acyclic graph where each vertex* $u \in V(\mathsf{T})$ *is labeled by local data* $z^{(u)}_{\mathsf{loc}}$ *and each edge* $e \in E(\mathsf{T})$ *is labeled by a message* $z^{(e)} \neq \bot$*. The* **output** *of a transcript* T*, denoted* $\mathsf{o}(\mathsf{T})$*, is* $z^{(e)}$ *where* $e = (u, v)$ *is the lexicographically-first edge such that* v *is a sink.*

Definition 3.3.3. *A vertex* $u \in V(\mathsf{T})$ *is* φ**-compliant** *for* $\varphi \in \mathsf{F}$ *if for all outgoing edges* $e = (u, v) \in E(\mathsf{T})$*:*

- *(base case) if* u *has no incoming edges,* $\varphi(z^{(e)}, z^{(u)}_{\mathsf{loc}}, \bot, \ldots, \bot)$ *accepts;*
- *(recursive case) if* u *has incoming edges* $e_1, \ldots, e_m$*,* $\varphi(z^{(e)}, z^{(u)}_{\mathsf{loc}}, z^{(e_1)}, \ldots, z^{(e_m)})$ *accepts.*

We say that T *is* φ**-compliant** *if all of its vertices are* φ*-compliant.*

Completeness. PCD has perfect completeness if for every adversary $\mathcal{A}$ the following holds:

$$\Pr\left[\begin{array}{c} \left(\begin{array}{c} \varphi \in \mathsf{F} \\ \wedge\ \varphi(z, z_{\mathsf{loc}}, z_1, \ldots, z_m) = 1 \\ \wedge\ \big(\forall\, i, z_i = \bot \vee \forall\, i,\ \mathbb{V}(\mathsf{ivk}, z_i, \pi_i) = 1\big) \end{array}\right) \\ \Downarrow \\ \mathbb{V}(\mathsf{ivk}, z, \pi) = 1 \end{array} \;\middle|\; \begin{array}{r} \mathbb{pp} \leftarrow \mathbb{G}(1^\lambda) \\ (\varphi, z, z_{\mathsf{loc}}, [z_i, \pi_i]_{i=1}^m) \leftarrow \mathcal{A}(\mathbb{pp}) \\ (\mathsf{ipk}, \mathsf{ivk}) \leftarrow \mathbb{I}(\mathbb{pp}, \varphi) \\ \pi \leftarrow \mathbb{P}(\mathsf{ipk}, z, z_{\mathsf{loc}}, [z_i, \pi_i]_{i=1}^m) \end{array}\right] = 1 \ .$$

Knowledge soundness. PCD has knowledge soundness (with respect to auxiliary input distribution $\mathcal{D}$) if for every expected polynomial-time adversary $\tilde{\mathbb{P}}$ there exists an expected polynomial-time extractor $\mathbb{E}_{\tilde{\mathbb{P}}}$ such that for every set Z,

$$\begin{aligned} &\Pr\left[\begin{array}{c} \varphi \in \mathsf{F} \\ \wedge\ (\mathbb{pp}, \mathsf{ai}, \varphi, \mathsf{o}(\mathsf{T}), \mathsf{ao}) \in Z \\ \wedge\ \mathsf{T} \text{ is } \varphi\text{-compliant} \end{array} \;\middle|\; \begin{array}{r} \mathbb{pp} \leftarrow \mathbb{G}(1^\lambda) \\ \mathsf{ai} \leftarrow \mathcal{D}(\mathbb{pp}) \\ (\varphi, \mathsf{T}, \mathsf{ao}) \leftarrow \mathbb{E}_{\tilde{\mathbb{P}}}(\mathbb{pp}, \mathsf{ai}) \end{array}\right] \\ &\geq \Pr\left[\begin{array}{c} \varphi \in \mathsf{F} \\ \wedge\ (\mathbb{pp}, \mathsf{ai}, \varphi, \mathsf{o}, \mathsf{ao}) \in Z \\ \wedge\ \mathcal{V}(\mathsf{ivk}, \mathsf{o}, \pi) = 1 \end{array} \;\middle|\; \begin{array}{r} \mathbb{pp} \leftarrow \mathbb{G}(1^\lambda) \\ \mathsf{ai} \leftarrow \mathcal{D}(\mathbb{pp}) \\ (\varphi, \mathsf{o}, \pi, \mathsf{ao}) \leftarrow \tilde{\mathbb{P}}(\mathbb{pp}, \mathsf{ai}) \\ (\mathsf{ipk}, \mathsf{ivk}) \leftarrow \mathbb{I}(\mathbb{pp}, \varphi) \end{array}\right] - \mathrm{negl}(\lambda) \ . \end{aligned}$$

Zero knowledge. PCD has (statistical) zero knowledge if there exists a probabilistic polynomial-time simulator $\mathbb{S}$ such that for every honest adversary $\mathcal{A}$ the distributions below are statistically close:

$$\left\{ (\mathbb{pp}, \varphi, z, \pi) \;\middle|\; \begin{array}{r} \mathbb{pp} \leftarrow \mathbb{G}(1^\lambda) \\ (\varphi, z, z_{\mathsf{loc}}, [z_i, \pi_i]_{i=1}^m) \leftarrow \mathcal{A}(\mathbb{pp}) \\ (\mathsf{ipk}, \mathsf{ivk}) \leftarrow \mathbb{I}(\mathbb{pp}, \varphi) \\ \pi \leftarrow \mathbb{P}(\mathsf{ipk}, z, z_{\mathsf{loc}}, [z_i, \pi_i]_{i=1}^m) \end{array} \right\} \text{ and } \left\{ (\mathbb{pp}, \varphi, z, \pi) \;\middle|\; \begin{array}{r} (\mathbb{pp}, \tau) \leftarrow \mathbb{S}(1^\lambda) \\ (\varphi, z, z_{\mathsf{loc}}, [z_i, \pi_i]_{i=1}^m) \leftarrow \mathcal{A}(\mathbb{pp}) \\ \pi \leftarrow \mathbb{S}(\tau, \varphi, z) \end{array} \right\} \ .$$

An adversary is honest if its output satisfies the implicant of the completeness condition with probability 1, namely: $\varphi \in \mathsf{F}$, $\varphi(z, z_{\mathsf{loc}}, z_1, \ldots, z_m) = 1$, and either $\forall\, i,\ z_i = \bot$ or $\forall\, i,\ \mathbb{V}(\mathsf{ivk}, z_i, \pi_i) = 1$.

Efficiency. The generator $\mathbb{G}$, prover $\mathbb{P}$, indexer $\mathbb{I}$, and verifier $\mathbb{V}$ run in polynomial time. A proof π has size $\mathrm{poly}(\lambda, |\varphi|)$; in particular, it is not permitted to grow with each application of $\mathbb{P}$.

3.3.3 Instantiating the random oracle

Almost all results in this work are proved in the *random oracle model*, and so we give definitions which include random oracles. The single exception is our construction of proof-carrying data, in Section 3.5.1. We do not know how to build PCD schemes which are secure in the random oracle model from any standard assumption. Instead, we show that assuming the existence of a non-interactive argument with security in the standard (CRS) model, we obtain a PCD scheme that is also secure in the standard (CRS) model.

For this reason, the definition of PCD above is stated in the standard model (without oracles). We do not explicitly define non-interactive arguments in the standard model; the definition is easily obtained by removing the random oracle from the definitions in Section 3.3.1.

3.3.4 Post-quantum security

The definitions of both non-interactive arguments (in the standard model) and proof-carrying data can be strengthened, in a straightforward way, to express post-quantum security. In particular, we replace "polynomial-size circuit" and "polynomial-time algorithm" with their quantum analogues. Since we do not prove post-quantum security of any construction in the random oracle model, we do not discuss the quantum random oracle model.

3.3.5 Commitment schemes

We define commitment schemes and specify the Pedersen commitment scheme (used throughout this work).

Definition 3.3.4. *A* **commitment scheme** *is a tuple* $\mathsf{CM} = (\mathsf{Setup}, \mathsf{Trim}, \mathsf{Commit})$ *with the following syntax.*

- $\mathsf{CM.Setup}$*, on input a* **message format** L*, outputs public parameters* pp*, which in particular specify a message universe* $\mathcal{M}_{\mathsf{pp}}$ *and a commitment universe* $\mathcal{C}_{\mathsf{pp}}$*.*
- $\mathsf{CM.Trim}$*, on input public parameters* pp *and a* **trim specification** ℓ*, outputs a commitment key* ck *containing a description of a message space* $\mathcal{M}_{\mathsf{ck}} \subseteq \mathcal{M}_{\mathsf{pp}}$ *corresponding to* ℓ*.*
- $\mathsf{CM.Commit}$*, on input a commitment key* ck*, a message* $m \in \mathcal{M}_{\mathsf{ck}}$*, and randomness* ω*, outputs a commitment* $C \in \mathcal{C}_{\mathsf{pp}}$*.*

The commitment scheme CM *is* **binding** *if, for every message format* L *with* $|L| = \mathrm{poly}(\lambda)$ *and every expected polynomial-time adversary* $\mathcal{A}$*, the following holds:*

$$\Pr\left[\begin{array}{c} m_1 \in \mathcal{M}_{\mathsf{ck}_1}\,,\, m_2 \in \mathcal{M}_{\mathsf{ck}_2} \\ \wedge\, m_1 \neq m_2 \\ \wedge\, \mathsf{CM.Commit}(\mathsf{ck}_1, m_1; \omega_1) = \mathsf{CM.Commit}(\mathsf{ck}_2, m_2; \omega_2) \end{array} \middle| \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{CM.Setup}^{\rho}(1^{\lambda}, L) \\ \begin{pmatrix} \ell_1, m_1, \omega_1 \\ \ell_2, m_2, \omega_2 \end{pmatrix} \leftarrow \mathcal{A}^{\rho}(\mathsf{pp}) \\ \mathsf{ck}_1 \leftarrow \mathsf{CM.Trim}^{\rho}(\mathsf{pp}, \ell_1) \\ \mathsf{ck}_2 \leftarrow \mathsf{CM.Trim}^{\rho}(\mathsf{pp}, \ell_2) \end{array} \right] = \mathrm{negl}(\lambda)\ .$$

Note that $m_1 \neq m_2$ *is well-defined since* $\mathcal{M}_{\mathsf{ck}_1}, \mathcal{M}_{\mathsf{ck}_2} \subseteq \mathcal{M}_{\mathsf{pp}}$*.*

Remark 3.3.5. The binding property is stated for expected polynomial time adversaries, since this is how it will be used in this work. This is equivalent to the standard definition of binding (i.e., for polynomial size adversaries) via a non-uniform reduction.

The **Pedersen commitment scheme** $\mathsf{CM} = (\mathsf{Setup}, \mathsf{Trim}, \mathsf{Commit})$ operates as follows, for some algorithm $\mathsf{SampleGrp}$ that outputs $(\mathbb{G}, q, G)$ where $\mathbb{G}$ is a group of prime order q generated by G.

- The message format L and trim specification ℓ are nonnegative integers with $\ell \leq L$.
- $\mathsf{CM.Setup}(1^\lambda, L)$ runs $(\mathbb{G}, q, G) \leftarrow \mathsf{SampleGrp}(1^\lambda)$, samples $\vec{G} = (G_1, \ldots, G_L, H) \in \mathbb{G}^{L+1}$ uniformly at random, and outputs $\mathsf{pp} := ((\mathbb{G}, q, G), \vec{G})$; $\mathcal{M}_{\mathsf{pp}} := \mathbb{F}^L$ where $\mathbb{F}$ is the prime field of size q, and $\mathcal{C}_{\mathsf{pp}} := \mathbb{G}$.
- $\mathsf{CM.Trim}(\mathsf{pp}, \ell)$ outputs $\mathsf{ck} = ((\mathbb{G}, q, G), (G_1, \ldots, G_\ell, H))$; this key determines $\mathcal{M}_{\mathsf{ck}} := \mathbb{F}^\ell$.
- $\mathsf{CM.Commit}(\mathsf{ck}, m; \omega)$ outputs $\sum_{i=1}^{\ell} m_i \cdot G_i + \omega \cdot H$, where $\omega \in \mathbb{F}$.

CM is binding when the discrete logarithm problem is hard in $\mathbb{G}$ as sampled by $\mathsf{SampleGrp}$. CM is perfectly hiding: for any message m, $\mathsf{CM.Commit}(\mathsf{ck}, m; \omega)$ is uniformly random in $\mathbb{G}$ when ω is uniformly random in $\mathbb{F}$. CM satisfies the following homomorphic property: for all keys ck, $\alpha, \beta \in \mathbb{F}$, $m_1, m_2 \in \mathbb{F}^\ell$, $\omega_1, \omega_2 \in \mathbb{F}$,

$$\alpha \cdot \mathsf{CM.Commit}(\mathsf{ck}, m_1; \omega_1) + \beta \cdot \mathsf{CM.Commit}(\mathsf{ck}, m_2; \omega_2) = \mathsf{CM.Commit}(\mathsf{ck}, \alpha m_1 + \beta m_2; \alpha\omega_1 + \beta\omega_2)$$

where "$\cdot$" above represents scalar multiplication in $\mathbb{G}$ (the natural action of $\mathbb{F}$ on $\mathbb{G}$).

3.4 Split accumulation schemes for relations

Let $\Phi\colon \{0,1\}^* \to \{0,1\}$ be a (relation) predicate and $\mathcal{H}$ a randomized oracle algorithm that outputs predicate parameters pp_Φ (see below). A **split accumulation scheme for** $(\Phi, \mathcal{H})$ is a tuple of algorithms $\mathsf{AS} = (\mathrm{G}, \mathrm{I}, \mathrm{P}, \mathrm{V}, \mathrm{D})$ of which P, V have access to the same random oracle ρ. The algorithms have the following syntax and properties.

Syntax. The algorithms comprising AS have the following syntax:

- *Generator:* On input a security parameter λ (in unary), G samples and outputs public parameters pp.
- *Indexer:* On input public parameters pp, predicate parameters pp_Φ (generated by $\mathcal{H}$), and a predicate index i_Φ, I deterministically computes and outputs a triple $(\mathsf{apk}, \mathsf{avk}, \mathsf{dk})$ consisting of an accumulator proving key apk, an accumulator verification key avk, and a decision key dk.[8]
- *Accumulation prover:* On input the accumulator proving key apk, predicate inputs $[(\mathsf{qx}_i, \mathsf{qw}_i)]_{i=1}^n$, and old accumulators $[\mathsf{acc}_j]_{j=1}^m = [(\mathsf{acc}_j.\mathbb{x}, \mathsf{acc}_j.\mathbb{w})]_{j=1}^m$, P outputs a new accumulator $\mathsf{acc} = (\mathsf{acc}.\mathbb{x}, \mathsf{acc}.\mathbb{w})$ and a proof pf for the accumulation verifier.
- *Accumulation verifier:* On input the accumulator verification key avk, predicate input instances $[\mathsf{qx}_i]_{i=1}^n$, accumulator instances $[\mathsf{acc}_j.\mathbb{x}]_{j=1}^m$, a new accumulator instance $\mathsf{acc}.\mathbb{x}$, and a proof pf, V outputs a bit indicating whether $\mathsf{acc}.\mathbb{x}$ correctly accumulates $[(\mathsf{qx}_i, \mathsf{qw}_i)]_{i=1}^n$ and $[\mathsf{acc}_j.\mathbb{x}]_{j=1}^m$.
- *Decider:* On input the decision key dk, and an accumulator $\mathsf{acc} = (\mathsf{acc}.\mathbb{x}, \mathsf{acc}.\mathbb{w})$, D outputs a bit indicating whether acc is a valid accumulator.

These algorithms must satisfy two properties, *completeness* and *knowledge soundness*, defined below. We additionally define a notion of zero knowledge that we use to achieve zero knowledge PCD (see Section 3.5).

Completeness. For every (unbounded) adversary $\mathcal{A}$,

$$\Pr\left[\begin{array}{c} \forall\, j \in [m],\ \mathrm{D}(\mathsf{dk}, \mathsf{acc}_j) = 1 \\ \forall\, i \in [n],\ \Phi(\mathsf{pp}_\Phi, \mathsf{i}_\Phi, \mathsf{qx}_i, \mathsf{qw}_i) = 1 \\ \Downarrow \\ \mathrm{V}^\rho(\mathsf{avk}, [\mathsf{qx}_i]_{i=1}^n, [\mathsf{acc}_j.\mathbb{x}]_{j=1}^m, \mathsf{acc}.\mathbb{x}, \mathsf{pf}) = 1 \\ \mathrm{D}(\mathsf{dk}, \mathsf{acc}) = 1 \end{array} \middle| \begin{array}{r} \rho \leftarrow \mathcal{U}(\lambda) \\ \mathsf{pp} \leftarrow \mathrm{G}(1^\lambda) \\ \mathsf{pp}_\Phi \leftarrow \mathcal{H}(1^\lambda) \\ (\mathsf{i}_\Phi, [(\mathsf{qx}_i, \mathsf{qw}_i)]_{i=1}^n, [\mathsf{acc}_j]_{j=1}^m) \leftarrow \mathcal{A}^\rho(\mathsf{pp}, \mathsf{pp}_\Phi) \\ (\mathsf{apk}, \mathsf{avk}, \mathsf{dk}) \leftarrow \mathrm{I}(\mathsf{pp}, \mathsf{pp}_\Phi, \mathsf{i}_\Phi) \\ (\mathsf{acc}, \mathsf{pf}) \leftarrow \mathrm{P}^\rho(\mathsf{apk}, [(\mathsf{qx}_i, \mathsf{qw}_i)]_{i=1}^n, [\mathsf{acc}_j]_{j=1}^m) \end{array}\right] = 1 \ .$$

Note that for $m = n = 0$ the precondition on the left-hand side holds vacuously and this is required for the completeness condition to be non-trivial.

[8] In some schemes, for efficiency, the indexer I should have oracle access to the predicate parameters pp_Φ and predicate index i_Φ, rather than reading them in full. All of our constructions and statements extend, in a straightforward way, to this case.

Knowledge soundness. There exists an extractor E running in expected polynomial time such that for every adversary $\tilde{\mathrm{P}}$ running in expected (non-uniform) polynomial time and auxiliary input distribution $\mathcal{D}$, the following probability is negligibly close to 1:

$$\Pr\left[\begin{array}{c|r} \begin{array}{c} \mathrm{V}^{\rho}(\mathsf{avk}, [\mathsf{qx}_i]_{i=1}^n, [\mathsf{acc}_j.\mathbb{x}]_{j=1}^m, \mathsf{acc}.\mathbb{x}, \mathsf{pf}) = 1 \\ \mathrm{D}(\mathsf{dk}, \mathsf{acc}) = 1 \\ \Downarrow \\ \forall\, i \in [n],\ \Phi(\mathsf{pp}_\Phi, \mathsf{i}_\Phi, \mathsf{qx}_i, \mathsf{qw}_i) = 1 \\ \forall\, j \in [m],\ \mathrm{D}\big(\mathsf{dk}, (\mathsf{acc}_j.\mathbb{x}, \mathsf{acc}_j.\mathbb{w})\big) = 1 \end{array} & \begin{array}{r} \rho \leftarrow \mathcal{U}(\lambda) \\ \mathsf{pp} \leftarrow \mathrm{G}(1^\lambda) \\ \mathsf{pp}_\Phi \leftarrow \mathcal{H}(1^\lambda) \\ \mathsf{ai} \leftarrow \mathcal{D}(1^\lambda) \\ (\mathsf{i}_\Phi, [\mathsf{qx}_i]_{i=1}^n, [\mathsf{acc}_j.\mathbb{x}]_{j=1}^m, \mathsf{acc}, \mathsf{pf}; r) \leftarrow \tilde{\mathrm{P}}^{\rho}(\mathsf{pp}, \mathsf{pp}_\Phi, \mathsf{ai}) \\ ([\mathsf{qw}_i]_{i=1}^n, [\mathsf{acc}_j.\mathbb{w}]_{j=1}^m) \leftarrow \mathrm{E}^{\tilde{\mathrm{P}}, \rho}(\mathsf{pp}, \mathsf{pp}_\Phi, \mathsf{ai}, r) \\ (\mathsf{apk}, \mathsf{avk}, \mathsf{dk}) \leftarrow \mathrm{I}(\mathsf{pp}, \mathsf{pp}_\Phi, \mathsf{i}_\Phi) \end{array} \end{array}\right].$$

Zero knowledge. There exists a polynomial-time simulator S such that for every polynomial-size "honest" adversary $\mathcal{A}$ (see below) the following distributions are (statistically/computationally) indistinguishable:

$$\left\{(\rho, \mathsf{pp}, \mathsf{pp}_\Phi, \mathsf{i}_\Phi, \mathsf{acc}) \;\middle|\; \begin{array}{r} \rho \leftarrow \mathcal{U}(\lambda) \\ \mathsf{pp} \leftarrow \mathrm{G}(1^\lambda) \\ \mathsf{pp}_\Phi \leftarrow \mathcal{H}(1^\lambda) \\ (\mathsf{i}_\Phi, [(\mathsf{qx}_i, \mathsf{qw}_i)]_{i=1}^n, [\mathsf{acc}_j]_{j=1}^m) \leftarrow \mathcal{A}^{\rho}(\mathsf{pp}, \mathsf{pp}_\Phi) \\ (\mathsf{apk}, \mathsf{avk}, \mathsf{dk}) \leftarrow \mathrm{I}(\mathsf{pp}, \mathsf{pp}_\Phi, \mathsf{i}_\Phi) \\ (\mathsf{acc}, \mathsf{pf}) \leftarrow \mathrm{P}^{\rho}(\mathsf{apk}, [(\mathsf{qx}_i, \mathsf{qw}_i)]_{i=1}^n, [\mathsf{acc}_j]_{j=1}^m) \end{array}\right\}$$

and

$$\left\{(\rho[\mu], \mathsf{pp}, \mathsf{pp}_\Phi, \mathsf{i}_\Phi, \mathsf{acc}) \;\middle|\; \begin{array}{r} \rho \leftarrow \mathcal{U}(\lambda) \\ (\mathsf{pp}, \tau) \leftarrow \mathrm{S}^{\rho}(1^\lambda) \\ \mathsf{pp}_\Phi \leftarrow \mathcal{H}(1^\lambda) \\ (\mathsf{i}_\Phi, [(\mathsf{qx}_i, \mathsf{qw}_i)]_{i=1}^n, [\mathsf{acc}_j]_{j=1}^m) \leftarrow \mathcal{A}^{\rho}(\mathsf{pp}, \mathsf{pp}_\Phi) \\ (\mathsf{acc}, \mu) \leftarrow \mathrm{S}^{\rho}(\tau, \mathsf{pp}_\Phi, \mathsf{i}_\Phi) \end{array}\right\}.$$

Here $\mathcal{A}$ is *honest* if it outputs, with probability 1, a tuple $(\mathsf{i}_\Phi, [(\mathsf{qx}_i, \mathsf{qw}_i)]_{i=1}^n, [\mathsf{acc}_j]_{j=1}^m)$ such that $\Phi(\mathsf{pp}_\Phi, \mathsf{i}_\Phi, \mathsf{qx}_i, \mathsf{qw}_i) = 1$ and $\mathrm{D}(\mathsf{dk}, \mathsf{acc}_j) = 1$ for all $i \in [n]$ and $j \in [m]$. Note that the simulator S is *not* required to simulate the accumulation verifier proof pf.

Remark 3.4.1 (predicates with oracles)**.** In Section 3.8 we accumulate predicates Φ that themselves have access to oracles, as do their associated parameter generation algorithms $\mathcal{H}$. These oracles are *disjoint* from the random oracle ρ used by the accumulation scheme. The definitions above can be adapted to this setting by providing all algorithms $(\mathrm{G}, \mathrm{I}, \mathrm{P}, \mathrm{V}, \mathrm{D})$ of the accumulation scheme, adversaries $\mathcal{A}$ and $\tilde{\mathrm{P}}$, the extractor E, and the simulator S) with access to these oracles.

3.4.1 Special case: accumulators and predicate inputs are identical

Some accumulation schemes have the property that the decider is equal to the predicate itself: $\mathrm{D}(\mathsf{dk}, \mathsf{acc}) \equiv \Phi(\mathsf{pp}_\Phi, \mathsf{i}_\Phi, \mathsf{acc}.\mathbb{x}, \mathsf{acc}.\mathbb{w})$. This implies that predicate inputs and accumulators have

the same form, and are split in the same way. In this case, the definitions can be simplified. Below we state these simplified definitions because we use them in Sections 3.7 and 3.11.

Completeness. For every (unbounded) adversary $\mathcal{A}$:

$$\Pr\left[\begin{array}{c|r} \forall\, i \in [n],\ \Phi(\mathsf{pp}_\Phi, \mathsf{i}_\Phi, \mathsf{qx}_i, \mathsf{qw}_i) = 1 & \rho \leftarrow \mathcal{U}(\lambda) \\ \Downarrow & \mathsf{pp} \leftarrow \mathrm{G}(1^\lambda) \\ \mathrm{V}^\rho(\mathsf{avk}, [\mathsf{qx}_i]_{i=1}^n, \mathsf{acc}.\mathbb{x}, \mathsf{pf}) = 1 & \mathsf{pp}_\Phi \leftarrow \mathcal{H}(1^\lambda) \\ \mathrm{D}(\mathsf{dk}, \mathsf{acc}) = 1 & (\mathsf{i}_\Phi, [(\mathsf{qx}_i, \mathsf{qw}_i)]_{i=1}^n) \leftarrow \mathcal{A}(\mathsf{pp}, \mathsf{pp}_\Phi) \\ & (\mathsf{apk}, \mathsf{avk}, \mathsf{dk}) \leftarrow \mathrm{I}(\mathsf{pp}, \mathsf{pp}_\Phi, \mathsf{i}_\Phi) \\ & (\mathsf{acc}, \mathsf{pf}) \leftarrow \mathrm{P}^\rho(\mathsf{apk}, [(\mathsf{qx}_i, \mathsf{qw}_i)]_{i=1}^n) \end{array}\right] = 1 \ .$$

Knowledge soundness. There exists an extractor E running in expected polynomial time such that for every adversary $\tilde{\mathrm{P}}$ running in expected (non-uniform) polynomial time and auxiliary input distribution $\mathcal{D}$,

$$\Pr\left[\begin{array}{c|r} & \rho \leftarrow \mathcal{U}(\lambda) \\ \mathrm{V}^\rho(\mathsf{avk}, [\mathsf{qx}_i]_{i=1}^n, \mathsf{acc}.\mathbb{x}, \mathsf{pf}) = 1 & \mathsf{pp} \leftarrow \mathrm{G}(1^\lambda) \\ \mathrm{D}(\mathsf{dk}, \mathsf{acc}) = 1 & \mathsf{pp}_\Phi \leftarrow \mathcal{H}(1^\lambda) \\ \Downarrow & \mathsf{ai} \leftarrow \mathcal{D}(1^\lambda) \\ \forall\, i \in [n],\ \Phi(\mathsf{pp}_\Phi, \mathsf{i}_\Phi, \mathsf{qx}_i, \mathsf{qw}_i) = 1 & (\mathsf{i}_\Phi, [\mathsf{qx}_i]_{i=1}^n, \mathsf{acc}, \mathsf{pf}; r) \leftarrow \tilde{\mathrm{P}}^\rho(\mathsf{pp}, \mathsf{pp}_\Phi, \mathsf{ai}) \\ & [\mathsf{qw}_i]_{i=1}^n \leftarrow \mathrm{E}^{\tilde{\mathrm{P}}, \rho}(\mathsf{pp}, \mathsf{pp}_\Phi, \mathsf{ai}, r) \\ & (\mathsf{apk}, \mathsf{avk}, \mathsf{dk}) \leftarrow \mathrm{I}(\mathsf{pp}, \mathsf{pp}_\Phi, \mathsf{i}_\Phi) \end{array}\right] \geq 1 - \mathrm{negl}(\lambda) \ .$$

3.4.2 A relaxation of knowledge soundness

The definitions of knowledge soundness that we presented so far are convenient for proving schemes secure in the random oracle model, but are stronger than what we need. To prove security for PCD in Section 3.5 a weaker notion of "multi-instance" extraction will suffice. This is motivated by analyses in the quantum random oracle model, where the no-cloning principle necessitates that the extractor simulate the oracle itself in order to extract. In contrast, in the classical setting the extractor may simply "observe" the adversary's queries to the real oracle, which justifies the prior definition. Below we state the property we use, and then explain how it is implied (in the classical setting) by the prior definitions of knowledge soundness.

Knowledge soundness (with respect to auxiliary input distribution $\mathcal{D}$). For every (non-uniform) adversary $\tilde{\mathrm{P}}$ running in expected polynomial time there exists an extractor E running in expected polynomial time such that for every set Z the following probabilities are within $\mathrm{negl}(\lambda)$ of each other:

$$\Pr\left[\begin{array}{c} \left(\mathsf{pp}, \mathsf{pp}_\Phi, \mathsf{ai}, \begin{bmatrix} \mathsf{i}_\Phi^{(k)} \\ \mathsf{acc}^{(k)} \\ [\mathsf{qx}_i^{(k)}]_{i=1}^n \\ [\mathsf{acc}_j.\mathbb{x}^{(k)}]_{j=1}^m \end{bmatrix}_{k=1}^{\ell}, \mathsf{ao}\right) \in Z \\ \wedge \\ \left\{\begin{array}{c} \forall\, j \in [m],\ \mathrm{D}(\mathsf{dk}^{(k)}, \mathsf{acc}_j^{(k)}) = 1 \\ \forall\, i \in [n],\ \Phi(\mathsf{pp}_\Phi, \mathsf{i}_\Phi^{(k)}, \mathsf{qx}_i^{(k)}, \mathsf{qw}_i^{(k)}) = 1 \end{array}\right\}_{k=1}^{\ell} \end{array} \middle| \begin{array}{r} \rho \leftarrow \mathcal{U}(\lambda) \\ \mathsf{pp} \leftarrow \mathrm{G}(1^\lambda) \\ \mathsf{pp}_\Phi \leftarrow \mathcal{H}(1^\lambda) \\ \mathsf{ai} \leftarrow \mathcal{D}(1^\lambda) \\ \left(\begin{bmatrix} \mathsf{i}_\Phi^{(k)} \\ \mathsf{acc}^{(k)} \\ [(\mathsf{qx}_i^{(k)}, \mathsf{qw}_i^{(k)})]_{i=1}^n \\ [\mathsf{acc}_j^{(k)}]_{j=1}^m \end{bmatrix}_{k=1}^{\ell}, \mathsf{ao}\right) \leftarrow \mathrm{E}_{\tilde{\mathrm{P}}}(\mathsf{pp}, \mathsf{pp}_\Phi, \mathsf{ai}) \\ \forall\, k,\ (\mathsf{apk}^{(k)}, \mathsf{avk}^{(k)}, \mathsf{dk}^{(k)}) \leftarrow \mathrm{I}(\mathsf{pp}, \mathsf{pp}_\Phi, \mathsf{i}_\Phi^{(k)}) \end{array}\right]$$

and

$$\Pr\left[\begin{array}{c} \left(\mathsf{pp}, \mathsf{pp}_\Phi, \mathsf{ai}, \begin{bmatrix} \mathsf{i}_\Phi^{(k)} \\ \mathsf{acc}^{(k)} \\ [\mathsf{qx}_i^{(k)}]_{i=1}^n \\ [\mathsf{acc}_j.\mathbb{x}^{(k)}]_{j=1}^m \end{bmatrix}_{k=1}^{\ell}, \mathsf{ao}\right) \in Z \\ \wedge \\ \left\{\begin{array}{c} \mathrm{V}^\rho(\mathsf{avk}^{(k)}, [\mathsf{qx}_i^{(k)}]_{i=1}^n, [\mathsf{acc}_j.\mathbb{x}^{(k)}]_{j=1}^m, \mathsf{acc}.\mathbb{x}^{(k)}, \mathsf{pf}^{(k)}) = 1 \\ \mathrm{D}(\mathsf{dk}^{(k)}, \mathsf{acc}^{(k)}) = 1 \end{array}\right\}_{k=1}^{\ell} \end{array} \middle| \begin{array}{r} \rho \leftarrow \mathcal{U}(\lambda) \\ \mathsf{pp} \leftarrow \mathrm{G}(1^\lambda) \\ \mathsf{pp}_\Phi \leftarrow \mathcal{H}(1^\lambda) \\ \mathsf{ai} \leftarrow \mathcal{D}(1^\lambda) \\ \left(\begin{bmatrix} \mathsf{i}_\Phi^{(k)} \\ \mathsf{acc}^{(k)} \\ [\mathsf{qx}_i^{(k)}]_{i=1}^n \\ [\mathsf{acc}_j.\mathbb{x}^{(k)}]_{j=1}^m \\ \mathsf{pf}^{(k)} \end{bmatrix}_{k=1}^{\ell}, \mathsf{ao}\right) \leftarrow \tilde{\mathrm{P}}^\rho(\mathsf{pp}, \mathsf{pp}_\Phi, \mathsf{ai}) \\ \forall\, k,\ (\mathsf{apk}^{(k)}, \mathsf{avk}^{(k)}, \mathsf{dk}^{(k)}) \leftarrow \mathrm{I}(\mathsf{pp}, \mathsf{pp}_\Phi, \mathsf{i}_\Phi^{(k)}) \end{array}\right]$$

The above definition is implied. In the classical setting, the above definition is implied by the definition of knowledge soundness given earlier in this section. The multi-instance extractor $\mathrm{E}_{\tilde{\mathrm{P}}}$ as follows:

$\mathrm{E}_{\tilde{\mathrm{P}}}(\mathsf{pp}_{\mathsf{AS}}, \mathsf{pp}_\Phi, \mathsf{ai})$:

1. Initialize the table $\mathsf{tr}\colon \{0,1\}^* \rightharpoonup \{0,1\}^\lambda$ to be empty.
2. Run $([\mathsf{i}_\Phi^{(k)}, \mathsf{acc}^{(k)}, [\mathsf{qx}_i^{(k)}]_{i=1}^n, [\mathsf{acc}_j.\mathbb{x}^{(k)}]_{j=1}^m, \mathsf{pf}^{(k)}]_{k=1}^\ell, \mathsf{ao}; r) \leftarrow \tilde{\mathrm{P}}^{(\cdot)}(\mathsf{pp}, \mathsf{pp}_\Phi, \mathsf{ai})$, simulating its access to the random oracle using tr.
3. For each $k \in [\ell]$, let $\tilde{\mathrm{P}}^{(k)}$ equal $\tilde{\mathrm{P}}$ with its output is restricted to the index k. Run
$$([\mathsf{qw}_i^{(k)}]_{i=1}^n, [\mathsf{acc}_j.\mathbb{w}^{(k)}]_{j=1}^m) \leftarrow \mathrm{E}^{\tilde{\mathrm{P}}^{(k)},(\cdot)}(\mathsf{pp}, \mathsf{pp}_\Phi, \mathsf{ai}, r)$$
simulating its access to the random oracle using tr.
4. Output $\left(\left[\mathsf{i}_\Phi^{(k)}, \mathsf{acc}^{(k)}, [(\mathsf{qx}_i^{(k)}, \mathsf{qw}_i^{(k)})]_{i=1}^n, [(\mathsf{acc}_j.\mathbb{x}^{(k)}, \mathsf{acc}_j.\mathbb{w}^{(k)})]_{j=1}^m\right]_{k=1}^\ell, \mathsf{ao}\right)$.

3.5 PCD from arguments of knowledge with split accumulation

We formally restate and then prove Theorem 3, which provides a construction of proof-carrying data (PCD) from any NARK that has a split accumulation scheme with certain efficiency properties.

First, we provide definitions and notation for these properties.

Definition 3.5.1 (accumulation for ARG)**.** *We say that* $\mathsf{AS} = (\mathrm{G}, \mathrm{I}, \mathrm{P}, \mathrm{V}, \mathrm{D})$ *is a split accumulation scheme for the non-interactive argument system* $\mathsf{ARG} = (\mathcal{G}, \mathcal{I}, \mathcal{P}, \mathcal{V})$ *if* AS *is a split accumulation scheme for the pair* $(\Phi_{\mathcal{V}}, \mathcal{H}_{\mathsf{ARG}} := \mathcal{G})$ *where* $\Phi_{\mathcal{V}}$ *is defined below:*

$\Phi_{\mathcal{V}}(\mathsf{pp}_\Phi = \mathsf{pp}, \mathsf{i}_\Phi = \mathbb{i}, \mathsf{qx} = (\mathbb{x}, \pi.\mathbb{x}), \mathsf{qw} = \pi.\mathbb{w})$*:*

1. $(\mathrm{ipk}, \mathrm{ivk}) \leftarrow \mathcal{I}(\mathsf{pp}, \mathbb{i})$.
2. *Output* $\mathcal{V}(\mathrm{ivk}, \mathbb{x}, (\pi.\mathbb{x}, \pi.\mathbb{w}))$.

Definition 3.5.2. *Let* $\mathsf{AS} = (\mathrm{G}, \mathrm{I}, \mathrm{P}, \mathrm{V}, \mathrm{D})$ *be an accumulation scheme for a non-interactive argument (see Definition 3.5.1). We denote by* $\mathrm{V}^{(\lambda,m,N,k)}$ *the circuit corresponding to the computation of the accumulation verifier* V*, for security parameter* λ*, when checking the accumulation of* m *instance-proof pairs and accumulators, on an index of size at most* N*, where each instance is of size at most* k*.*

We denote by $\mathsf{v}(\lambda, m, N, k)$ *the size of the circuit* $\mathrm{V}^{(\lambda,m,N,k)}$*, by* $|\mathsf{avk}(\lambda, m, N)|$ *the size of the accumulator verification key* avk*, and by* $|\mathsf{acc}.\mathbb{x}(\lambda, m, N)|$ *the size of an accumulator instance.*

Note that here we have specified that the size of $\mathsf{acc}.\mathbb{x}$ is bounded by a function of λ, m, N; in particular, it *may not* depend on the number of instances accumulated, or on the input size bound k.

When we invoke the accumulation verifier in our construction of PCD, an instance will consist of an accumulator verification key, an accumulator *instance*, and some additional data of size ℓ. Thus the size of the accumulation verifier circuit used in the scheme is given by

$$\mathsf{v}^*(\lambda, m, N, \ell) := \mathsf{v}(\lambda, m, N, |\mathsf{avk}(\lambda, m, N)| + |\mathsf{acc}.\mathbb{x}(\lambda, m, N)| + \ell) \ .$$

The notion of "sublinear verification" which is important here is that v^* is sublinear in N. The following theorem shows that when this is the case, this accumulation scheme can be used to construct PCD.

Theorem 3.5.3. *There exists a polynomial-time transformation* T *such that if* $\mathsf{ARG} = (\mathcal{G}, \mathcal{I}, \mathcal{P}, \mathcal{V})$ *is a NARK for circuit satisfiability and* AS *is a split accumulation scheme for* ARG *then* $\mathsf{PCD} = (\mathbb{G}, \mathbb{I}, \mathbb{P}, \mathbb{V}) := \mathrm{T}(\mathsf{ARG}, \mathsf{AS})$ *is a PCD scheme for constant-depth compliance predicates, provided*

$$\exists\, \epsilon \in (0,1) \text{ and a polynomial } \alpha \text{ s.t. } \mathsf{v}^*(\lambda, m, N, \ell) = O(N^{1-\epsilon} \cdot \alpha(\lambda, m, \ell)) \ .$$

Moreover:

- *If* ARG *and* AS *are secure against quantum adversaries, then* PCD *is secure against quantum adversaries.*
- *If* ARG *and* AS *are (post-quantum) zero knowledge, then* PCD *is (post-quantum) zero knowledge.*

- *If the size of the predicate* $\varphi\colon \mathbb{F}^{(m+2)\ell} \to \mathbb{F}$ *is* $f = \omega(\alpha(\lambda, m, \ell)^{1/\epsilon})$ *then:*
  - *the cost of running* $\mathbb{I}$ *is equal to the cost of running both* $\mathcal{I}$ *and* I *on an index of size* $f + o(f)$*;*
  - *the cost of running* $\mathbb{P}$ *is equal to the cost of accumulating* m *instance-proof pairs using* P*, and running* $\mathcal{P}$*, on an index of size* $f + o(f)$ *and instance of size* $o(f)$*;*
  - *the cost of running* $\mathbb{V}$ *is equal to the cost of running both* $\mathcal{V}$ *and* D *on an index of size* $f + o(f)$ *and an instance of size* $o(f)$*.*

This last point gives the conditions for a *sublinear additive* recursive overhead; i.e., when the *additional* cost of proving that φ is satisfied recursively is asymptotically smaller than the cost of proving that φ is satisfied locally. Note that the smaller the compliance predicate φ, the more efficient the accumulation scheme has to be in order to achieve this.

Remark 3.5.4 (accumulator instance size)**.** Theorem 3.5.3 requires that the size of an accumulator instance $\mathsf{acc}.\mathbb{x}$ be *independent* of the instance size k. This is achieved by our split accumulation scheme in Section 3.8.2. It is also straightforward to convert any split accumulation scheme into one that satisfies this condition, using a collision resistant hash function h. Specifically, the accumulator instance of the new scheme will be $h(\mathsf{acc}.\mathbb{x})$, and $\mathsf{acc}.\mathbb{x}$ is appended to the accumulator witness and accumulation proof. The accumulation verifier and decider then simply verify the hash in addition to performing their original computation.

3.5.1 Construction

Let $\mathsf{ARG} = (\mathcal{G}, \mathcal{I}, \mathcal{P}, \mathcal{V})$ be a non-interactive argument for circuit satisfiability and $\mathsf{AS} = (\mathrm{G}, \mathrm{I}, \mathrm{P}, \mathrm{V}, \mathrm{D})$ an accumulation scheme for ARG (see Definition 3.5.1). Below we construct a PCD scheme $\mathsf{PCD} = (\mathbb{G}, \mathbb{I}, \mathbb{P}, \mathbb{V})$.

Given a compliance predicate $\varphi\colon \mathbb{F}^{(m+2)\ell} \to \mathbb{F}$, the circuit that realizes the recursion is as follows.

$R_{\mathrm{V},\varphi}^{(\lambda,N,k)}\big((\mathsf{avk}, z, \mathsf{acc}.\mathbb{x}), (z_{\mathsf{loc}}, [z_i, \pi_i.\mathbb{x}, \mathsf{acc}_i.\mathbb{x}]_{i=1}^m, \mathsf{pf})\big)$:

1. Check that the compliance predicate $\varphi(z, z_{\mathsf{loc}}, z_1, \ldots, z_m)$ accepts.
2. If there exists $i \in [m]$ such that $z_i \neq \bot$, check that the NARK accumulation verifier accepts:

$$\mathrm{V}^{(\lambda,m,N,k)}(\mathsf{avk}, [\mathsf{qx}_i]_{i=1}^m, [\mathsf{acc}_i.\mathbb{x}]_{i=1}^m, \mathsf{acc}.\mathbb{x}, \mathsf{pf}) = 1 \text{ where } \mathsf{qx}_i := \big((\mathsf{avk}, z_i, \mathsf{acc}_i.\mathbb{x}), \pi_i.\mathbb{x}\big) \ .$$

3. If the above checks hold, output 1; otherwise, output 0.

Above, $\mathrm{V}^{(\lambda,m,N,k)}$ refers to the circuit representation of V with input size appropriate for security parameter λ, number of instance-proof pairs and accumulators m, circuit size N, and circuit input size k.

Next we describe the generator $\mathbb{G}$, indexer $\mathbb{I}$, prover $\mathbb{P}$, and verifier $\mathbb{V}$ of the PCD scheme.

- $\mathbb{G}(1^\lambda)$: Sample $\mathsf{pp} \leftarrow \mathcal{G}(1^\lambda)$ and $\mathsf{pp}_{\mathsf{AS}} \leftarrow \mathrm{G}(1^\lambda)$, and output $\mathbb{pp} := (\mathsf{pp}, \mathsf{pp}_{\mathsf{AS}})$.

- $\mathbb{I}(\mathbb{pp}, \varphi)$:
  1. Compute the integer $N := N(\lambda, |\varphi|, m, \ell)$, where N is defined in Lemma 3.5.5 below.
  2. Construct the circuit $R := R_{\mathrm{V},\varphi}^{(\lambda,N,k)}$ where $k := |\mathsf{avk}(\lambda, N)| + |\mathsf{acc}.\mathbb{x}(\lambda, m, N)| + \ell$.
  3. Compute the index key pair $(\mathrm{ipk}, \mathrm{ivk}) := \mathcal{I}(\mathsf{pp}, R)$ for the circuit R for the NARK.
  4. Compute the index key triple $(\mathsf{apk}, \mathsf{dk}, \mathsf{avk}) := \mathrm{I}(\mathsf{pp}_{\mathsf{AS}}, \mathsf{pp}_{\Phi} = \mathsf{pp}, \mathsf{i}_{\Phi} = R)$ for the accumulator.
  5. Output the proving key $\mathsf{ipk} := (\mathrm{ipk}, \mathsf{apk})$ and verification key $\mathsf{ivk} := (\mathrm{ivk}, \mathsf{dk}, \mathsf{avk})$.

- $\mathbb{P}(\mathsf{ipk}, z, z_{\mathsf{loc}}, [z_i, (\pi_i, \mathsf{acc}_i)]_{i=1}^m)$:
  1. If $z_i = \bot$ for all $i \in [m]$ then sample $(\mathsf{acc}, \mathsf{pf}) \leftarrow \mathrm{P}(\mathsf{apk}, \bot)$.
  2. If $z_i \neq \bot$ for some $i \in [m]$ then:
     a) set predicate input instance $\mathsf{qx}_i := ((\mathsf{avk}, z_i, \mathsf{acc}_i.\mathbb{x}), \pi_i.\mathbb{x})$;
     b) set predicate input witness $\mathsf{qw}_i := (\mathsf{acc}_i.\mathbb{w}, \pi_i.\mathbb{w})$;
     c) sample $(\mathsf{acc}, \mathsf{pf}) \leftarrow \mathrm{P}(\mathsf{apk}, [(\mathsf{qx}_i, \mathsf{qw}_i)]_{i=1}^m, [\mathsf{acc}_i]_{i=1}^m)$.
  3. Sample $\pi \leftarrow \mathcal{P}\big(\mathrm{ipk}, (\mathsf{avk}, z, \mathsf{acc}.\mathbb{x}), (z_{\mathsf{loc}}, [z_i, \pi_i.\mathbb{x}, \mathsf{acc}_i.\mathbb{x}]_{i=1}^m, \mathsf{pf})\big)$.
  4. Output (π, acc).

- $\mathbb{V}(\mathsf{ivk}, z, (\pi, \mathsf{acc}))$: Accept if both $\mathcal{V}(\mathrm{ivk}, (\mathsf{avk}, z, \mathsf{acc}.\mathbb{x}), \pi)$ and $\mathrm{D}(\mathsf{dk}, \mathsf{acc})$ accept.

3.5.2 Completeness

Let $\mathcal{A}$ be any adversary that causes the completeness condition of PCD to be satisfied with probability p. We construct an adversary $\mathcal{B}$, as follows, that causes the completeness condition of AS to be satisfied with probability at most p.

$\mathcal{B}(\mathsf{pp}, \mathsf{pp}_{\mathsf{AS}})$:
1. Set $\mathbb{pp} := (\mathsf{pp}, \mathsf{pp}_{\mathsf{AS}})$ and compute $(\varphi, z, z_{\mathsf{loc}}, [z_i, \pi_i, \mathsf{acc}_i]_{i=1}^m) \leftarrow \mathcal{A}(\mathbb{pp})$.
2. Set $(\mathsf{apk}, \mathsf{dk}, \mathsf{avk}) := \mathrm{I}(\mathsf{pp}_{\mathsf{AS}}, \mathsf{pp}, R_{\mathrm{V},\varphi}^{(\lambda,N,k)})$.
3. Construct $[(\mathsf{qx}_i, \mathsf{qw}_i)]_{i=1}^m$ as in the PCD prover $\mathbb{P}$.
4. Output $(R_{\mathrm{V},\varphi}^{(\lambda,N,k)}, [(\mathsf{qx}_i, \mathsf{qw}_i)]_{i=1}^m, [\mathsf{acc}_i]_{i=1}^m)$.

Suppose that $\mathcal{A}$ outputs $(\varphi, z, z_{\mathsf{loc}}, [z_i, \pi_i, \mathsf{acc}_i]_{i=1}^m)$ such that the completeness precondition is satisfied, but the PCD verifier rejects, i.e., $\mathbb{V}(\mathsf{ivk}, z, (\pi, \mathsf{acc})) = 0$. Then, by construction of $\mathbb{V}$, it holds that either $\mathcal{V}(\mathrm{ivk}, (\mathsf{avk}, z, \mathsf{acc}.\mathbb{x}), \pi) = 0$ or $\mathrm{D}(\mathsf{dk}, \mathsf{acc}) = 0$. If $z_i = \bot$ for all i, then by perfect completeness of ARG both of these algorithms output 1; hence there exists i such that $z_i \neq \bot$. Hence it holds that for all i, $\mathbb{V}(\mathsf{ivk}, z_i, (\pi_i, \mathsf{acc}_i)) = 1$, whence for all i, $\mathcal{V}(\mathrm{ivk}, (\mathsf{avk}, z_i, \mathsf{acc}_i.\mathbb{x}), \pi_i) = \Phi_{\mathcal{V}}(\mathsf{pp}, R_{\mathrm{V},\varphi}^{(\lambda,N,k)}, (\mathsf{avk}, z_i, \mathsf{acc}_i.\mathbb{x}), \pi_i) = 1$ and $\mathrm{D}(\mathsf{dk}, \mathsf{acc}_i) = 1$.

If $\mathcal{V}(\mathrm{ivk}, (\mathsf{avk}, z, \mathsf{acc}.\mathbb{x}), \pi) = 0$, then, by perfect completeness of ARG, we know that $R_{\mathrm{V},\varphi}^{(\lambda,N,k)}$ rejects $\big((\mathsf{avk}, z, \mathsf{acc}), (z_{\mathsf{loc}}, [z_i, \pi_i.\mathbb{x}, \mathsf{acc}_i.\mathbb{x}]_{i=1}^m), \mathsf{pf}\big)$, and so $\mathrm{V}(\mathsf{avk}, [\mathsf{qx}_i]_{i=1}^m, [\mathsf{acc}_i.\mathbb{x}]_{i=1}^m, \mathsf{acc}.\mathbb{x}) = 0$. Otherwise, $\mathrm{D}(\mathsf{dk}, \mathsf{acc}) = 0$.

Now consider the completeness experiment for AS with adversary $\mathcal{B}$. Since $\mathsf{pp}, \mathsf{pp}_{\mathsf{AS}}$ are drawn identically to the PCD experiment, the distribution of the output of $\mathcal{A}$ is identical. Hence in particular

it holds that for all i, $\Phi_{\mathcal{V}}(\mathsf{pp}, R_{\mathrm{V},\varphi}^{(\lambda,N,k)}, (\mathsf{avk}, z_i, \mathsf{acc}_i), \pi_i) = 1$ and $\mathrm{D}(\mathsf{dk}, \mathsf{acc}_i) = 1$. By the above, it holds that either $\mathrm{V}(\mathsf{avk}, [\mathsf{qx}_i]_{i=1}^m, [\mathsf{acc}_i.\mathbb{x}]_{i=1}^m, \mathsf{acc}) = 0$ or $\mathrm{D}(\mathsf{dk}, \mathsf{acc}) = 0$, and so $\mathcal{B} := (\mathcal{B}_1, \mathcal{B}_2)$ causes the completeness condition for AS to be satisfied with probability at most p.

3.5.3 Knowledge soundness

The extracted transcript T will be a tree, so for convenience we associate the label $z^{(u,v)}$ of the unique outgoing edge of a node u with the node u itself, so that the node u is labelled with $(z^{(u)}, z_{\mathsf{loc}}^{(u)})$. In this proof we also associate with each node u a NARK proof $\pi^{(u)}$ and an accumulator $\mathsf{acc}^{(u)}$, so the full label for a node is $(z^{(u)}, z_{\mathsf{loc}}^{(u)}, \pi^{(u)}, \mathsf{acc}^{(u)})$. One can transform such a transcript into one that satisfies Definition 3.3.2.

Given a malicious prover $\tilde{\mathbb{P}}$, we will construct an extractor $\mathbb{E}_{\tilde{\mathbb{P}}}$ that satisfies knowledge soundness.

We do so via an iterative process that constructs a sequence of extractors $\mathbb{E}_1, \ldots, \mathbb{E}_d$ where d is the depth of φ and $\mathbb{E}_j$ outputs a tree of depth $j+1$. The extractor $\mathbb{E}_{\tilde{\mathbb{P}}}$ is then equal to $\mathbb{E}_d$.

In the base case, we define $\mathbb{E}_0(\mathbb{pp}, \mathsf{ai})$ to compute $(\varphi, \mathsf{o}, \pi, \mathsf{acc}) \leftarrow \tilde{\mathbb{P}}(\mathbb{pp}, \mathsf{ai})$ and output (φ, T_0), where T_0 is a single node labeled with $(\mathsf{o}, \pi, \mathsf{acc})$.

Next, we construct the extractor $\mathbb{E}_j$ inductively for each recursion depth $j \in [d]$, given that we have already constructed $\mathbb{E}_{j-1}$. We use the notation $l_{\mathsf{T}}(j)$ to denote the vertices of T at depth j (so that $l_{\mathsf{T}}(0) := \varnothing$ and $l_{\mathsf{T}}(1)$ is the singleton containing the root). We proceed in several steps.

- First, we construct a NARK prover $\tilde{\mathcal{P}}_j$ as follows:

  $\tilde{\mathcal{P}}_j(\mathsf{pp}, (\mathsf{pp}_{\mathsf{AS}}, \mathsf{ai}))$:
  1. Compute $(\varphi, \mathsf{T}_{j-1}, \mathsf{ao}) \leftarrow \mathbb{E}_{j-1}((\mathsf{pp}, \mathsf{pp}_{\mathsf{AS}}), \mathsf{ai})$.
  2. For each vertex $v \in l_{\mathsf{T}_{j-1}}(j)$, denote its label by $(z^{(v)}, \pi^{(v)}, \mathsf{acc}^{(v)})$.
  3. Run the argument indexer $(\mathrm{ipk}, \mathrm{ivk}) := \mathcal{I}(\mathsf{pp}, R_{\mathrm{V},\varphi}^{(\lambda,N,k)})$.
  4. Run the accumulator indexer $(\mathsf{apk}, \mathsf{dk}, \mathsf{avk}) := \mathrm{I}(\mathsf{pp}_{\mathsf{AS}}, \mathsf{pp}, R_{\mathrm{V},\varphi}^{(\lambda,N,k)})$.
  5. Output

  $$(\vec{\mathbb{i}}, \vec{\mathbb{x}}, \vec{\pi}, \mathsf{ao}') := \left(\vec{R}, (\mathsf{avk}, z^{(v)}, \mathsf{acc}^{(v)}.\mathbb{x})_{v \in l_{\mathsf{T}_{j-1}}(j)}, (\pi^{(v)})_{v \in l_{\mathsf{T}_{j-1}}(j)}, (\varphi, \mathsf{T}_{j-1}, \mathsf{ao})\right)$$

  where $\vec{R}$ is the vector $(R_{\mathrm{V},\varphi}^{(\lambda,N,k)}, \ldots, R_{\mathrm{V},\varphi}^{(\lambda,N,k)})$ of the appropriate length.

- Second, we let $\mathcal{E}_{\tilde{\mathcal{P}}_j}$ be the extractor that corresponds to $\tilde{\mathcal{P}}_j$, via the knowledge soundness of the non-interactive argument ARG.

- Third, we construct an accumulation scheme prover $\tilde{\mathrm{P}}_j$ as follows:

$\tilde{\mathrm{P}}_j(\mathsf{pp}_{\mathsf{AS}}, (\mathsf{pp}, \mathsf{ai}))$:

1. Run the extractor $(\vec{\mathbb{i}}, \vec{\mathbb{x}}, \vec{\mathbb{w}}, \mathsf{ao}') \leftarrow \mathcal{E}_{\tilde{\mathcal{P}}_j}(\mathsf{pp}, (\mathsf{pp}_{\mathsf{AS}}, \mathsf{ai}))$.
2. Parse the auxiliary output ao' as $(\varphi, \mathsf{T}', \mathsf{ao})$. If T' is not a transcript of depth j, abort.
3. For each vertex $v \in l_{\mathsf{T}'}(j)$,
   - obtain $\mathsf{acc}^{(v)}$ from T';
   - obtain the local data $z^{(v)}_{\mathsf{loc}}$, input messages $\left(z_i^{(v)}, \pi_i^{(v)}.\mathbb{x}, \mathsf{acc}_i^{(v)}.\mathbb{x}\right)_{i\in[m]}$ and accumulation proof $\mathsf{pf}^{(v)}$ from $\mathbb{w}^{(v)}$;
   - append $z^{(v)}_{\mathsf{loc}}$ to the label of v in T';
   - let $S_j := \{v \in l_{\mathsf{T}'}(j) : \exists i,\ z_i^{(v)} \neq \bot\}$;
   - attach m children to each $v \in S_j$, where the i-th child is labeled with $z_i^{(v)}$;
   - define $\mathsf{qx}_i^{(v)} := \left((\mathsf{avk}, z_i^{(v)}, \mathsf{acc}_i^{(v)}.\mathbb{x}), \pi_i^{(v)}.\mathbb{x}\right)$.
4. Output $\left(\left(\mathbb{i}^{(v)}, \mathsf{acc}^{(v)}, \mathsf{pf}^{(v)}, [\mathsf{qx}_i^{(v)}]_{i=1}^m, [\mathsf{acc}_i^{(v)}.\mathbb{x}]_{i=1}^m\right)_{v\in S_j}, (\varphi, \mathsf{T}', \mathsf{ao})\right)$.

- Fourth, we let $\mathrm{E}_{\tilde{\mathrm{P}}_j}$ be the extractor corresponding to $\tilde{\mathrm{P}}_j$, by the knowledge soundness of the split accumulation scheme AS.
- Finally, we define the extractor $\mathbb{E}_j$ as follows:

$\mathbb{E}_j(\mathbb{pp} = (\mathsf{pp}, \mathsf{pp}_{\mathsf{AS}}), \mathsf{ai})$:

1. Run the extractor $\left(\left(\mathbb{i}^{(v)}, \mathsf{acc}^{(v)}, [\mathsf{qx}_i^{(v)}, \mathsf{qw}_i^{(v)}]_{i=1}^m, [\mathsf{acc}_i^{(v)}]_{i=1}^m\right)_{v\in S_j}, \mathsf{ao}'\right) \leftarrow \mathrm{E}_{\tilde{\mathrm{P}}_j}(\mathsf{pp}, \mathsf{pp}_{\mathsf{AS}}, \mathsf{ai})$.
2. Parse the auxiliary output ao' as $(\varphi, \mathsf{T}', \mathsf{ao})$. If T' is not a transcript of depth j, abort.
3. Let $S_j := \{v \in l_{\mathsf{T}'}(j) : \exists i,\ z_i^{(v)} \neq \bot\}$.
4. Parse each $\mathsf{qx}_i^{(v)}$ as $((\mathsf{avk}^{(v)}, z_i^{(v)}, \mathsf{acc}_i^{(v)}.\mathbb{x}), \pi_i^{(v)}.\mathbb{x})$ and $\mathsf{qw}_i^{(v)}$ as $\pi_i^{(v)}.\mathbb{w}$; combine each pair $(\pi_i^{(v)}.\mathbb{x}, \pi_i^{(v)}.\mathbb{w})$ into a proof $\pi_i^{(v)}$.
5. Output $(\varphi, \mathsf{T}_j, \mathsf{ao})$ where T_j is the transcript constructed from T' by adding, for each vertex $v \in S_j$, $(\pi_i^{(v)}, \mathsf{acc}_i^{(v)})$ to the label of its i-th child.

We now show that $\mathbb{E}_{\tilde{\mathbb{P}}}$ runs in expected polynomial time and that it outputs a transcript that is φ-compliant.

Running time of the extractor. It follows from the extraction guarantees of ARG and AS that $\mathbb{E}_j$ runs in expected time polynomial in the expected running time of $\mathbb{E}_{j-1}$. Hence if $d(\varphi)$ is a constant, $\mathbb{E}_{\tilde{\mathbb{P}}} = \mathbb{E}_{d(\varphi)}$ runs in expected polynomial time.

Correctness of the extractor. Fix a set Z, and suppose that $\tilde{\mathbb{P}}$'s output falls in Z and causes $\mathbb{V}$ to accept, with probability μ. We show by induction that, for all $j \in \{0, \ldots, d\}$, the transcript T_j output by $\mathbb{E}_j$ is φ-compliant up to depth j, and that for all $v \in \mathsf{T}_j$, both $\mathcal{V}(\mathrm{ivk}, (\mathsf{avk}, z^{(v)}, \mathsf{acc}^{(v)}.\mathbb{x}), \pi^{(v)})$ and $\mathrm{D}(\mathsf{dk}, \mathsf{acc}^{(v)})$ accept, and that $(\mathbb{pp}, \mathsf{ai}, \varphi, \mathsf{o}(\mathsf{T}_j), \mathsf{ao}) \in Z$ and $\varphi \in \mathsf{F}$, with probability $\mu - \mathrm{negl}(\lambda)$.

For $j = 0$ the statement holds by assumption.

Now suppose that $(\varphi, \mathsf{T}_{j-1}) \leftarrow \mathbb{E}_{j-1}(\mathbb{pp}, \mathsf{ai})$ is such that T_{j-1} is φ-compliant up to depth $j-1$, and that both $\mathcal{V}(\mathrm{ivk}, (\mathsf{avk}, z^{(v)}, \mathsf{acc}^{(v)}.\mathbb{x}), \pi^{(v)})$ and $\mathrm{D}(\mathsf{dk}, \mathsf{acc}^{(v)})$ accept for all $v \in \mathsf{T}_{j-1}$ with probability $\mu - \mathrm{negl}(\lambda)$.

Let $(\vec{\mathbb{i}}, (\mathsf{avk}_v, z^{(v)}, \mathsf{acc}^{(v)}.\mathbb{x})_v, (\pi^{(v)})_v, (\varphi, \mathsf{T}'), \vec{\mathbb{w}})$ be the output of $\mathcal{E}_{\tilde{\mathcal{P}}_j}(\mathsf{pp}, (\mathsf{pp}_{\mathsf{AS}}, \mathsf{ai}))$.
We let $(\mathsf{pp}, (\mathsf{pp}_{\mathsf{AS}}, \mathsf{ai}), \vec{\mathbb{i}}, (\mathsf{avk}_v, z^{(v)}, \mathsf{acc}^{(v)}.\mathbb{x})_v, (\varphi, \mathsf{T}', \mathsf{ao})) \in Z'$ if and only if, for $(\mathsf{apk}, \mathsf{dk}, \mathsf{avk}) \leftarrow \mathrm{I}(\mathsf{pp}_{\mathsf{AS}}, \mathsf{pp}, R^{(\lambda,N,k)}_{\mathrm{V},\varphi})$ it holds that:

- $((\mathsf{pp}, \mathsf{pp}_{\mathsf{AS}}), \mathsf{ai}, \varphi, \mathsf{o}(\mathsf{T}'), \mathsf{ao}) \in Z$ and $\varphi \in \mathsf{F}$;
- $\mathbb{i}^{(v)} = R^{(\lambda,N,k)}_{\mathrm{V},\varphi}$ and $\mathsf{avk}_v = \mathsf{avk}$ for all v;
- T' is φ-compliant up to depth $j-1$;
- $\mathrm{D}(\mathsf{dk}, \mathsf{acc}^{(v)})$ accepts for all $v \in \mathsf{T}'$; and
- for $v \in l_{\mathsf{T}'}(j)$, v is labeled in T' with $(z^{(v)}, \pi^{(v)}, \mathsf{acc}^{(v)})$.

By knowledge soundness, with probability $\mu - \mathrm{negl}(\lambda)$, $(\mathsf{pp}, (\mathsf{pp}_{\mathsf{AS}}, \mathsf{ai}), \vec{\mathbb{i}}, (\mathrm{ivk}_v, z^{(v)})_v, (\varphi, \mathsf{T}')) \in Z'$ and for every vertex $v \in l_{\mathsf{T}'}(j)$, $(R^{(\lambda,N,k)}_{\mathrm{V},\varphi}, (\mathsf{avk}_v, z^{(v)}, \mathsf{acc}^{(v)}), \mathbb{w}^{(v)}) \in \mathcal{R}_{\mathrm{R1CS}}$. Here we use Z' and the auxiliary output in the knowledge soundness definition of ARG to ensure consistency between the values $z^{(v)}$ and T', and to ensure that T' is φ-compliant and that the decider accepts.

Consider some $v \in l_{\mathsf{T}'}(j)$. Since $(R^{(\lambda,N,k)}_{\mathrm{V},\varphi}, (\mathsf{avk}^{(v)}, z^{(v)}, \mathsf{acc}^{(v)}.\mathbb{x}), \mathbb{w}^{(v)}) \in \mathcal{R}_{\mathrm{R1CS}}$, we obtain from $\mathbb{w}^{(v)}$ either:

- local data $z^{(v)}_{\mathsf{loc}}$, input messages $\left(z^{(v)}_i, \pi^{(v)}_i.\mathbb{x}, \mathsf{acc}^{(v)}_i.\mathbb{x}\right)_{i\in[m]}$ and proof pf such that the PCD predicate $\varphi(z^{(v)}, z_{\mathsf{loc}}, z_1, \ldots, z_m)$ accepts, and the accumulation verifier $\mathrm{V}^{(\lambda,N,k)}(\mathsf{avk}^{(v)}, [\mathsf{qx}^{(v)}_i]^m_{i=1}, [\mathsf{acc}^{(v)}_i.\mathbb{x}]^m_{i=1}, \mathsf{acc}^{(v)}, \mathsf{pf}^{(v)})$ accepts, where $\mathsf{qx}^{(v)}_i := ((\mathsf{avk}^{(v)}, z^{(v)}_i, \mathsf{acc}^{(v)}_i.\mathbb{x}), \pi^{(v)}_i.\mathbb{x})$; or
- local data $z^{(v)}_{\mathsf{loc}}$ such that $\varphi(z^{(v)}, z^{(v)}_{\mathsf{loc}}, \bot, \ldots, \bot)$ accepts.

In both cases we append $z^{(v)}_{\mathsf{loc}}$ to the label of v. In the latter case, v has no children and so is φ-compliant by the base case condition. In the former case we label the children of v with $(z_i, \pi_i, \mathsf{acc}_i)$, and so v is φ-compliant.

We define $(\mathsf{pp}_{\mathsf{AS}}, \mathsf{pp}, \mathsf{ai}, (\mathbb{i}^{(v)}, \mathsf{acc}^{(v)}, [\mathsf{qx}^{(v)}_i]^m_{i=1}, [\mathsf{acc}^{(v)}_i.\mathbb{x}]^m_{i=1})_v, (\varphi, \mathsf{T}', \mathsf{ao})) \in Z''$ if and only if

- $((\mathsf{pp}, \mathsf{pp}_{\mathsf{AS}}), \mathsf{ai}, \varphi, \mathsf{o}(\mathsf{T}'), \mathsf{ao}) \in Z$ and $\varphi \in \mathsf{F}$,
- $\mathbb{i}^{(v)} = R^{(\lambda,N,k)}_{\mathrm{V},\varphi}$ for all v,
- T' is φ-compliant up to depth j,
- for all v, $\mathsf{qx}^{(v)}_i = ((\mathsf{avk}, z^{(v)}_i, \mathsf{acc}^{(v)}_i.\mathbb{x}), \pi^{(v)}_i)$ where $(\mathsf{apk}, \mathsf{avk}, \mathsf{dk}) \leftarrow \mathrm{I}(\mathsf{pp}_{\mathsf{AS}}, \mathsf{pp}_\Phi, \mathsf{i}_\Phi)$, and
- for $u \in l_{\mathsf{T}'}(j+1)$, where u is the i-th child of $v \in l_{\mathsf{T}'}(j)$, u is labeled in T' with $z^{(v)}_i$.

Let $\left(\left(\mathbb{i}^{(v)}, \mathsf{acc}^{(v)}, [\mathsf{qx}^{(v)}_i, \mathsf{qw}^{(v)}_i]^m_{i=1}, [\mathsf{acc}^{(v)}_i]^m_{i=1}\right)_{v\in S_j}, \mathsf{ao}'\right) \leftarrow \mathrm{E}_{\tilde{\mathrm{P}}_j}(\mathsf{pp}_{\mathsf{AS}}, \mathsf{pp}, \mathsf{ai})$. By knowledge soundness of the accumulation scheme, $(\mathsf{pp}, \mathsf{pp}_\Phi, \mathsf{ai}, (\mathbb{i}^{(v)}, \mathsf{acc}^{(v)}, [\mathsf{qx}^{(v)}_i]^m_{i=1}, [\mathsf{acc}^{(v)}_i.\mathbb{x}]^m_{i=1})_v, \mathsf{ao}') \in Z''$, and it holds that for all descendants u of v in T_j, $\mathrm{D}(\mathsf{dk}, \mathsf{acc}^{(u)})$ accepts and the predicate $\Phi_{\mathcal{V}}(\mathsf{pp}, R^{(\lambda,N,k)}_{\mathrm{V},\varphi}, (\mathsf{avk}, z^{(u)}, \mathsf{acc}^{(u)}.\mathbb{x}), \pi^{(u)}_{\mathsf{in}}) = \mathcal{V}(\mathrm{ivk}, (\mathsf{avk}, z^{(u)}, \mathsf{acc}^{(u)}.\mathbb{x}), \pi^{(u)}_{\mathsf{in}})$ accepts, with probability $\mu - \mathrm{negl}(\lambda)$; this completes the inductive step.

Hence by induction, $(\varphi, \mathsf{T}, \mathsf{ao}) \leftarrow \mathbb{E}(\mathbb{pp}, \mathsf{ai})$ has φ-compliant T, $(\mathbb{pp}, \mathsf{ai}, \varphi, \mathsf{o}(\mathsf{T}), \mathsf{ao}) \in Z$, and $\varphi \in \mathsf{F}$, with probability $\mu - \mathrm{negl}(\lambda)$.

3.5.4 Zero knowledge

The simulator $\mathbb{S}$ operates as follows.

$\mathbb{S}(1^\lambda)$:
1. Sample simulated parameters for the non-interactive argument: $(\mathsf{pp}, \tau) \leftarrow \mathcal{S}(1^\lambda)$.
2. Sample simulated parameters for the accumulation scheme: $(\mathsf{pp}_{\mathsf{AS}}, \tau_{\mathsf{AS}}) \leftarrow \mathrm{S}(1^\lambda)$.
3. Output $(\mathbb{pp} := (\mathsf{pp}, \mathsf{pp}_{\mathsf{AS}}), (\mathsf{pp}, \mathsf{pp}_{\mathsf{AS}}, \tau, \tau_{\mathsf{AS}}))$.

$\mathbb{S}((\mathsf{pp}, \mathsf{pp}_{\mathsf{AS}}, \tau, \tau_{\mathsf{AS}}), \varphi, z)$:
1. Compute accumulator keys: $(\mathsf{apk}, \mathsf{dk}, \mathsf{avk}) := \mathrm{I}(\mathsf{pp}_{\mathsf{AS}}, \mathsf{pp}_\Phi = \mathsf{pp}, \mathsf{i}_\Phi = R_{\mathrm{V},\varphi}^{(\lambda,N,k)})$.
2. Sample simulated accumulator: $\mathsf{acc} \leftarrow \mathrm{S}(\tau_{\mathsf{AS}}, \mathsf{pp}_\Phi = \mathsf{pp}, \mathsf{i}_\Phi = R_{\mathrm{V},\varphi}^{(\lambda,N,k)})$.
3. Sample simulated argument: $\pi \leftarrow \mathcal{S}(\tau, R_{\mathrm{V},\varphi}^{(\lambda,N,k)}, (\mathsf{avk}, z, \mathsf{acc}.\mathbb{x}))$.
4. Output (π, acc).

We consider the following sequence of hybrids.

- $\mathbf{H}_0$: The original experiment.
- $\mathbf{H}_1$: As $\mathbf{H}_0$, but the public parameters pp and proof π are generated by the simulator $\mathcal{S}$ for ARG.
- $\mathbf{H}_2$: As $\mathbf{H}_1$, but the public parameters $\mathsf{pp}_{\mathsf{AS}}$ and accumulator acc is generated by the simulator S for AS.

We need to argue that $\mathbf{H}_0$ and $\mathbf{H}_2$ are indistinguishable.

Since $\mathcal{A}$ is honest (for PCD), by completeness of AS it induces an honest adversary for ARG, whence $\mathbf{H}_0$ and $\mathbf{H}_1$ are indistinguishable by the zero knowledge property of ARG. Note that since they are part of the witness, the input and accumulator lists $[(\mathsf{qx}_i, \mathsf{qw}_i)]_{i=1}^n, [\mathsf{acc}_j]_{j=1}^m$ and verifier proof pf are not used in $\mathbf{H}_1$. Hence, since $\mathcal{A}$ induces an honest adversary for AS and the simulated pp is indistinguishable from the real pp (sampled by $\mathcal{G}(1^\lambda)$), $\mathbf{H}_1$ and $\mathbf{H}_2$ are indistinguishable by the zero knowledge property of AS.

3.5.5 Efficiency

The efficiency argument follows from Lemma 3.5.5 and is essentially identical to that of [BCMS20], and so we will not repeat it. We note only that the quantity v^* (i) describes the size of the *accumulation* verifier, which in particular need not read the entire NARK proof, which may be large, and (ii) is a function of the size of the accumulator *instance* alone; the accumulator witness may be large.

Lemma 3.5.5. *Suppose that for every security parameter $\lambda \in \mathbb{N}$, arity m, and message size $\ell \in \mathbb{N}$ the ratio of accumulation verifier circuit size to index size $\mathsf{v}^*(\lambda, m, N, \ell)/N$ is monotone decreasing in N. Then there exists a size function $N(\lambda, f, m, \ell)$ such that*

$$\forall\, \lambda, f, m, \ell \in \mathbb{N} \quad S(\lambda, f, m, \ell, N(\lambda, f, m, \ell)) \leq N(\lambda, f, m, \ell) \ .$$

Moreover if for some $\epsilon > 0$ and some increasing function α it holds that, for all N, λ, m, ℓ sufficiently large,

$$\mathsf{v}^*(\lambda, m, N, \ell) \leq N^{1-\epsilon}\alpha(\lambda, m, \ell)$$

then, for all λ, m, ℓ sufficiently large, $N(\lambda, f, m, \ell) \leq O(f + \alpha(\lambda, m, \ell)^{1/\epsilon})$.

3.5.6 Post-quantum security

We consider post-quantum knowledge soundness and zero knowledge.

Knowledge soundness. In the quantum setting, $\tilde{\mathbb{P}}$ is taken to be a polynomial-size *quantum* circuit; hence also $\tilde{\mathcal{P}}_j, \mathcal{E}_{\tilde{\mathcal{P}}_j}, \tilde{\mathrm{P}}_j, \mathrm{E}_{\tilde{\mathrm{P}}_j}, \mathbb{E}_j$ are quantum circuits for all j, as is the final extractor $\mathbb{E}$. Our definition of knowledge soundness is such that our proof then generalizes immediately to show security against quantum adversaries. In particular, the only difficulty arising from quantum adversaries is that they can generate their own randomness, whereas in the classical case we can force an adversary to behave deterministically by fixing its randomness. This difference is resolved by our strong adaptive knowledge extraction property, which we use to enforce that the extractor's output is consistent with the transcript obtained so far.

Zero knowledge. From the argument in the preceding section it is clear that, by modifying the definitions of zero knowledge as appropriate for the quantum setting, if ARG and AS both achieve post-quantum zero knowledge, then so does PCD.

3.6 An expected-time forking lemma

We establish useful notation for algorithms with access to oracles (Section 3.6.1), and then provide an expected-time forking lemma with negligible loss (Section 3.6.2). We use this technical lemma to prove the security of split accumulation schemes in later sections.

3.6.1 Notation for oracle algorithms

Let A be a t-query oracle algorithm with access to an oracle $\rho\colon \{0,1\}^* \to \{0,1\}^\lambda$. For $\vec{\mathsf{a}} = (\mathsf{a}_1, \ldots, \mathsf{a}_t) \in (\{0,1\}^\lambda)^t$, we denote by $(\mathsf{q}, \mathsf{o}; \mathsf{tr}, r) \leftarrow A^{\vec{\mathsf{a}}}(x)$ the following procedure: run A on input x, and answer the i-th query q_i of A to its oracle with a_i for each i; output $(\mathsf{q}, \mathsf{o}; \mathsf{tr}, r)$, where r is the randomness used by A. We write $(\mathsf{q}, \mathsf{o}; \mathsf{tr}, r) \leftarrow A^{\rho}(x)$ to denote the same procedure when each a_i is adaptively set to $\rho(\mathsf{q}_i)$.

We assume without loss of generality that A makes no duplicate queries; in particular, we can interpret tr as partial function $\mathsf{tr}\colon \{0,1\}^* \rightharpoonup \{0,1\}^\lambda$. For a query transcript $\mathsf{tr} = [(\mathsf{q}_i, \mathsf{a}_i)]_{i=1}^t$ and query q, if $\mathsf{q} = \mathsf{q}_j$ for some $j \in [t]$ then let j be the smallest such index, and define $\mathsf{tr}_\mathsf{q} := [(\mathsf{q}_i, \mathsf{a}_i)]_{i=1}^{j-1}$. That is, tr is truncated to the query *before* the first query to q. If q does not appear in tr, define $\mathsf{tr}_\mathsf{q} := \bot$.

3.6.2 An expected-time forking lemma

We give an expected-time forking lemma that is suitable for our setting. In particular, it handles adversaries with an expected running time guarantee, which is a requirement of our knowledge soundness definition.

Lemma 3.6.1. *Let* p *be a predicate computable in time* t_p*. There exists an algorithm* Fork *such that for every public parameter string* $\mathsf{pp} \in \{0,1\}^{\mathrm{poly}(\lambda)}$ *and oracle algorithm* A*,*

$$\Pr\left[\begin{array}{c|r} \mathsf{tr}_\mathsf{q} \neq \bot \wedge \mathsf{p}(\mathsf{pp}, (\mathsf{q}, \rho(\mathsf{q})), \mathsf{o}, \mathsf{tr}_\mathsf{q}) = 1 & \rho \leftarrow \mathcal{U}(\lambda) \\ \Downarrow & (\mathsf{q}, \mathsf{o}; \mathsf{tr}, r) \leftarrow A^{\rho}(\mathsf{pp}) \\ \forall j \in [N],\ \mathsf{p}(\mathsf{pp}, (\mathsf{q}, \mathsf{b}_j), \mathsf{o}_j, \mathsf{tr}_\mathsf{q}) = 1 & [\mathsf{b}_j, \mathsf{o}_j]_{j=1}^N \leftarrow \mathsf{Fork}^A(\mathsf{pp}, 1^N, (\mathsf{q}, \rho(\mathsf{q})), \mathsf{o}, \mathsf{tr}_\mathsf{q}, r) \\ \wedge\ \mathsf{b}_1, \ldots, \mathsf{b}_N \textit{ are pairwise distinct} & \end{array}\right] \geq 1 - \frac{2N\sqrt{t}}{2^{\lambda/2}} \ .$$

In the above experiment, Fork *runs in expected time* $O(tN \cdot (t_A + t_\mathsf{p}))$*, where* t *is a strict bound on the number of oracle queries made by* A *and* t_A *is its* **expected** *running time.*

Proof. The algorithm Fork on input $(\mathsf{pp}, 1^N, (\mathsf{q}, \mathsf{a}), \mathsf{o}, \mathsf{tr}, r)$ operates as follows.

1. If $\mathsf{tr} = \bot$ or $\mathsf{p}(\mathsf{pp}, (\mathsf{q}, \mathsf{a}), \mathsf{o}, \mathsf{tr}) = 0$, output $\bot$.
2. Parse tr as $[(\mathsf{q}_1, \mathsf{a}_1), \ldots, (\mathsf{q}_{i-1}, \mathsf{a}_{i-1})]$.
3. Set $\mathsf{b}_1 := \mathsf{a}$ and $\mathsf{o}_1 := \mathsf{o}$.
4. Set $J := 1$ and repeat the following until $J = N$:
   a) Draw $\mathsf{a}'_i, \ldots, \mathsf{a}'_t \leftarrow \{0,1\}^\lambda$.
   b) Run $A^{\mathsf{a}_1, \ldots, \mathsf{a}_{i-1}, \mathsf{a}'_i, \ldots, \mathsf{a}'_t}(\mathsf{pp}; r)$ until it halts and outputs $(\mathsf{q}', \mathsf{o}'; \mathsf{tr}', r')$. If r' is longer than r, set $r := r'$.

c) If $\mathsf{q}' = \mathsf{q}$ (in particular, $\mathsf{tr}'_{\mathsf{q}} = \mathsf{tr}_{\mathsf{q}}$) and $\mathsf{p}(\mathsf{pp}, (\mathsf{q}, \mathfrak{a}'_i), \mathsf{o}', \mathsf{tr}) = 1$, set $J := J + 1$, $\mathfrak{b}_J := \mathfrak{a}'_i$, and $\mathsf{o}_J := \mathsf{o}'$.

5. Output $(\mathfrak{b}_1, \mathsf{o}_1, \ldots, \mathfrak{b}_N, \mathsf{o}_N)$.

For the purposes of analysis, we consider an experiment where both A and Fork obtain their randomness from a shared infinite tape $\sigma \in \{0,1\}^*$. This is indistinguishable from the real experiment since we can view the (common) randomness generated by all runs of A as being the prefix of σ.

Let $S_i := \{(\vec{\mathfrak{a}}, \sigma) : (\mathsf{q}, \mathsf{o}; \mathsf{tr}) \leftarrow A^{\vec{\mathfrak{a}}}(\mathsf{pp}; \sigma) \wedge |\mathsf{tr}_{\mathsf{q}}| = i \wedge \mathsf{p}(\mathsf{pp}, (\mathsf{q}, \mathfrak{a}_i), \mathsf{o}, \mathsf{tr}_{\mathsf{q}}) = 1\}$. Define

$$\delta_i(\mathfrak{a}_1, \ldots, \mathfrak{a}_{i-1}; \sigma) := \Pr_{\mathfrak{a}'_i, \ldots, \mathfrak{a}'_t \in \{0,1\}^\lambda}[(\mathfrak{a}_1, \ldots, \mathfrak{a}_{i-1}, \mathfrak{a}'_i, \ldots, \mathfrak{a}'_t, \sigma) \in S_i] \ .$$

Observe that if $(\vec{\mathfrak{a}}, \sigma) \in S_i$ then the probability that one iteration of Step 4 increments J is $\delta_i(\mathfrak{a}_1, \ldots, \mathfrak{a}_{i-1}; \sigma)$. If the precondition on the left of the probability statement holds then Fork does not terminate in Step 1. In this case $\delta_i(\mathfrak{a}_1, \ldots, \mathfrak{a}_{i-1}; \sigma) > 0$, and Fork's output (if it halts) satisfies "$\forall j \in [N],\ \mathsf{p}(\mathsf{pp}, (\mathsf{q}, \mathfrak{b}_j), \mathsf{o}_j, \mathsf{tr}_{\mathsf{q}}) = 1$".

We now bound the expected running time of Fork. Let T_A, T_{Fork} be random variables denoting the running time of A, Fork respectively, and let $t(\mathfrak{a}_1, \ldots, \mathfrak{a}_{i-1}; \sigma)$ denote the expected running time of a single iteration of Step 4. The number of iterations between $J = j$ and $J = j + 1$, which we denote $X^{(j)}$, is geometrically distributed with parameter $\delta_i(\mathfrak{a}_1, \ldots, \mathfrak{a}_{i-1}; \sigma) > 0$ when $(\vec{\mathfrak{a}}, \sigma) \in S_i$. We denote the *time* between these increments of J by $T^{(j)}$ and note that, when $(\vec{\mathfrak{a}}; \sigma) \in S_i$, $\mathbb{E}[T^{(j)} \mid X^{(j)} = m] = m \cdot t(\mathfrak{a}_1, \ldots, \mathfrak{a}_{i-1}; \sigma)$ by linearity. Let

$$f(\mathfrak{a}_1, \ldots, \mathfrak{a}_{i-1}; \sigma) := \begin{cases} \frac{t(\mathfrak{a}_1, \ldots, \mathfrak{a}_{i-1}; \sigma)}{\delta_i(\mathfrak{a}_1, \ldots, \mathfrak{a}_{i-1}; \sigma)} & \text{if } \delta_i(\mathfrak{a}_1, \ldots, \mathfrak{a}_{i-1}; \sigma) \neq 0 \\ 0 & \text{otherwise} \end{cases} .$$

By the law of total expectation:

$$
\begin{aligned}
&\mathbb{E}[T_{\mathsf{Fork}}] \\
&= \mathbb{E}_{\vec{\mathfrak{a}},\sigma}\left[\,\mathbb{E}[T_{\mathsf{Fork}} \mid (\vec{\mathfrak{a}},\sigma)]\,\right] \\
&= \mathbb{E}_{\vec{\mathfrak{a}},\sigma}\left[\sum_{j=1}^{N}\sum_{m=1}^{\infty}\Pr[X^{(j)}=m \mid (\vec{\mathfrak{a}},\sigma)]\cdot\mathbb{E}[T^{(j)} \mid X^{(j)}=m,(\vec{\mathfrak{a}},\sigma)]\right] \\
&= \frac{N}{2^{t\lambda}}\cdot\sum_{i=1}^{t}\mathbb{E}_{\sigma}\left[\sum_{\vec{\mathfrak{a}}\text{ s.t. }(\vec{\mathfrak{a}},\sigma)\in S_i}\sum_{m=1}^{\infty}\left(1-\delta_i(\mathfrak{a}_1,\ldots,\mathfrak{a}_{i-1};\sigma)\right)^{m-1}\cdot\delta_i(\mathfrak{a}_1,\ldots,\mathfrak{a}_{i-1};\sigma)\cdot m\cdot t(\mathfrak{a}_1,\ldots,\mathfrak{a}_{i-1};\sigma)\right] \\
&= \frac{N}{2^{t\lambda}}\cdot\sum_{i=1}^{t}\mathbb{E}_{\sigma}\left[\sum_{\vec{\mathfrak{a}}\text{ s.t. }(\vec{\mathfrak{a}},\sigma)\in S_i}\frac{t(\mathfrak{a}_1,\ldots,\mathfrak{a}_{i-1};\sigma)}{\delta_i(\mathfrak{a}_1,\ldots,\mathfrak{a}_{i-1};\sigma)}\right] \\
&= \frac{N}{2^{t\lambda}}\cdot\sum_{i=1}^{t}\mathbb{E}_{\sigma}\left[\sum_{\vec{\mathfrak{a}}\text{ s.t. }(\vec{\mathfrak{a}},\sigma)\in S_i}f(\mathfrak{a}_1,\ldots,\mathfrak{a}_{i-1};\sigma)\right] \\
&\leq N\cdot\sum_{i=1}^{t}\mathbb{E}_{\mathfrak{a}_1,\ldots,\mathfrak{a}_{i-1},\sigma}[t(\mathfrak{a}_1,\ldots,\mathfrak{a}_{i-1};\sigma)] \\
&= N\cdot t\cdot(\mathbb{E}[T_A]+t_{\mathsf{p}}+O(t))\ .
\end{aligned}
$$

Above the inequality follows because for all functions $f(\mathfrak{a}_1,\ldots,\mathfrak{a}_{i-1};\sigma)$ into $\mathbb{R}$ and $\sigma\in\{0,1\}^*$,

$$
\begin{aligned}
\sum_{\vec{\mathfrak{a}}\text{ s.t. }(\vec{\mathfrak{a}},\sigma)\in S_i}f(\mathfrak{a}_1,\ldots,\mathfrak{a}_{i-1};\sigma) &= \sum_{\mathfrak{a}_1,\ldots,\mathfrak{a}_{i-1}}f(\mathfrak{a}_1,\ldots,\mathfrak{a}_{i-1};\sigma)\sum_{\mathfrak{a}_i,\ldots,\mathfrak{a}_t}1_{S_i}(\vec{\mathfrak{a}},\sigma) \\
&= 2^{(t-i+1)\lambda}\sum_{\mathfrak{a}_1,\ldots,\mathfrak{a}_{i-1}}f(\mathfrak{a}_1,\ldots,\mathfrak{a}_{i-1};\sigma)\cdot\delta_i(\mathfrak{a}_1,\ldots,\mathfrak{a}_{i-1};\sigma)\ .
\end{aligned}
$$

It remains to show that the $\mathfrak{b}_1,\ldots,\mathfrak{b}_N$ are pairwise distinct. Similarly to the above, it can be shown that the expected number of iterations is at most Nt, and so the probability that Fork performs more than $\sqrt{t}\cdot 2^{\lambda/2}$ iterations is at most $\frac{Nt}{\sqrt{t}\cdot 2^{\lambda/2}}=\frac{N\sqrt{t}}{2^{\lambda/2}}$. Conditioned on this, the probability that in any iteration we draw $\mathfrak{a}'_i$ such that $\mathfrak{a}'_i=\mathfrak{b}_j$ for any $j<J$ is at most $\frac{N\sqrt{t}2^{\lambda/2}}{2^{\lambda}}=\frac{N\sqrt{t}}{2^{\lambda/2}}$. By a union bound we obtain that the probability that there exist two elements among $\mathfrak{b}_1,\ldots,\mathfrak{b}_N$ that are equal is at most $2\cdot\frac{N\sqrt{t}}{2^{\lambda/2}}$. □

The following corollary enables extraction from protocols with two sequential oracle queries; e.g., those arising from the Fiat–Shamir transformation applied to a five-message protocol. It is shown by "recursively" applying the above forking lemma to an adversary constructed using the Fork algorithm itself.

Corollary 1. *Let* p *be a predicate computable in time* t_{p}*. There exists an algorithm* Fork_2 *such that for all* $\mathsf{pp} \in \{0,1\}^{\mathrm{poly}(\lambda)}$ *and oracle algorithms* A*,*

$$\Pr\left[\begin{array}{c|r} \begin{array}{c} \mathsf{tr}_{\mathsf{q}} \neq \bot \wedge \\ \mathsf{p}(\mathsf{pp},(\mathsf{q},\rho(\mathsf{q})),\mathsf{o},\rho(\rho(\mathsf{q}),\mathsf{o}),\mathsf{o}',\mathsf{tr}_{\mathsf{q}}) = 1 \\ \Downarrow \\ \mathfrak{b}_1,\ldots,\mathfrak{b}_N \textit{ are pairwise distinct} \\ \wedge\, \forall j \in [N], \\ \mathsf{p}(\mathsf{pp},(\mathsf{q},\mathfrak{b}_j),\mathsf{o}_j,\mathfrak{b}'_{j,k},\mathsf{o}'_{j,k},\mathsf{tr}_{\mathsf{q}}) = 1 \\ \wedge\, \mathfrak{b}'_{j,1},\ldots,\mathfrak{b}'_{j,N'} \textit{ are pairwise distinct} \end{array} & \begin{array}{r} \rho \leftarrow \mathcal{U}(\lambda) \\ (\mathsf{q},\mathsf{o},\mathsf{o}';\mathsf{tr},r) \leftarrow A^{\rho}(\mathsf{pp}) \\ [\mathfrak{b}_j,\mathsf{o}_j,[\mathfrak{b}'_{j,k},\mathsf{o}'_{j,k}]_{k=1}^{N'}]_{j=1}^{N} \\ \leftarrow \mathsf{Fork}_2^{A}(\mathsf{pp},1^N,1^{N'},(\mathsf{q},\rho(\mathsf{q})),\mathsf{o}, \\ \rho(\rho(\mathsf{q}),\mathsf{o}),\mathsf{o}',\mathsf{tr},r) \end{array} \end{array}\right] \geq 1 - \frac{3NN'\sqrt{t}}{2^{\lambda/2}}$$

In the above experiment, Fork_2 *runs in expected time* $O(t^2NN' \cdot (t_A + t_{\mathsf{p}}))$*, where* t *is a strict bound on the number of oracle queries made by* A *and* t_A *is its* **expected** *running time.*

3.7 Split accumulation for Hadamard products

We construct a split accumulation scheme for the Hadamard products. We define the predicate to accumulate and then state our theorem. The remainder of the section is dedicated to proving the theorem.

Definition 3.7.1. *The* **Hadamard product predicate** Φ_{HP} *takes as input: (i) public parameters* $\mathsf{pp}_\Phi = \mathsf{pp}_{\mathsf{CM}}$ *for the Pedersen commitment scheme (for messages of some maximum length L); (ii) an index* $\mathsf{i}_\Phi = \ell$ *specifying a message length (at most L); (iii) an instance* $\mathsf{qx} = (C_1, C_2, C_3) \in \mathbb{G}^3$ *consisting of three Pedersen commitments; (iv) a witness* $\mathsf{qw} = (a, b, \omega_1, \omega_2, \omega_3)$ *consisting of two vectors* $a, b \in \mathbb{F}^\ell$ *and three opening randomness elements* $\omega_1, \omega_2, \omega_3 \in \mathbb{F}$*. The predicate* Φ_{HP} *computes the commitment key* $\mathsf{ck} := \mathsf{CM.Trim}(\mathsf{pp}_{\mathsf{CM}}, \ell)$ *for messages of length ℓ and checks that*

$$C_1 = \mathsf{CM.Commit}(\mathsf{ck}, a; \omega_1) \ \wedge \ C_2 = \mathsf{CM.Commit}(\mathsf{ck}, b; \omega_2) \ \wedge \ C_3 = \mathsf{CM.Commit}(\mathsf{ck}, a \circ b; \omega_3) \ . \tag{3.1}$$

Theorem 3.7.2. *The scheme* $\mathsf{AS} = (\mathrm{G}, \mathrm{I}, \mathrm{P}, \mathrm{V}, \mathrm{D})$ *constructed in Section 3.7.1 is a zero-knowledge split accumulation scheme in the random oracle model for the Hadamard product predicate in Definition 3.7.1.* AS *achieves the efficiency stated below.*

- Generator: $\mathrm{G}(1^\lambda)$ *runs in time* $O(\lambda)$.
- Indexer: *The time of* $\mathrm{I}(\mathsf{pp}, \mathsf{pp}_\Phi, \mathsf{i}_\Phi = \ell)$ *is dominated by the time to run* CM.Trim *with message length ℓ.*
- Accumulation prover: *The time of* $\mathrm{P}^\rho(\mathsf{apk}, [(\mathsf{qx}_i, \mathsf{qw}_i)]_{i=1}^n, [\mathsf{acc}_j]_{j=1}^m)$ *is dominated by* $O(n+m) \cdot \ell$ *group scalar multiplications and* $\tilde{O}(n+m) \cdot \ell$ *field additions/multiplications.*
- Accumulation verifier: $\mathrm{V}^\rho(\mathsf{avk}, [\mathsf{qx}_i]_{i=1}^n, [\mathsf{acc}_j.\mathbb{x}]_{j=1}^m, \mathsf{acc}.\mathbb{x}, \mathsf{pf})$ *requires making* 2 *calls to the random oracle,* $O(n+m)$ *field additions/multiplications, and* $O(n+m)$ *group scalar multiplications.*
- Decider: *The time of* $\mathrm{D}(\mathsf{dk}, \mathsf{acc})$ *equals the time to run the predicate* Φ_{HP}.
- Sizes: *An accumulator* acc *is split into an accumulator instance* $\mathsf{acc}.\mathbb{x}$ *of* 3 *group elements, and an accumulator witness* $\mathsf{acc}.\mathbb{w}$ *of* $O(\ell)$ *field elements. An accumulation proof* pf *consists of* $O(n+m)$ *group elements.*

3.7.1 Construction

We describe the accumulation scheme $\mathsf{AS} = (\mathrm{G}, \mathrm{I}, \mathrm{P}, \mathrm{V}, \mathrm{D})$ for the Hadamard product predicate Φ_{HP}. An accumulator acc is split in two parts that are analogous to instance-witness pairs given to Φ_{HP} (see Definition 3.7.1). Jumping ahead, the decider D is equal to the predicate Φ_{HP}; hence, there is no distinction between inputs and prior accumulators, and so it suffices to accumulate inputs only.

Generator. The generator G receives as input $\mathsf{pp} := 1^\lambda$ and outputs 1^λ. (In other words, G does not have to create additional public parameters beyond those used by Φ_{HP}.)

Indexer. On input the accumulator parameters pp, predicate parameters $\mathsf{pp}_\Phi = \mathsf{pp}_{\mathsf{CM}}$, and a predicate index $\mathsf{i}_\Phi = \ell$, the indexer I computes the commitment key $\mathsf{ck} := \mathsf{CM.Trim}(\mathsf{pp}_{\mathsf{CM}}, \ell)$, and

then outputs the accumulator proving key $\mathsf{apk} := (\mathsf{ck}, \ell)$, the accumulator verification key $\mathsf{avk} := \ell$, and the decision key $\mathsf{dk} := \mathsf{ck}$.

Accumulation prover. On input the accumulation proving key apk and predicate instance-witness pairs $[(\mathsf{qx}_i, \mathsf{qw}_i)]_{i=1}^n$ (of the same form as split accumulators $[\mathsf{acc}_j]_{j=1}^m = [(\mathsf{acc}_j.\mathbb{x}, \mathsf{acc}_j.\mathbb{w})]_{j=1}^m$), P works as below.

$\mathrm{P}^\rho(\mathsf{apk} = (\mathsf{ck}, \ell), [(\mathsf{qx}_i, \mathsf{qw}_i)]_{i=1}^n)$:

1. For each $i \in [n]$, parse the predicate instance qx_i as $(C_{1,i}, C_{2,i}, C_{3,i})$.
2. For each $i \in [n]$, parse the predicate witness qw_i as $(a_i, b_i, \omega_{1,i}, \omega_{2,i}, \omega_{3,i})$.
3. Sample $a^\star, b^\star \in \mathbb{F}^\ell$ and $\omega_1^\star, \omega_2^\star, \omega_3^\star \in \mathbb{F}$ and compute

$$C_1^\star := \mathsf{CM.Commit}(\mathsf{ck}, a^\star; \omega_1^\star) \ ,$$
$$C_2^\star := \mathsf{CM.Commit}(\mathsf{ck}, b^\star; \omega_2^\star) \ ,$$
$$C_3^\star := \mathsf{CM.Commit}(\mathsf{ck}, a^\star \circ b_1 + a_n \circ b^\star; \omega_3^\star) \ .$$

4. Use the random oracle to compute the challenge $\mu := \rho(\ell, [\mathsf{qx}_i]_{i=1}^n, C_1^\star, C_2^\star, C_3^\star) \in \mathbb{F}$.
5. Compute $a(X, \mu) := \sum_{i=1}^n X^{i-1} \mu^{i-1} a_i + \mu^n a^\star \in \mathbb{F}^\ell[X]$.
6. Compute $b(X, \mu) := \sum_{i=1}^n X^{n-i} b_i + \mu b^\star \in \mathbb{F}^\ell[X]$.
7. Compute the product polynomial $a(X, \mu) \circ b(X, \mu)$, which is of the form $\sum_{i=1}^{2n-1} X^{i-1} t_i \in \mathbb{F}^\ell[X]$.
8. For each $i \in [2n-1] \setminus \{n\}$, compute the commitment $C_{t,i} := \mathsf{CM.Commit}(\mathsf{ck}, t_i; 0) \in \mathbb{G}$.
9. Use the random oracle to compute the challenge $\nu := \rho(\mu, [C_{t,i}, C_{t,n+i}]_{i=1}^{n-1}) \in \mathbb{F}$.
10. Compute the commitment to $a(\nu, \mu)$: $C_1 := \sum_{i=1}^n \nu^{i-1} \mu^{i-1} C_{1,i} + \mu^n C_1^\star \in \mathbb{G}$.
11. Compute the commitment to $b(\nu, \mu)$: $C_2 := \sum_{i=1}^n \nu^{n-i} C_{2,i} + \mu C_2^\star \in \mathbb{G}$.
12. Compute the commitment to $a(\nu, \mu) \circ b(\nu, \mu)$:

$$C_3 := \sum_{i=1}^{n-1} \nu^{i-1} C_{t,i} + \nu^{n-1}(\mu^n C_3^\star + \sum_{i=1}^n \mu^{i-1} C_{3,i}) + \sum_{i=1}^{n-1} \nu^{n+i-1} C_{t,n+1} \in \mathbb{G} \ .$$

13. Compute opening value and opening randomness for C_1:

$$a := \sum_{i=1}^n \nu^{i-1} \mu^{i-1} a_i + \mu^n a^\star \in \mathbb{F}^\ell \quad \text{and} \quad \omega_1 := \sum_{i=1}^n \nu^{i-1} \mu^{i-1} \omega_{1,i} + \mu^n \omega_1^\star \in \mathbb{F} \ .$$

14. Compute opening value and opening randomness for C_2:

$$b := \sum_{i=1}^n \nu^{n-i} b_i + \mu b^\star \in \mathbb{F}^\ell \quad \text{and} \quad \omega_2 := \sum_{i=1}^n \nu^{n-i} \omega_{2,i} + \mu \omega_2^\star \in \mathbb{F} \ .$$

15. Compute opening randomness for C_3:

$$\omega_3 := \nu^{n-1}(\mu^n \omega_3^\star + \sum_{i=1}^n \mu^{i-1} \omega_{3,i}) \in \mathbb{F} \ .$$

16. Set the accumulator $\mathsf{acc} := (\mathsf{acc}.\mathbb{x}, \mathsf{acc}.\mathbb{w})$ where $\mathsf{acc}.\mathbb{x} := (C_1, C_2, C_3)$ and $\mathsf{acc}.\mathbb{w} := (a, b, \omega_1, \omega_2, \omega_3)$.
17. Set the accumulation proof $\mathsf{pf} := (C_1^\star, C_2^\star, C_3^\star, [C_{t,i}, C_{t,n+i}]_{i=1}^{n-1})$.
18. Output $(\mathsf{acc}, \mathsf{pf})$.

Accumulation verifier. On input the accumulator verification key avk, predicate instances $[\mathsf{qx}_i]_{i=1}^n$ (of the same form as accumulator instances $[\mathsf{acc}_j.\mathbb{x}]_{j=1}^m$), a new accumulator instance $\mathsf{acc}.\mathbb{x}$, and an accumulation proof pf, V works as below.

$\mathrm{V}^{\rho}(\mathsf{avk} = \ell, [\mathsf{qx}_i]_{i=1}^{n}, \mathsf{acc}.\mathbb{x}, \mathsf{pf})$:
1. Compute $\mu := \rho(\ell, [\mathsf{qx}_i]_{i=1}^{n}, C_1^{\star}, C_2^{\star}, C_2^{\star})$ and $\nu := \rho(\mu, [C_{t,i}, C_{t,n+i}]_{i=1}^{n-1})$.
2. Check that $\mathsf{acc}.\mathbb{x}.C_1 = \sum_{i=1}^{n} \nu^{i-1}\mu^{i-1}\mathsf{qx}_i.C_1 + \mu^n C_1^{\star}$.
3. Check that $\mathsf{acc}.\mathbb{x}.C_2 = \sum_{i=1}^{n} \nu^{n-i}\mathsf{qx}_i.C_2 + \mu C_2^{\star}$.
4. Check that $\mathsf{acc}.\mathbb{x}.C_3 = \sum_{i=1}^{n-1} \nu^{i-1} C_{t,i} + \nu^{n-1}(\mu^n C_3^{\star} + \sum_{i=1}^{n} \mu^{i-1}\mathsf{qx}_i.C_3) + \sum_{i=1}^{n-1} \nu^{n+i-1} C_{t,n+i}$.

Decider. On input the decision key $\mathsf{dk} = \mathsf{ck}$ and an accumulator $\mathsf{acc} = (\mathsf{acc}.\mathbb{x}, \mathsf{acc}.\mathbb{w})$, D performs the checks from the Hadamard product predicate Φ_{HP} on acc (see Equation (3.1)). That is, D checks that $\mathsf{acc}.\mathbb{x}.C_1 = \mathsf{CM.Commit}(\mathsf{ck}, \mathsf{acc}.\mathbb{w}.a; \mathsf{acc}.\mathbb{w}.\omega_1)$, $\mathsf{acc}.\mathbb{x}.C_2 = \mathsf{CM.Commit}(\mathsf{ck}, \mathsf{acc}.\mathbb{w}.b; \mathsf{acc}.\mathbb{w}.\omega_2)$, and $\mathsf{acc}.\mathbb{x}.C_3 = \mathsf{CM.Commit}(\mathsf{ck}, \mathsf{acc}.\mathbb{w}.a \circ \mathsf{acc}.\mathbb{w}.b; \mathsf{acc}.\mathbb{w}.\omega_3)$.

3.7.2 Proof of Theorem 3.7.2

We prove that the accumulation scheme constructed in the previous section satisfies the claimed efficiency properties, achieves completeness, and achieves zero knowledge. Then in Section 3.7.2.1 we prove that it achieves knowledge soundness.

Efficiency. We now analyze the efficiency of our accumulation scheme.

- *Generator:* $\mathrm{G}(1^{\lambda})$ outputs 1^{λ}, and hence runs in time $O(\lambda)$.
- *Indexer:* $\mathrm{I}^{\rho}(\mathsf{pp}, \mathsf{pp}_{\Phi}, \mathsf{i}_{\Phi})$ runs $\mathsf{CM.Trim}$ with message length ℓ.
- *Accumulation prover:* $\mathrm{P}^{\rho}(\mathsf{apk}, [(\mathsf{qx}_i, \mathsf{qw}_i)]_{i=1}^{n})$ performs $O(n) \cdot \ell$ group scalar multiplications and $\tilde{O}(n) \cdot \ell$ field additions/multiplications. (The quasilinear cost in n is due to multiplication of polynomials of degree n.)
- *Accumulation verifier:* $\mathrm{V}^{\rho}(\mathsf{avk}, [\mathsf{qx}_i]_{i=1}^{n}, \mathsf{acc}.\mathbb{x}, \mathsf{pf})$ makes 2 calls to the random oracle, $O(n)$ field operations, and $5n - 5$ group scalar multiplications.
- *Decider:* $\mathrm{D}(\mathsf{dk}, \mathsf{acc})$ invokes the Hadamard product predicate Φ_{HP} and performs 3ℓ scalar multiplications.
- *Sizes:* The accumulator instance $\mathsf{acc}.\mathbb{x}$ consists of 3 group elements. The accumulator witness $\mathsf{acc}.\mathbb{w}$ consists of $2\ell + 3$ field elements. The accumulation proof pf consists of $2n - 2$ group elements.

Completeness. Since we need only accumulate predicate inputs (as accumulators are split like predicate inputs and the decider equals the predicate being accumulated), it suffices to demonstrate that the simplified completeness property from Section 3.4.1 holds. Fix an (unbounded) adversary $\mathcal{A}$. For each $i \in [n]$, since

$$\Phi_{\mathsf{HP}}\big(\mathsf{pp}_{\Phi}, \mathsf{i}_{\Phi} = \ell, \mathsf{qx}_i = (C_{1,i}, C_{2,i}, C_{3,i}), \mathsf{qw}_i = (a_i, b_i, \omega_{1,i}, \omega_{2,i}, \omega_{3,i})\big) = 1 \ ,$$

we know that $C_{1,i} = \mathsf{CM.Commit}(\mathsf{ck}, a_i; \omega_{1,i})$, $C_{2,i} = \mathsf{CM.Commit}(\mathsf{ck}, b_i; \omega_{2,i})$, and $C_{3,i} = \mathsf{CM.Commit}(\mathsf{ck}, a_i \circ b_i; \omega_{3,i})$. This implies for $a := \sum_{i=1}^{n} \nu^{i-1}\mu^{i-1}a_i + \mu^n a^\star$ and $b := \sum_{i=1}^{n} \mu^{n-i}b_i + \mu b^\star$ that $a \circ b = \sum_{i=1}^{2n-1} \nu^{i-1} t_i$ and that $t_n = \mu^n(a^\star \circ b_1 + a_n \circ b^\star) + \sum_{i=1}^{n} \mu^{i-1} a_i \circ b_i$. Further we have that $\omega_1 = \sum_{i=1}^{n} \nu^{i-1}\mu^{i-1}\omega_{1,i} + \mu^n\omega_1^\star$, $\omega_2 = \sum_{i=1}^{n} \mu^{n-i}\omega_{2,i} + \mu\omega_2^\star$, and $\omega_3 = \nu^{n-1}(\mu^n\omega_3^\star + \sum_{i=1}^{n}\mu^{i-1}\omega_{3,i})$. This implies that $C_1 = \mathsf{CM.Commit}(\mathsf{ck}, a; \omega_1)$, $C_2 = \mathsf{CM.Commit}(\mathsf{ck}, b; \omega_2)$, and $C_3 = \mathsf{CM.Commit}(\mathsf{ck}, a \circ b; \omega_3)$; that is, the new accumulator is accepted by the decider. That the accumulation verifier accepts the corresponding instance parts also follows form the above equations, and the homomorphic properties of the Pedersen commitment.

Zero knowledge. Consider the simulator S for AS that works as follows:

$\mathrm{S}^\rho(\tau = \bot, \mathsf{pp}_\Phi = \mathsf{pp}_{\mathsf{CM}}, \mathsf{i}_\Phi = \ell)$:

1. Sample vectors $a, b \in \mathbb{F}^\ell$.
2. Sample opening randomness elements $\omega_1, \omega_2, \omega_3 \in \mathbb{F}$.
3. Compute $C_1 := \mathsf{CM.Commit}(\mathsf{ck}, a; \omega_1)$.
4. Compute $C_2 := \mathsf{CM.Commit}(\mathsf{ck}, b; \omega_2)$.
5. Compute $C_3 := \mathsf{CM.Commit}(\mathsf{ck}, a \circ b; \omega_3)$.
6. Set the accumulator instance $\mathsf{acc}.\mathbb{x} := (C_1, C_2, C_3)$.
7. Set the accumulator witness $\mathsf{acc}.\mathbb{w} := (a, b, \omega_1, \omega_2, \omega_3)$.
8. Output $\mathsf{acc} := (\mathsf{acc}.\mathbb{x}, \mathsf{acc}.\mathbb{w})$.

By construction, the sampled accumulator satisfies the decider. Moreover, the accumulator is distributed identically to an accumulator output by the (honest) accumulation prover. This is because all elements of the accumulator are random within the respective domains subject only to the condition that the decider accepts the accumulator.

3.7.2.1 Knowledge soundness

We need only accumulate predicate inputs (as accumulators are split like predicate inputs and the decider equals the predicate being accumulated), so it suffices to demonstrate that the simplified knowledge soundness property from Section 3.4.1 holds. We describe an extractor and then analyze why it satisfies the property.

Define the following algorithm:

$A^\rho\big((\mathsf{pp}, \mathsf{pp}_\Phi, \mathsf{ai})\big)$:

1. $(\mathsf{i}_\Phi = \ell, [\mathsf{qx}_i]_{i=1}^n, \mathsf{acc}, \mathsf{pf}) \leftarrow \tilde{\mathrm{P}}^\rho(\mathsf{pp}, \mathsf{pp}_\Phi, \mathsf{ai})$.
2. Parse the accumulation proof pf as $(C_1^\star, C_2^\star, C_3^\star, [C_{t,i}, C_{t,n+i}]_{i=1}^{n-1})$.
3. Set the query $\mathsf{q} := (\ell, [\mathsf{qx}_i]_{i=1}^n, C_1^\star, C_2^\star, C_3^\star)$.
4. Set the first output o to be $[C_{t,i}, C_{t,n+i}]_{i=1}^{n-1}$.
5. Set the second output o' to be the accumulator acc.
6. Query the random oracle ρ at q and at $(\rho(\mathsf{q}), \mathsf{o})$.
7. Output $(\mathsf{q}, \mathsf{o}, \mathsf{o}')$.

Define the forking lemma predicate:

$\mathsf{p}\big((\mathsf{pp}, \mathsf{pp}_\Phi, \mathsf{ai}), (\mathsf{q}, \mathfrak{a}), \mathsf{o}, \mathfrak{a}', \mathsf{o}', \mathsf{tr}\big)$:

1. Parse the query q as $(\ell, [\mathsf{qx}_i]_{i=1}^n, C_1^\star, C_2^\star, C_3^\star)$.
2. Parse the first output o as $[C_{t,i}, C_{t,n+i}]_{i=1}^{n-1}$.
3. Parse the second output o' as an accumulator acc.
4. Set the accumulation proof $\mathsf{pf} := (C_1^\star, C_2^\star, C_3^\star, [C_{t,i}, C_{t,n+i}]_{i=1}^{n-1})$.
5. Compute $(\mathsf{apk}, \mathsf{avk}, \mathsf{dk}) := \mathrm{I}(\mathsf{pp}, \mathsf{pp}_\Phi, \ell)$.
6. Check that $\mathfrak{a} \neq 0$.
7. Check that $\mathrm{V}(\mathsf{avk}, [\mathsf{qx}_i]_{i=1}^n, \mathsf{acc}.\mathbb{x}, \mathsf{pf})$ outputs 1 when answering its first random oracle query with $\mathfrak{a}$ and its second random oracle query with $\mathfrak{a}'$.
8. Check that $\mathrm{D}(\mathsf{dk}, \mathsf{acc})$ outputs 1.

For the remainder of the proof we implicitly consider only the case that $\mathrm{V}^\rho(\mathsf{avk}, [\mathsf{qx}_i]_{i=1}^n, \mathsf{acc}.\mathbb{x}, \mathsf{pf}) = 1$ and $\mathrm{D}(\mathsf{dk}, \mathsf{acc}) = 1$ for $(\mathsf{i}_\Phi, [\mathsf{qx}_i]_{i=1}^n, \mathsf{acc}, \mathsf{pf}) \leftarrow \tilde{\mathrm{P}}^\rho(\mathsf{pp}, \mathsf{pp}_\Phi, \mathsf{ai})$ and $(\mathsf{apk}, \mathsf{avk}, \mathsf{dk}) := \mathrm{I}(\mathsf{pp}, \mathsf{pp}_\Phi, \ell)$; otherwise, the implication holds vacuously. In this case the output of A satisfies p with probability $1 - \mathrm{negl}(\lambda)$, where the negligible loss accounts for the case that $\rho(\mathsf{q}) = 0$. Let Fork_2 be the algorithm given by applying Corollary 1 to the forking lemma predicate p.

$\mathrm{E}^{\tilde{\mathrm{P}}, \rho}(\mathsf{pp}, \mathsf{pp}_\Phi, \mathsf{ai}, r)$:

1. Run $(\mathsf{q}, \mathsf{o}, \mathsf{o}'; \mathsf{tr}) \leftarrow A^\rho\big((\mathsf{pp}, \mathsf{pp}_\Phi, \mathsf{ai}); r\big)$.
2. Parse q as $(\ell, [\mathsf{qx}_i]_{i=1}^n, C_1^\star, C_2^\star, C_3^\star)$, o as $[C_{t,i}, C_{t,n+i}]_{i=1}^{n-1}$, and o' as acc.
3. Set the accumulation proof $\mathsf{pf} := (C_1^\star, C_2^\star, C_3^\star, [C_{t,i}, C_{t,n+i}]_{i=1}^{n-1})$.
4. Run $[\mu_j, \mathsf{o}_j, [\nu_{j,k}, \mathsf{o}'_{j,k}]_{k=1}^{2n-1}]_{j=1}^{n+1} \leftarrow \mathsf{Fork}_2^A(\mathsf{pp}, 1^{n+1}, 1^{2n-1}, (\mathsf{q}, \rho(\mathsf{q})), \mathsf{o}, \rho(\rho(\mathsf{q}), \mathsf{o}), \mathsf{o}', \mathsf{tr}, r)$.
5. For each $j \in [n+1]$ and for each $k \in [2n-1]$:
   parse $\mathsf{o}'_{j,k}$ as $\mathsf{acc}^{(j,k)} = \big((C_{1,\star}^{(j,k)}, C_{2,\star}^{(j,k)}, C_{3,\star}^{(j,k)}), (a_\star^{(j,k)}, b_\star^{(j,k)}, \omega_{1,\star}^{(j,k)}, \omega_{2,\star}^{(j,k)}, \omega_{3,\star}^{(j,k)})\big)$.
6. Set U_j to be the Vandermonde matrix on $(\mu_j\nu_{1,1}, \ldots, \mu_j\nu_{1,n})$.
7. Set V_j to be the *descending* Vandermonde matrix on $(\nu_{j,1}, \ldots, \nu_{j,n})$: $V_j := \begin{pmatrix} \nu_{j,1}^{n-1} & \nu_{j,1}^{n-2} & \cdots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ \nu_{j,n}^{n-1} & \nu_{j,n}^{n-2} & \cdots & 1 \end{pmatrix}$.
8. If U_1, U_2, V_1, V_2 are not invertible, abort. Otherwise, compute

$$\begin{pmatrix} \bar{a}_1 & \bar{\omega}_{1,1} \\ a_2 & \omega_{1,2} \\ \vdots & \vdots \\ a_n & \omega_{1,n} \end{pmatrix} := U_1^{-1} \begin{pmatrix} a_\star^{(1,1)} & \omega_{1,\star}^{(1,1)} \\ \vdots & \vdots \\ a_\star^{(1,n)} & \omega_{1,\star}^{(1,n)} \end{pmatrix} \qquad \begin{pmatrix} b_1 & \omega_{2,1} \\ \vdots & \vdots \\ b_{n-1} & \omega_{2,n-1} \\ \bar{b}_n & \bar{\omega}_{2,n} \end{pmatrix} := V_1^{-1} \begin{pmatrix} b_\star^{(1,1)} & \omega_{2,\star}^{(1,1)} \\ \vdots & \vdots \\ b_\star^{(1,n)} & \omega_{2,\star}^{(1,n)} \end{pmatrix}$$

$$\begin{pmatrix} \bar{a}'_1 & \bar{\omega}'_{1,1} \\ a'_2 & \omega'_{1,2} \\ \vdots & \vdots \\ a'_n & \omega'_{1,n} \end{pmatrix} := U_2^{-1} \begin{pmatrix} a_\star^{(2,1)} & \omega_{1,\star}^{(2,1)} \\ \vdots & \vdots \\ a_\star^{(2,n)} & \omega_{1,\star}^{(2,n)} \end{pmatrix} \qquad \begin{pmatrix} b'_1 & \omega'_{2,1} \\ \vdots & \vdots \\ b'_{n-1} & \omega'_{2,n-1} \\ \bar{b}'_n & \bar{\omega}'_{2,n} \end{pmatrix} := V_2^{-1} \begin{pmatrix} b_\star^{(2,1)} & \omega_{2,\star}^{(2,1)} \\ \vdots & \vdots \\ b_\star^{(2,n)} & \omega_{2,\star}^{(2,n)} \end{pmatrix}$$

9. Compute

$$a_1 := \frac{\mu_2^n \bar{a}_1 - \mu_1^n \bar{a}'_1}{\mu_2^n - \mu_1^n} \qquad \omega_{1,1} := \frac{\mu_2^n \bar{\omega}_{1,1} - \mu_1^n \bar{\omega}'_{1,1}}{\mu_2^n - \mu_1^n}$$
$$b_n := \frac{\mu_2 \bar{b}_n - \mu_1 \bar{b}'_n}{\mu_2 - \mu_1} \qquad \omega_{2,n} := \frac{\mu_2 \bar{\omega}_{2,n} - \mu_1 \bar{\omega}'_{2,n}}{\mu_2 - \mu_1}$$

10. For each $j \in [n+1]$:
    a) Set P_j to be the Vandermonde matrix on $(\nu_{j,1}, \ldots, \nu_{j,2n-1})$.
    b) If P_j is not invertible, abort. Otherwise, compute

$$\begin{pmatrix} \tau_1^{(j)} \\ \vdots \\ \tau_{2n-1}^{(j)} \end{pmatrix} := P_j^{-1} \begin{pmatrix} \omega_{3,\star}^{(j,1)} \\ \vdots \\ \omega_{3,\star}^{(j,2n-1)} \end{pmatrix} .$$

11. Set M to be the Vandermonde matrix on $(\mu_1, \ldots, \mu_{n+1})$.
12. If M is not invertible, abort. Otherwise, compute

$$\begin{pmatrix} \omega_{3,1} \\ \vdots \\ \omega_{3,n+1} \end{pmatrix} := M^{-1} \begin{pmatrix} \tau_n^{(1)} \\ \vdots \\ \tau_n^{(n+1)} \end{pmatrix} .$$

13. For each $i \in [n]$, set $\mathsf{qw}_i := (a_i, b_i, \omega_{1,i}, \omega_{2,i}, \omega_{3,i})$.
14. Output $\big(\mathsf{i}_\Phi, [(\mathsf{qx}_i, \mathsf{qw}_i)]_{i=1}^n, \mathsf{acc}, \mathsf{pf}\big)$.

By the properties of Fork_2 guaranteed in Corollary 1, $\mathrm{E}_{\tilde{\mathsf{P}}}$ runs in expected polynomial time and, moreover, except with probability $\mathrm{negl}(\lambda)$ the following event E occurs:

> $\{\mu_j\}_{j\in[n+1]}$ are pairwise distinct
> and $\forall\, j \in [n+1]$ it holds that $\{\nu_{j,k}\}_{k\in[2n-1]}$ are pairwise distinct
> and $\forall\, j \in [n+1]\, \forall\, k \in [2n-1]$ it holds that $\mathsf{p}\big((\mathsf{pp}, \mathsf{pp}_\Phi, \mathsf{ai}), (\mathsf{q}, \mu_j), \mathsf{o}_j, \nu_{j,k}, \mathsf{o}'_{j,k}, \mathsf{tr}_\mathsf{q}\big) = 1$.

Conditioned on E, since the challenges are all distinct, the Vandermonde matrices $\{U_j\}_{j=1,2}$, $\{V_j\}_{j=1,2}$, $\{P_j\}_{j=1,\ldots,n+1}$, and M are all invertible, and so the extractor does not abort. (Note that for U_j to be invertible, we need that $\mu_j \neq 0$, which we guarantee by the definition of p.)

The claim below completes the proof, because it is immediate from that claim and the above discussion that with all but negligible probability, for all $i \in [n]$,

$$\Phi_{\mathsf{HP}}\big(\mathsf{pp}_\Phi, \mathsf{i}_\Phi = \ell, \mathsf{qx}_i = (C_{1,i}, C_{2,i}, C_{3,i}), \mathsf{qw}_i = (a_i, b_i, \omega_{1,i}, \omega_{2,i}, \omega_{3,i})\big) = 1 \ .$$

Claim 3.7.3. *The event E implies that for every $i \in [n]$ it holds that:*

$$C_{1,i} = \mathsf{CM.Commit}(\mathsf{ck}, a_i; \omega_{1,i}) \ ,$$
$$C_{2,i} = \mathsf{CM.Commit}(\mathsf{ck}, b_i; \omega_{2,i}) \ ,$$
$$C_{3,i} = \mathsf{CM.Commit}(\mathsf{ck}, a_i \circ b_i; \omega_{3,i}) \ .$$

Proof. Define the following vectors:

$$\begin{aligned}
\forall j \in [n],\ \vec{C}_1^{(j)} &:= (C_{1,1} + \mu_j^n C_1^\star, C_{1,2}, \ldots, C_{1,n}) & \forall j \in [n],\ \vec{C}_{1,\star}^{(j)} &:= (C_{1,\star}^{(j,1)}, \ldots, C_{1,\star}^{(j,n)}) \\
\forall j \in [n],\ \vec{C}_2^{(j)} &:= (C_{2,1}, \ldots, C_{2,n-1}, C_{2,n} + \mu_j C_2^\star) & \forall j \in [n],\ \vec{C}_{2,\star}^{(j)} &:= (C_{2,\star}^{(j,1)}, \ldots, C_{2,\star}^{(j,n)}) \\
\vec{C}_3 &:= (C_{3,1}, \ldots, C_{3,n}, C_3^\star) & \forall j \in [n],\ \vec{C}_{3,\star}^{(j)} &:= (C_{3,\star}^{(j,1)}, \ldots, C_{3,\star}^{(j,2n-1)})
\end{aligned}$$

For each $j \in [n]$, define the following vector

$$\vec{C}_t^{(j)} := \left(C_{t,1}^{(j)}, \ldots C_{t,n-1}^{(j)}, \mu_j^n C_3^\star + \textstyle\sum_{i=1}^n \mu_j^{i-1} C_{3,i}, C_{t,n+1}^{(j)}, \ldots, C_{t,2n-1}^{(j)}\right) \ .$$

Fix $j \in [n]$ and $k \in [2n-1]$. Since the accumulation verifier accepts $(\mathsf{avk}, [\mathsf{qx}_i]_{i=1}^n, \mathsf{acc}^{(j,k)}, \mathsf{pf})$, we have

$$\vec{C}_{1,\star} = U_1 \cdot \vec{C}_1^{(1)}\ ,\ \vec{C}_{2,\star} = V_1 \cdot \vec{C}_2^{(1)}\ ,\ \vec{C}_{3,\star}^{(j)} = P_j \cdot \vec{C}_t^{(j)}\ ,\ \vec{C}_{1,\star} = U_2 \cdot \vec{C}_1^{(2)}\ ,\ \vec{C}_{2,\star} = V_2 \cdot \vec{C}_2^{(2)}\ .$$

Moreover, since the decider accepts $(\mathsf{dk}, \mathsf{acc}^{(j,k)})$, it holds that

$$\begin{aligned}
C_{1,\star}^{(j,k)} &= \mathsf{CM.Commit}(\mathsf{ck}, a_\star^{(j,k)}; \omega_{1,\star}^{(j,k)})\ , \\
C_{2,\star}^{(j,k)} &= \mathsf{CM.Commit}(\mathsf{ck}, b_\star^{(j,k)}; \omega_{2,\star}^{(j,k)})\ , \\
C_{3,\star}^{(j,k)} &= \mathsf{CM.Commit}(\mathsf{ck}, a_\star^{(j,k)} \circ b_\star^{(j,k)}; \omega_{3,\star}^{(j,k)})\ .
\end{aligned}$$

Using the homomorphic property of $\mathsf{CM.Commit}$, and because $\vec{C}_1 = U_1^{-1}\vec{C}_{1,\star}$, it holds for all $i \in \{2, \ldots, n\}$ that

$$\begin{aligned}
C_{1,i} &= \textstyle\sum_{k=1}^n U_1^{-1}[i,k] C_{1,\star}^{(1,k)} \\
&= \textstyle\sum_{k=1}^n U_1^{-1}[i,k] \mathsf{CM.Commit}(\mathsf{ck}, a_\star^{(j,k)}; \omega_{1,\star}^{(j,k)}) \\
&= \mathsf{CM.Commit}(\mathsf{ck}, \textstyle\sum_{k=1}^n U_1^{-1}[i,k] a_\star^{(1,k)}; \sum_{k=1}^n U_1^{-1}[i,k] \omega_{1,\star}^{(1,k)}) \\
&= \mathsf{CM.Commit}(\mathsf{ck}, a_i; \omega_{1,i})\ .
\end{aligned}$$

Similarly, since $\vec{C}_2 = V_1^{-1}\vec{C}_{2,\star}$, it holds that for all $i \in \{1, \ldots, n-1\}$, $C_{2,i} = \mathsf{CM.Commit}(\mathsf{ck}, b_i; \omega_{2,i})$. Furthermore,

$$\begin{aligned}
C_{1,1} + \mu_1^n C_1^\star &= \mathsf{CM.Commit}(\mathsf{ck}, \bar{a}_1; \bar{\omega}_{1,1})\ , & C_{1,1} + \mu_2^n C_1^\star &= \mathsf{CM.Commit}(\mathsf{ck}, \bar{a}_1'; \bar{\omega}_{1,1}')\ , \\
C_{2,n} + \mu_1 C_2^\star &= \mathsf{CM.Commit}(\mathsf{ck}, \bar{b}_n; \bar{\omega}_{2,n})\ , & C_{2,n} + \mu_2 C_2^\star &= \mathsf{CM.Commit}(\mathsf{ck}, \bar{b}_2'; \bar{\omega}_{2,n}')\ .
\end{aligned}$$

From this we can see that if $\mu_1 \neq \mu_2$ (which is implied by E) then $C_{1,1} = \mathsf{CM.Commit}(\mathsf{ck}, a_1; \omega_{1,1})$ and $C_{2,n} = \mathsf{CM.Commit}(\mathsf{ck}, b_n; \omega_{2,n})$. Define $a^\star := \frac{\bar{a}_1 - \bar{a}_1'}{\mu_1^n - \mu_2^n}$, $\omega_1^\star := \frac{\bar{\omega}_{1,1} - \bar{\omega}_{1,1}'}{\mu_1^n - \mu_2^n}$ and $b^\star := \frac{\bar{b}_n - \bar{b}_n'}{\mu_1 - \mu_2}$, $\omega_2^\star := \frac{\bar{\omega}_{2,n} - \bar{\omega}_{2,n}'}{\mu_1 - \mu_2}$. By the homomorphic property of the commitment scheme we have that for all $j \in [n]$

$$\begin{aligned}
C_{1,1} + \mu_j^n C_1^\star &= \mathsf{CM.Commit}(\mathsf{ck}, a_1 + \mu_j^n a^\star; \omega_{1,1} + \mu_j^n \omega_1^\star)\ , \\
C_{2,n} + \mu_j C_2^\star &= \mathsf{CM.Commit}(\mathsf{ck}, b_n + \mu_j b^\star; \omega_{2,n} + \mu_j \omega_2^\star)\ .
\end{aligned}$$

Fix $j \in [n]$. Recall that U_j is the Vandermonde matrix on $(\mu_j\nu_{j,1}, \ldots, \mu_j\nu_{j,n})$, and that V_j is the *descending* Vandermonde matrix on $(\nu_{j,1}, \ldots, \nu_{j,n})$. Observe that since $\vec{C}^{(j)}_{1,\star} = U_j \cdot \vec{C}^{(j)}_1$ and $\vec{C}^{(j)}_{2,\star} = V_j \cdot \vec{C}^{(j)}_2$, $C^{(j,k)}_{3,\star} = \mathsf{CM.Commit}(\mathsf{ck}, (a^\star\mu_j^n + \sum_{i=1}^n a_i\mu_j^{i-1}\nu_{j,k}^{i-1}) \circ (b^\star\mu_j + \sum_{i=1}^n b_i\nu_{j,k}^{n-i}); \omega^{(j,k)}_{3,\star})$. For $i \in [2n-1]$, let $t_i^{(j)}$ be the coefficient of X^{i-1} in the polynomial $z_j(X) := (a^\star\mu_j^n + \sum_{i=1}^n a_i\mu_j^{i-1}X^{i-1}) \circ (b^\star\mu_j + \sum_{i=1}^n b_iX^{n-i})$, so that $C^{(j,k)}_{3,\star} = \mathsf{CM.Commit}(\mathsf{ck}, \sum_{i=1}^{2n-1} t_i^{(j)}\nu_{j,k}^{i-1}; \omega^{(j,k)}_{3,\star})$. Recall that $t_n^{(j)} = \mu^n \cdot (a^\star \circ b_1 + a_n \circ b^\star) + \sum_{i=1}^n \mu_j^{i-1} a_i \circ b_i$.

Next, for each $j \in [n]$, since $\vec{C}^{(j)}_t = P_j^{-1}\vec{C}^{(j)}_{3,\star}$, it follows that for $i \in [2n-1] \setminus \{n\}$, $C^{(j)}_{t,i} = \mathsf{CM.Commit}(\mathsf{ck}, t_i^{(j)}; \tau_i^{(j)})$ and that $C^{(j)}_{t,n} := \mu_j^n C_3^\star + \sum_{i=1}^n \mu_j^{i-1} C_{3,i} = \mathsf{CM.Commit}(\mathsf{ck}, t_n^{(j)}; \tau_n^{(j)})$. Letting $\vec{C}_\circ := (C^{(1)}_{t,n}, \ldots, C^{(n+1)}_{t,n})$ we see that $\vec{C}_\circ = M \cdot \vec{C}_3$, so that for all $i \in [n]$, $C_{3,i} = \mathsf{CM.Commit}(\mathsf{ck}, a_i \circ b_i; \omega_{3,i})$. Note that the $(n+1)$-th entry of $\vec{C}_3$ is $C_3^\star$, which commits to $a^\star \circ b_1 + a_n \circ b^\star$. □

3.8 Split accumulation for R1CS

In Section 3.8.1 we describe a zkNARK for R1CS and then in Section 3.8.2 we describe a split accumulation scheme for it; security proofs are in Section 3.8.3.

3.8.1 zkNARK for R1CS

We describe a zkNARK for R1CS (see Definition 3.8.1) in the ROM; the protocol is the result of applying the Fiat–Shamir transformation to an underlying sigma protocol for R1CS based on Pedersen commitments. Following the definition of a non-interactive argument in the ROM from Section 3.3.1, we describe the generator $\mathcal{G}$, indexer $\mathcal{I}$, prover $\mathcal{P}$, and verifier $\mathcal{V}$.

Definition 3.8.1. *The indexed relation $\mathcal{R}_{\mathrm{R1CS}}(\mathbb{F})$ is the set of all triples $(\mathbb{i}, \mathbb{x}, \mathbb{w})$ where $\mathbb{i} = (A, B, C, \mathsf{n})$ is a triple of three coefficient matrices in $\mathbb{F}^{\mathsf{M}\times\mathsf{N}}$ and an instance size $\mathsf{n} \in \mathbb{N}$, $\mathbb{x} = x \in \mathbb{F}^{\mathsf{n}}$ is an R1CS input, and $\mathbb{w} = w \in \mathbb{F}^{\mathsf{N}-\mathsf{n}}$ is an R1CS witness such that $Az \circ Bz = Cz$ for $z := (x, w)$.*

Generator. The generator $\mathcal{G}$ has query access to a random oracle ρ_{NARK} (but happens not to use it here) and receives as input the security parameter λ in unary and works as follows. Sample the description of a prime-order group $(\mathbb{G}, q, G) \leftarrow \mathsf{SampleGrp}(1^\lambda)$; here q is the prime order of the group and G is a generator for the group; henceforth we denote by $\mathbb{F}$ the field of prime order q. Output the public parameters $\mathsf{pp} := (\mathbb{G}, q, G)$.

Indexer. The indexer $\mathcal{I}$ has query access to a random oracle ρ_{NARK}, receives as input public parameters pp and an index $\mathbb{i} = (A, B, C, \mathsf{n})$, and works as follows. Use the random oracle to hash the coefficient matrices: $\tau := \rho_{\mathsf{NARK}}(A, B, C, \mathsf{n})$. Letting M be the number of rows in a coefficient matrix, use the random oracle ρ_{NARK} to sample group generators to form a commitment key $\mathsf{ck} := (G_1, \ldots, G_{\mathsf{M}}, H) \in \mathbb{G}^{\mathsf{M}+1}$ for the Pedersen commitment with messages in $\mathbb{F}^{\mathsf{M}}$ (the extra group element H is used for hiding). Output the index proving key $\mathrm{ipk} := (\mathsf{ck}, A, B, C, \mathsf{n}, \tau)$ and index verification key $\mathrm{ivk} := \mathrm{ipk}$. (Here, unlike in the split accumulation scheme in Section 3.8.2, the indexer can be folded into the prover and verifier as the verifier runs in linear time.)

Prover. The prover $\mathcal{P}$ has query access to a random oracle ρ_{NARK}, receives as input the index proving key $\mathrm{ipk} = (\mathsf{ck}, A, B, C, \mathsf{n}, \tau)$, an instance $\mathbb{x} = x \in \mathbb{F}^{\mathsf{n}}$, and a witness $\mathbb{w} = w \in \mathbb{F}^{\mathsf{N}-\mathsf{n}}$, and works as follows.

1. Assemble the full assignment $z := (x, w) \in \mathbb{F}^{\mathsf{N}}$.
2. Sample randomness $r \in \mathbb{F}^{\mathsf{N}-\mathsf{n}}$ that will be used to blind the witness w.
3. Compute linear combinations of the full assignment z and (padded) randomness r (they are in $\mathbb{F}^{\mathsf{M}}$):

$$z_A := Az\,, \qquad z_B := Bz\,, \qquad z_C := Cz\,,$$
$$r_A := A\begin{bmatrix}0^{\mathsf{n}}\\ r\end{bmatrix}\,, \qquad r_B := B\begin{bmatrix}0^{\mathsf{n}}\\ r\end{bmatrix}\,, \qquad r_C := C\begin{bmatrix}0^{\mathsf{n}}\\ r\end{bmatrix}\,.$$

4. Commit to all the linear combinations: sample $\omega_A, \omega_B, \omega_C, \omega'_A, \omega'_B, \omega'_C \in \mathbb{F}$ and compute

$$C_A := \mathsf{CM.Commit}(\mathsf{ck}, z_A; \omega_A)\,, \quad C_B := \mathsf{CM.Commit}(\mathsf{ck}, z_B; \omega_B)\,, \quad C_C := \mathsf{CM.Commit}(\mathsf{ck}, z_C; \omega_C)\,,$$
$$C'_A := \mathsf{CM.Commit}(\mathsf{ck}, r_A; \omega'_A)\,, \quad C'_B := \mathsf{CM.Commit}(\mathsf{ck}, r_B; \omega'_B)\,, \quad C'_C := \mathsf{CM.Commit}(\mathsf{ck}, r_C; \omega'_C)\,.$$

5. Commit to cross terms: sample $\omega_1, \omega_2 \in \mathbb{F}$ and compute

$$C_1 := \mathsf{CM.Commit}(\mathsf{ck}, z_A \circ r_B + z_B \circ r_A; \omega_1) \quad \text{and} \quad C_2 := \mathsf{CM.Commit}(\mathsf{ck}, r_A \circ r_B; \omega_2)\ .$$

6. Set $\pi_1 := (C_A, C_B, C_C, C'_A, C'_B, C'_C, C_1, C_2)$ as the sigma protocol's prover commitment.
7. Use the random oracle to compute the sigma protocol's challenge $\gamma := \rho_{\mathsf{NARK}}(\tau, x, \pi_1) \in \mathbb{F}$.
8. Blind the witness by computing $s := w + \gamma r \in \mathbb{F}^{\mathsf{N}-\mathsf{n}}$.
9. Blind the randomness for linear combinations: $\sigma_A := \omega_A + \gamma\omega'_A$, $\sigma_B := \omega_B + \gamma\omega'_B$, $\sigma_C := \omega_C + \gamma\omega'_C$.
10. Blind the randomness for cross terms: $\sigma_\circ := \omega_C + \gamma\omega_1 + \gamma^2\omega_2$.
11. Set $\pi_2 := (s, \sigma_A, \sigma_B, \sigma_C, \sigma_\circ)$ as the sigma protocol's prover response.
12. Output the proof string $\pi := (\pi_1, \pi_2)$.

Verifier. The prover $\mathcal{V}$ has query access to a random oracle ρ_{NARK}, receives as input the index verification key $\mathrm{ivk} = (\mathsf{ck}, A, B, C, \mathsf{n}, \tau)$ and an instance $\mathbb{x} = x \in \mathbb{F}^{\mathsf{n}}$, and works as follows.

1. Parse the proof π as a pair (π_1, π_2) consisting of a sigma protocol commitment and response.
2. Use the random oracle to compute the sigma protocol's challenge $\gamma := \rho_{\mathsf{NARK}}(\tau, x, \pi_1) \in \mathbb{F}$.
3. Compute linear combinations of the shifted assignment (they are in $\mathbb{F}^{\mathsf{M}}$):

$$s_A := A \begin{bmatrix} x \\ s \end{bmatrix}\ ,\ s_B := B \begin{bmatrix} x \\ s \end{bmatrix}\ ,\ s_C := C \begin{bmatrix} x \\ s \end{bmatrix}\ .$$

4. Check consistency of the linear combinations with the commitments:

$$C_A + \gamma C'_A = \mathsf{CM.Commit}(\mathsf{ck}, s_A; \sigma_A)\,,$$
$$C_B + \gamma C'_B = \mathsf{CM.Commit}(\mathsf{ck}, s_B; \sigma_B)\,,$$
$$C_C + \gamma C'_C = \mathsf{CM.Commit}(\mathsf{ck}, s_C; \sigma_C)\,.$$

5. Check consistency of the Hadamard product with the commitment:

$$C_C + \gamma C_1 + \gamma^2 C_2 = \mathsf{CM.Commit}(\mathsf{ck}, s_A \circ s_B; \sigma_\circ)\ .$$

3.8.2 Split accumulation for the zkNARK verifier

We describe a split accumulation scheme $\mathsf{AS} = (\mathrm{G}, \mathrm{I}, \mathrm{P}, \mathrm{V}, \mathrm{D})$ for the zkNARK for R1CS in Section 3.8.1. As a subroutine we use an accumulation scheme $\mathsf{AS}_{\mathsf{HP}} = (\mathrm{G}_{\mathsf{HP}}, \mathrm{I}_{\mathsf{HP}}, \mathrm{P}_{\mathsf{HP}}, \mathrm{V}_{\mathsf{HP}}, \mathrm{D}_{\mathsf{HP}})$ for the Hadamard product predicate Φ_{HP} (e.g., the one we construct in Section 3.7). We use domain

separation on the given random oracle ρ for different tasks: we use ρ_{HP} to denote the oracle used for one invocation of $\mathsf{AS}_{\mathsf{HP}}$; ρ_{NARK} to denote the oracle used to run the zkNARK for R1CS; and ρ_{AS} to denote the random oracle used by AS for other tasks. We use red text to denote features required to achieve zero knowledge accumulation, provided that $\mathsf{AS}_{\mathsf{HP}}$ is itself a zero knowledge accumulation scheme. (Dropping the red text leads to secure, but not zero knowledge, accumulation.)

Predicate inputs. Following Definition 3.5.1, the predicate to accumulate is the NARK verifier, with the following split in a predicate input q obtained from an R1CS instance x and proof $\pi = (\pi_1, \pi_2)$:

- The instance part of q consists of the R1CS input x and the sigma protocol's commitment π_1. This amounts to 8 group elements and n field elements (which is short).
- The witness part of q consists of the sigma protocol's response π_2. This amounts to $\mathsf{N} - \mathsf{n} + 4$ field elements (which is proportional to the number of rows of the R1CS matrices).

Accumulator. The format of an accumulator acc is as follows:

- The instance part of acc consists of $\mathsf{acc}.\mathbb{x} = (C_x, C_A, C_B, C_C, \mathsf{acc}_{\mathsf{HP}}.\mathbb{x})$.
- The witness part of acc consists of $\mathsf{acc}.\mathbb{w} = (x, s, \sigma_A, \sigma_B, \sigma_C, \mathsf{acc}_{\mathsf{HP}}.\mathbb{w})$.

Note that a split accumulator has a different format to a predicate input. The size of $\mathsf{acc}.\mathbb{x}$ does not depend on the size of the public input x, as required by our PCD construction (Theorem 3.5.3).

Generator. The generator G runs G_{HP} as a subroutine and outputs its output $\mathsf{pp} := \mathsf{pp}_{\mathsf{HP}}$.

Indexer. The indexer I receives as input accumulation public parameters $\mathsf{pp} = \mathsf{pp}_{\mathsf{HP}}$ (output by G), predicate public parameters $\mathsf{pp}_\Phi = \mathsf{pp}_{\mathsf{NARK}}$ (the public parameters of the NARK per Definition 3.5.1), predicate index $\mathsf{i}_\Phi = (A, B, C, \mathsf{n})$ (the index of the relation verified by the NARK per Definition 3.5.1), and works as follows:

- Invoke the NARK indexer $(\mathrm{ipk}, \mathrm{ivk}) := \mathcal{I}^{\rho_{\mathsf{NARK}}}_{\mathsf{NARK}}(\mathsf{pp}_{\mathsf{NARK}}, \mathsf{i}_\Phi)$, and then obtain ck and τ from ipk.
- Set the vector length to be $\ell := \mathsf{M}$, the number of rows in each R1CS coefficient matrix.
- Invoke $\mathrm{I}_{\mathsf{HP}}(\mathsf{pp}_{\mathsf{HP}}, \mathsf{ck}, \ell)$ to obtain $(\mathsf{apk}_{\mathsf{HP}}, \mathsf{avk}_{\mathsf{HP}}, \mathsf{dk}_{\mathsf{HP}})$. (Here we provide ck in place of $\mathsf{pp}_{\Phi_{\mathsf{HP}}}$, making use of the fact that for the Pedersen commitment, these have the same form.)
- Output $(\mathsf{apk}, \mathsf{avk}, \mathsf{dk}) := \big((A, B, C, \mathsf{n}, \tau, \mathsf{apk}_{\mathsf{HP}}), (\tau, \mathsf{n}, \mathsf{avk}_{\mathsf{HP}}), (A, B, C, \mathsf{n}, \mathsf{ck}, \mathsf{dk}_{\mathsf{HP}})\big)$.

Accumulation prover. On input the accumulation proving key $\mathsf{apk} = (A, B, C, \mathsf{n}, \tau, \mathsf{apk}_{\mathsf{HP}})$, predicate instance-witness pairs $[(\mathsf{qx}_i, \mathsf{qw}_i)]_{i=1}^n$, and old accumulators $[\mathsf{acc}_j]_{j=1}^m = [(\mathsf{acc}_j.\mathbb{x}, \mathsf{acc}_j.\mathbb{w})]_{j=1}^m$, P works as below.

1. For each $i \in [n]$:
   a) Compute the challenge of the i-th proof: $\gamma_i := \rho_{\mathsf{NARK}}(\tau, \mathsf{qx}_i)$.
   b) Set $\mathsf{qx}_{\mathsf{HP},i} := (\mathsf{qx}_i.C_A + \gamma_i \cdot \mathsf{qx}_i.C'_A, \mathsf{qx}_i.C_B + \gamma_i \cdot \mathsf{qx}_i.C'_B, \mathsf{qx}_i.C_C + \gamma_i \cdot \mathsf{qx}_i.C_1 + \gamma_i^2 \cdot \mathsf{qx}_i.C_2)$.
   c) Set $\mathsf{qw}_{\mathsf{HP},i} := \big(A \cdot (\mathsf{qx}_i.x, \mathsf{qw}_i.s), B \cdot (\mathsf{qx}_i.x, \mathsf{qw}_i.s), \mathsf{qw}_i.\sigma_A, \mathsf{qw}_i.\sigma_B, \mathsf{qw}_i.\sigma_\circ\big)$.
2. For each $j \in [m]$:
   a) Set $\mathsf{acc}_{\mathsf{HP},j}.\mathbb{x} := \mathsf{acc}_j.\mathbb{x}.\mathsf{acc}_{\mathsf{HP}}.\mathbb{x}$.
   b) Set $\mathsf{acc}_{\mathsf{HP},j}.\mathbb{w} := \mathsf{acc}_j.\mathbb{w}.\mathsf{acc}_{\mathsf{HP}}.\mathbb{w}$.

3. Accumulate Hadamard products:

$$(\mathsf{acc}_{\mathsf{HP}}, \mathsf{pf}_{\mathsf{HP}}) := \mathsf{AS}_{\mathsf{HP}}.\mathrm{P}^{\rho_{\mathsf{HP}}}(\mathsf{apk}_{\mathsf{HP}}, [(\mathsf{qx}_{\mathsf{HP},i}, \mathsf{qw}_{\mathsf{HP},i})]_{i=1}^{n}, [(\mathsf{acc}_{\mathsf{HP},j}.\mathbb{x}, \mathsf{acc}_{\mathsf{HP},j}.\mathbb{w})]_{j=1}^{m}) \ .$$

4. Sample randomness $x^\star \in \mathbb{F}^{\mathsf{n}}$, $s^\star \in \mathbb{F}^{\mathsf{N}-\mathsf{n}}$, and $\omega_A^\star, \omega_B^\star, \omega_C^\star \in \mathbb{F}$ and compute the following commitments:

$$\begin{aligned}
C_x^\star &:= \mathsf{CM.Commit}(\mathsf{ck}, x^\star; 0) \ , \\
C_A^\star &:= \mathsf{CM.Commit}\left(\mathsf{ck}, A \cdot \begin{bmatrix} x^\star \\ s^\star \end{bmatrix}; \omega_A^\star\right) \ , \\
C_B^\star &:= \mathsf{CM.Commit}\left(\mathsf{ck}, B \cdot \begin{bmatrix} x^\star \\ s^\star \end{bmatrix}; \omega_B^\star\right) \ , \\
C_C^\star &:= \mathsf{CM.Commit}\left(\mathsf{ck}, C \cdot \begin{bmatrix} x^\star \\ s^\star \end{bmatrix}; \omega_C^\star\right) \ .
\end{aligned}$$

5. Use the random oracle to compute $\beta := \rho_{\mathsf{AS}}(\tau, [\mathsf{acc}_j.\mathbb{x}]_{j=1}^m, [\mathsf{qx}_i]_{i=1}^n, C_x^\star, C_A^\star, C_B^\star, C_C^\star) \in \mathbb{F}$.
6. Compute the accumulator instance $\mathsf{acc}.\mathbb{x} := (C_x, C_A, C_B, C_C, \mathsf{acc}_{\mathsf{HP}}.\mathbb{x})$ where:

$$\begin{aligned}
C_x &:= \textstyle\sum_{j=1}^m \beta^{j-1} \cdot \mathsf{acc}_j.\mathbb{x}.C_x + \sum_{i=1}^n \beta^{m+i-1} \cdot \mathsf{CM.Commit}(\mathsf{ck}, \mathsf{qx}_i.x; 0) + \beta^{m+n} \cdot C_x^\star \ , \\
C_A &:= \textstyle\sum_{j=1}^m \beta^{j-1} \cdot \mathsf{acc}_j.\mathbb{x}.C_A + \sum_{i=1}^n \beta^{m+i-1} \cdot \left(\mathsf{qx}_i.C_A + \gamma_i \cdot \mathsf{qx}_i.C_A'\right) + \beta^{m+n} \cdot C_A^\star \ , \\
C_B &:= \textstyle\sum_{j=1}^m \beta^{j-1} \cdot \mathsf{acc}_j.\mathbb{x}.C_B + \sum_{i=1}^n \beta^{m+i-1} \cdot \left(\mathsf{qx}_i.C_B + \gamma_i \cdot \mathsf{qx}_i.C_B'\right) + \beta^{m+n} \cdot C_B^\star \ , \\
C_C &:= \textstyle\sum_{j=1}^m \beta^{j-1} \cdot \mathsf{acc}_j.\mathbb{x}.C_C + \sum_{i=1}^n \beta^{m+i-1} \cdot \left(\mathsf{qx}_i.C_C + \gamma_i \cdot \mathsf{qx}_i.C_C'\right) + \beta^{m+n} \cdot C_C^\star \ .
\end{aligned}$$

7. Compute the accumulator witness $\mathsf{acc}.\mathbb{w} := (x, s, \sigma_A, \sigma_B, \sigma_C, \mathsf{acc}_{\mathsf{HP}}.\mathbb{w})$ where:

$$\begin{aligned}
x &:= \textstyle\sum_{j=1}^m \beta^{j-1} \cdot \mathsf{acc}_j.\mathbb{w}.x + \sum_{i=1}^n \beta^{m+i-1} \cdot \mathsf{qx}_i.x + \beta^{m+n} \cdot x^\star \ , \\
s &:= \textstyle\sum_{j=1}^m \beta^{j-1} \cdot \mathsf{acc}_j.\mathbb{w}.s + \sum_{i=1}^n \beta^{m+i-1} \cdot \mathsf{qw}_i.s + \beta^{m+n} \cdot s^\star \ , \\
\sigma_A &:= \textstyle\sum_{j=1}^m \beta^{j-1} \cdot \mathsf{acc}_j.\mathbb{w}.\sigma_A + \sum_{i=1}^n \beta^{m+i-1} \cdot \mathsf{qw}_i.\sigma_A + \beta^{m+n} \cdot \omega_A^\star \ , \\
\sigma_B &:= \textstyle\sum_{j=1}^m \beta^{j-1} \cdot \mathsf{acc}_j.\mathbb{w}.\sigma_B + \sum_{i=1}^n \beta^{m+i-1} \cdot \mathsf{qw}_i.\sigma_B + \beta^{m+n} \cdot \omega_B^\star \ , \\
\sigma_C &:= \textstyle\sum_{j=1}^m \beta^{j-1} \cdot \mathsf{acc}_j.\mathbb{w}.\sigma_C + \sum_{i=1}^n \beta^{m+i-1} \cdot \mathsf{qw}_i.\sigma_C + \beta^{m+n} \cdot \omega_C^\star \ .
\end{aligned}$$

8. Set the accumulator $\mathsf{acc} := (\mathsf{acc}.\mathbb{x}, \mathsf{acc}.\mathbb{w})$ and accumulation proof $\mathsf{pf} := (\mathsf{pf}_{\mathsf{HP}}, C_x^\star, C_A^\star, C_B^\star, C_C^\star)$.
9. Output $(\mathsf{acc}, \mathsf{pf})$.

Accumulation verifier. On input the accumulator verification key $\mathsf{avk} = (\tau, \mathsf{n}, \mathsf{avk}_{\mathsf{HP}})$, predicate instances $[\mathsf{qx}_i]_{i=1}^n$, old accumulator instances $[\mathsf{acc}_j.\mathbb{x}]_{j=1}^m$, a new accumulator instance $\mathsf{acc}.\mathbb{x} = (C_x, C_A, C_B, C_C, \mathsf{acc}_{\mathsf{HP}}.\mathbb{x})$, and an accumulation proof $\mathsf{pf} = (\mathsf{pf}_{\mathsf{HP}}, C_x^\star, C_A^\star, C_B^\star, C_C^\star)$, V works as below.

1. Compute $[\gamma_i]_{i=1}^n$ as in Step 1a of the accumulation prover P.

2. Compute $[\mathsf{qx}_{\mathsf{HP},i}]_{i=1}^{n}$ as in Step 1b of the accumulation prover P.
3. Compute $[\mathsf{acc}_{\mathsf{HP},j}.\mathbb{x}]_{j=1}^{m}$ as in Step 2a of the accumulation prover P.
4. Check that $\mathsf{AS}_{\mathsf{HP}}.\mathrm{V}^{\rho_{\mathsf{HP}}}(\mathsf{avk}_{\mathsf{HP}}, [\mathsf{qx}_{\mathsf{HP},i}]_{i=1}^{n}, [\mathsf{acc}_{\mathsf{HP},j}.\mathbb{x}]_{j=1}^{m}, \mathsf{acc}_{\mathsf{HP}}.\mathbb{x}, \mathsf{pf}_{\mathsf{HP}}) = 1$.
5. Compute β as in Step 5 of the accumulation prover P.
6. Perform the assignments in Step 6 of the accumulation prover P as equality checks (between the new accumulator instance and the input instances and old accumulator instances).

Decider. On input the decision key $\mathsf{dk} = (A, B, C, \mathsf{n}, \mathsf{ck}, \mathsf{dk}_{\mathsf{HP}})$ and an accumulator acc, D works as follows.

1. Parse the accumulator instance $\mathsf{acc}.\mathbb{x}$ as $(C_x, C_A, C_B, C_C, \mathsf{acc}_{\mathsf{HP}}.\mathbb{x})$.
2. Parse the accumulator witness $\mathsf{acc}.\mathbb{w}$ as $(x, s, \sigma_A, \sigma_B, \sigma_C, \mathsf{acc}_{\mathsf{HP}}.\mathbb{w})$.
3. Compute $s_A := A \begin{bmatrix} x \\ s \end{bmatrix}$, $s_B := B \begin{bmatrix} x \\ s \end{bmatrix}$, $s_C := C \begin{bmatrix} x \\ s \end{bmatrix}$, which are vectors in $\mathbb{F}^{\mathsf{M}}$.
4. Check that $C_x = \mathsf{CM.Commit}(\mathsf{ck}, x; 0)$.
5. Check that $C_A = \mathsf{CM.Commit}(\mathsf{ck}, s_A; \sigma_A)$.
6. Check that $C_B = \mathsf{CM.Commit}(\mathsf{ck}, s_B; \sigma_B)$.
7. Check that $C_C = \mathsf{CM.Commit}(\mathsf{ck}, s_C; \sigma_C)$.
8. Set $\mathsf{acc}_{\mathsf{HP}} := (\mathsf{acc}_{\mathsf{HP}}.\mathbb{x}, \mathsf{acc}_{\mathsf{HP}}.\mathbb{w})$ and check that $\mathsf{AS}_{\mathsf{HP}}.\mathrm{D}(\mathsf{dk}_{\mathsf{HP}}, \mathsf{acc}_{\mathsf{HP}}) = 1$.

3.8.3 Security proofs

We prove that the non-interactive argument for R1CS in Section 3.8.1 satisfies the zero knowledge and knowledge soundness definitions from Section 3.3.1. Then we provide proof sketches that the accumulation scheme for it in Section 3.8.2 satisfies the zero knowledge and knowledge soundness definitions from Section 3.4.

Lemma 3.8.2. *The non-interactive argument for R1CS satisfies perfect zero knowledge.*

Proof. Consider the simulator $\mathcal{S}$ that is first given the security parameter λ in unary and invokes the generator $\mathcal{G}$ to sample the public parameters (in particular, there are no trapdoors). Subsequently, $\mathcal{S}$ receives as input an index $\mathbb{i} = (A, B, C, \mathsf{n})$ and an instance $\mathbb{x} = x \in \mathbb{F}^{\mathsf{n}}$, and works as follows.

1. Compute a commitment key ck and hash of coefficient matrices τ like the indexer $\mathcal{I}$ does.
2. Sample the following at random: $s \in \mathbb{F}^{\mathsf{N}-\mathsf{n}}$, $\sigma_A, \sigma_B, \sigma_C, \sigma_\circ \in \mathbb{F}$, and $C'_A, C'_B, C_1, C_2 \in \mathbb{G}$.
3. Set $\pi_2 := (x, s, \sigma_A, \sigma_B, \sigma_C, \sigma_\circ)$.
4. Compute $s_A := A \begin{bmatrix} x \\ s \end{bmatrix}$, $s_B := B \begin{bmatrix} x \\ s \end{bmatrix}$, $s_C := C \begin{bmatrix} x \\ s \end{bmatrix}$.
5. Compute $C_x := \mathsf{CM.Commit}(\mathsf{ck}, x; 0)$
6. Sample a random challenge $\gamma \in \mathbb{F}$.
7. Compute $C_A := \mathsf{CM.Commit}(\mathsf{ck}, s_A; \sigma_A) - \gamma C'_A$.
8. Compute $C_B := \mathsf{CM.Commit}(\mathsf{ck}, s_B; \sigma_B) - \gamma C'_B$.
9. Compute $C_C := \mathsf{CM.Commit}(\mathsf{ck}, s_A \circ s_B; \sigma_C) - \gamma C_1 - \gamma^2 C_2$.

10. Compute $C'_C := \gamma^{-1}\big(\mathsf{CM.Commit}(\mathsf{ck}, s_C; \sigma_C) - C_C\big)$.
11. Set $\pi_1 := (C_A, C_B, C_C, C'_A, C'_B, C'_C, C_1, C_2)$.
12. Program the random oracle ρ to output γ on input π_1.
13. Output $\pi := (\pi_1, \pi_2)$, along with the programming $\mu := [\pi_1 \mapsto \gamma]$.

By construction the output proof string π makes the verifier accept when its random oracle is programmed with μ. Moreover, the distribution of all elements in the proof string π is random subject to the condition that the proof string π is accepting. □

Lemma 3.8.3. *The non-interactive argument for R1CS satisfies knowledge soundness.*

Proof. We prove a stronger knowledge soundness property that what is required in Section 3.3.1: there exists an extractor $\mathcal{E}$ such that for every (non-uniform) adversary $\tilde{\mathcal{P}}$ running in expected polynomial time and auxiliary input distribution $\mathcal{D}$,

$$\Pr\left[\begin{array}{c|r} \mathcal{V}^{\rho}(\mathrm{ivk}, \mathbb{x}, \pi) = 1 & \rho \leftarrow \mathcal{U}(\lambda) \\ \Downarrow & \mathsf{pp} \leftarrow \mathcal{G}(1^{\lambda}) \\ (\mathbb{i}, \mathbb{x}, \mathbb{w}) \in \mathcal{R} & \mathsf{ai} \leftarrow \mathcal{D}(1^{\lambda}) \\ & (\mathbb{i}, \mathbb{x}, \pi; r) \leftarrow \tilde{\mathcal{P}}^{\rho}(\mathsf{pp}, \mathsf{ai}) \\ & \mathbb{w} \leftarrow \mathcal{E}^{\tilde{\mathcal{P}}, \rho}(\mathsf{pp}, \mathsf{ai}, r) \\ & (\mathrm{ipk}, \mathrm{ivk}) \leftarrow \mathcal{I}^{\rho}(\mathsf{pp}, \mathbb{i}) \end{array}\right] \geq 1 - \mathrm{negl}(\lambda) \ .$$

We construct the extractor $\mathcal{E}$ based on our forking lemma (Lemma 3.6.1).

Define the following algorithm:

$A^{\rho}((\mathsf{pp}, \mathsf{ai}))$:
1. $(\mathbb{i}, \mathbb{x}, \pi) \leftarrow \tilde{\mathcal{P}}^{\rho}(\mathsf{pp}, \mathsf{ai})$.
2. Compute $(\mathrm{ipk}, \mathrm{ivk}) := \mathcal{I}^{\rho}(\mathsf{pp}, \mathbb{i})$.
3. Parse the index verification key ivk as $(\mathsf{ck}, A, B, C, \mathsf{n}, \tau)$, and the proof string π as (π_1, π_2).
4. Set the query $\mathfrak{q} := (\tau, \mathbb{x}, \pi_1)$.
5. Set the output $\mathfrak{o}$ to $(\mathbb{i}, \pi_2)$.
6. Query the random oracle ρ at $\mathfrak{q}$.
7. Output $(\mathfrak{q}, \mathfrak{o})$.

Define the forking lemma predicate:

$\mathsf{p}\big((\mathsf{pp}, \mathsf{ai}), (\mathfrak{q}, \mathfrak{a}), \mathfrak{o}, \mathsf{tr}\big)$:
1. Parse the query $\mathfrak{q}$ as $(\tau, \mathbb{x}, \pi_1)$.
2. Parse the output $\mathfrak{o}$ as a pair $(\mathbb{i}, \pi_2)$.
3. Check that $\tau = \mathsf{tr}(\mathbb{i})$; if not, output 0.
4. Compute $(\mathrm{ipk}, \mathrm{ivk}) := \mathcal{I}^{\rho}(\mathsf{pp}, \mathbb{i})$, answering its queries to ρ with tr.
5. Check that $\mathcal{V}^{\rho}\big(\mathrm{ivk}, \mathbb{x}, (\pi_1, \pi_2)\big)$ outputs 1 when answering its query to ρ with $\mathfrak{a}$.

Let $\mathcal{E}$ be the extractor that runs the forking algorithm Fork^A obtained by applying Lemma 3.6.1 to p to obtain three outputs. With all but negligible probability it obtains $(\tau, \mathbb{x}, \pi_1)$ and tuples $(\gamma, (\mathbb{i}, \pi_2))$, $(\gamma', (\mathbb{i}', \pi_2'))$, $(\gamma'', (\mathbb{i}'', \pi_2''))$ satisfying p with $\gamma, \gamma', \gamma''$ pairwise distinct. This implies that (π_1, γ, π_2) is an accepting transcript for the underlying sigma protocol with respect to $\mathbb{i}$; similarly for (π_1, γ', π_2') with respect to $\mathbb{i}'$ and $(\pi_1, \gamma'', \pi_2'')$ with respect to $\mathbb{i}''$. Moreover, since $\tau = \mathsf{tr}(\mathbb{i}) = \mathsf{tr}(\mathbb{i}') = \mathsf{tr}(\mathbb{i}'')$, it holds by collision resistance of the random oracle that $\mathbb{i} = \mathbb{i}' = \mathbb{i}''$ with all but negligible probability. The extractor then computes and outputs $w := \frac{\gamma}{\gamma - \gamma'} s' - \frac{\gamma'}{\gamma - \gamma'} s \in \mathbb{F}^{\mathsf{N}-\mathsf{n}}$.

We argue that $\mathbb{w} := w$ is a valid witness for the index-instance pair $(\mathbb{i}, \mathbb{x})$ output by $\tilde{\mathcal{P}}$.

Define $r := \frac{1}{\gamma - \gamma'}(s - s') \in \mathbb{F}^{\mathsf{N}-\mathsf{n}}$. We first extract an opening of C_A to $A[x \| w]$. Since the verifier accepts, $C_A + \gamma C_A'$ opens to $A[x \| s] = A[x \| (w + \gamma r)]$; likewise for γ' and s'. Using the linear homomorphism of CM, we can solve the system to open C_A to $A[x \| w]$. Similar reasoning allows us to open C_B, C_C to $B[x \| w], C[x \| w]$ respectively.

By definition of w, r it holds that $s = w + \gamma r$ and $s' = w + \gamma' r$. Moreover by the binding property of CM it holds that $s_A'' = A[x \| (w + \gamma'' r)]$, and likewise for s_B''.

Next we use this fact with the Hadamard product check to show that the R1CS equation holds.

We argue that C_C commits to the Hadamard product of the vectors inside C_A and C_B. Note that the following holds as a polynomial identity in Y:

$$A \begin{bmatrix} x \\ w + Yr \end{bmatrix} \circ B \begin{bmatrix} x \\ w + Yr \end{bmatrix} \equiv A \begin{bmatrix} x \\ w \end{bmatrix} \circ B \begin{bmatrix} x \\ w \end{bmatrix} + \left(A \begin{bmatrix} x \\ w \end{bmatrix} \circ B \begin{bmatrix} 0 \\ r \end{bmatrix} + A \begin{bmatrix} 0 \\ r \end{bmatrix} \circ B \begin{bmatrix} x \\ w \end{bmatrix} \right) Y + \left(A \begin{bmatrix} 0 \\ r \end{bmatrix} \circ B \begin{bmatrix} 0 \\ r \end{bmatrix} \right) Y^2.$$

Since the NARK verifier accepts, we know that $C_C + \gamma C_1 + \gamma^2 C_2$ is a commitment to the evaluation of the above polynomial at γ; the same is true with respect to γ' and γ'' for the associated commitments. We can hence solve a linear system to open C_C to $A[x \| w] \circ B[x \| w]$. By the binding of CM, it then holds that $C[x \| w] = A[x \| w] \circ B[x \| w]$. This means that $\mathbb{w} = w$ is a valid R1CS witness with respect to $(\mathbb{i}, \mathbb{x})$. □

Lemma 3.8.4. *The split accumulation scheme for R1CS satisfies perfect zero knowledge.*

Proof. Let S_{HP} be the simulator for $\mathsf{AS}_{\mathsf{HP}}$, and suppose that it does not rely on a trapdoor or program the random oracle (this is the case for our construction in Section 3.7). Consider the simulator S for AS that works as follows:

$\mathrm{S}^{\rho}(\tau = \bot, \mathsf{pp}_{\Phi} = \mathsf{pp}_{\mathsf{NARK}}, \mathsf{i}_{\Phi} = (A, B, C, \mathsf{n}))$:

1. Sample $(x, s) \in \mathbb{F}^{\mathsf{N}}$.
2. Compute $s_A := A \cdot (x, s)$, $s_B := B \cdot (x, s)$, $s_C := C \cdot (x, s)$, which are vectors in $\mathbb{F}^{\mathsf{M}}$.
3. Compute $C_x := \mathsf{CM.Commit}(\mathsf{ck}, x; 0)$.
4. Compute $C_A := \mathsf{CM.Commit}(\mathsf{ck}, s_A; \sigma_A)$.
5. Compute $C_B := \mathsf{CM.Commit}(\mathsf{ck}, s_B; \sigma_B)$.
6. Compute $C_C := \mathsf{CM.Commit}(\mathsf{ck}, s_C; \sigma_C)$.
7. Sample $\mathsf{acc}_{\mathsf{HP}} \leftarrow \mathrm{S}_{\mathsf{HP}}(\tau_{\mathsf{HP}} = \bot, \mathsf{pp}_{\Phi}, \ell)$.
8. Set the accumulator instance $\mathsf{acc}.\mathbb{x} := (C_x, C_A, C_B, C_C, \mathsf{acc}_{\mathsf{HP}}.\mathbb{x})$.

9. Set the accumulator witness $\mathsf{acc}.\mathbb{w} := (x, s, \sigma_A, \sigma_B, \sigma_C, \mathsf{acc}_{\mathsf{HP}}.\mathbb{w})$.
10. Output $\mathsf{acc} := (\mathsf{acc}.\mathbb{x}, \mathsf{acc}.\mathbb{w})$.

By construction, the sampled accumulator satisfies the decider. Moreover, the accumulator is distributed identically as an accumulator output by the (honest) accumulation prover. This is because $\mathsf{acc}_{\mathsf{HP}}$ is sampled by the simulator S_{HP} for $\mathsf{AS}_{\mathsf{HP}}$ (which we have assumed is zero knowledge) and all other elements of the accumulator are random within the respective domains subject only to the condition that the decider accepts the accumulator. □

Lemma 3.8.5. *The split accumulation scheme for R1CS satisfies knowledge soundness.*

Proof. We describe an extractor E and then argue that it satisfies the knowledge property in Section 3.4; E has access to the random oracle ρ that consists of three domain-separated random oracles $\rho = (\rho_{\mathsf{AS}}, \rho_{\mathsf{HP}}, \rho_{\mathsf{NARK}})$.

Below we use the notation $(A^{\rho_{\mathsf{int}}})^{\rho_{\mathsf{ext}}}$ to distinguish between an "external" oracle ρ_{ext} that is exposed to the extractor and "internal" oracles ρ_{int} that are used only to run the adversary $\tilde{\mathrm{P}}$.

Define the following algorithm:

$(A^{\rho_{\mathsf{HP}}, \rho_{\mathsf{NARK}}})^{\rho_{\mathsf{AS}}}\big((\mathsf{pp}, \mathsf{pp}_\Phi, \mathsf{ai})\big)$:

1. $(\mathsf{i}_\Phi = (A, B, C, \mathsf{n}), [\mathsf{acc}_j.\mathbb{x}]_{j=1}^m, [\mathsf{qx}_i]_{i=1}^n, \mathsf{acc}, \mathsf{pf}) \leftarrow \tilde{\mathrm{P}}^\rho(\mathsf{pp}, \mathsf{pp}_\Phi, \mathsf{ai})$.
2. Compute $\tau := \rho_{\mathsf{NARK}}(A, B, C, \mathsf{n})$.
3. Set the query $\mathfrak{q} := (\tau, [\mathsf{acc}_j.\mathbb{x}]_{j=1}^m, [\mathsf{qx}_i]_{i=1}^n, C_x^\star, C_A^\star, C_B^\star, C_C^\star)$.
4. Set the output $\mathsf{o} := (\mathsf{i}_\Phi, \mathsf{acc}, \mathsf{pf})$.
5. Query the random oracle ρ_{AS} at $\mathfrak{q}$.
6. Output $(\mathfrak{q}, \mathsf{o})$.

Define the forking lemma predicate:

$\mathsf{p}^{\rho_{\mathsf{HP}}, \rho_{\mathsf{NARK}}}\big((\mathsf{pp}, \mathsf{pp}_\Phi, \mathsf{ai}), (\mathfrak{q}, \mathfrak{a}), \mathsf{o}, \mathsf{tr}\big)$:

1. Parse the query $\mathfrak{q}$ as $(\tau, [\mathsf{acc}_j.\mathbb{x}]_{j=1}^m, [\mathsf{qx}_i]_{i=1}^n, C_x^\star, C_A^\star, C_B^\star, C_C^\star)$.
2. Parse the output o as $(\mathsf{i}_\Phi, \mathsf{acc}, \mathsf{pf})$.
3. Check that $\tau = \rho_{\mathsf{NARK}}(\mathsf{i}_\Phi)$.
4. Compute $(\mathsf{apk}, \mathsf{avk}, \mathsf{dk}) := \mathrm{I}^{\rho_{\mathsf{NARK}}}(\mathsf{pp}, \mathsf{pp}_\Phi, \mathsf{i}_\Phi)$.
5. Check that $\mathrm{V}^{\rho_{\mathsf{HP}}, \rho_{\mathsf{NARK}}}(\mathsf{avk}, [\mathsf{qx}_i]_{i=1}^n, [\mathsf{acc}_j.\mathbb{x}]_{j=1}^m, \mathsf{acc}.\mathbb{x}, \mathsf{pf})$ outputs 1 when answering its query to ρ_{AS} with $\mathfrak{a}$.
6. Check that $\mathrm{D}(\mathsf{dk}, \mathsf{acc})$ outputs 1.

Finally, define an adversary $\tilde{\mathrm{P}}_{\mathsf{HP}}$ for the Hadamard product accumulation scheme $\mathsf{AS}_{\mathsf{HP}}$.

$(\tilde{\mathrm{P}}_{\mathsf{HP}}^{\rho_{\mathsf{AS}}, \rho_{\mathsf{NARK}}})^{\rho_{\mathsf{HP}}}(\mathsf{pp}, \mathsf{pp}_\Phi, \mathsf{ai})$:

1. $(\mathsf{i}_\Phi = (A, B, C, \mathsf{n}), [\mathsf{qx}_i]_{i=1}^n, [\mathsf{acc}_j.\mathbb{x}]_{j=1}^m, \mathsf{acc}, \mathsf{pf}) \leftarrow \tilde{\mathrm{P}}^\rho(\mathsf{pp} = \mathsf{pp}_{\mathsf{AS}_{\mathsf{HP}}}, \mathsf{pp}_\Phi = \mathsf{pp}_{\mathsf{HP}}, \mathsf{ai})$.
2. Compute $\mathsf{qx}_{\mathsf{HP},i}$ from qx_i for all $i \in [n]$ as in Step 1b of the accumulation prover.
3. Set the vector length to be $\ell := \mathsf{M}$, the number of rows in each R1CS coefficient matrix.

4. Output $(\ell, [\mathsf{qx}_{\mathsf{HP},i}]_{i=1}^{n}, [\mathsf{acc}_j.\mathbb{x}.\mathsf{acc}_{\mathsf{HP}}.\mathbb{x}]_{j=1}^{m})$.

For the remainder of the proof we implicitly consider only the case that the verifier and decider accept the malicious prover's output, i.e., $\mathrm{V}^{\rho}(\mathsf{avk}, [\mathsf{qx}_i]_{i=1}^{n}, [\mathsf{acc}_j.\mathbb{x}]_{j=1}^{m}, \mathsf{acc}.\mathbb{x}, \mathsf{pf}) = 1$ and $\mathrm{D}(\mathsf{dk}, \mathsf{acc}) = 1$ for $(\mathsf{i}_\Phi, [\mathsf{qx}_i]_{i=1}^{n}, [\mathsf{acc}_j.\mathbb{x}]_{j=1}^{m}, \mathsf{acc}, \mathsf{pf}) \leftarrow \tilde{\mathrm{P}}^{\rho}(\mathsf{pp}, \mathsf{pp}_\Phi, \mathsf{ai})$ and $(\mathsf{apk}, \mathsf{avk}, \mathsf{dk}) := \mathrm{I}(\mathsf{pp}, \mathsf{pp}_\Phi, \mathsf{i}_\Phi)$; otherwise, the implication holds vacuously. In this case the output of A satisfies p with probability 1. Let Fork be the algorithm given by applying Lemma 3.6.1 to the forking lemma predicate p.

$\mathrm{E}^{\tilde{\mathrm{P}},\rho}(\mathsf{pp}, \mathsf{pp}_\Phi, \mathsf{ai}, r)$ for $\rho = (\rho_{\mathsf{AS}}, \rho_{\mathsf{HP}}, \rho_{\mathsf{NARK}})$:

1. Run $(\mathsf{q}, \mathsf{o}; \mathsf{tr}) \leftarrow A^{\rho}\big((\mathsf{pp}, \mathsf{pp}_\Phi, \mathsf{ai}); r\big)$.
2. Parse the query q as $(\tau, [\mathsf{acc}_j.\mathbb{x}]_{j=1}^{m}, [\mathsf{qx}_i]_{i=1}^{n}, C_x^\star, C_A^\star, C_B^\star, C_C^\star)$.
3. Parse the output o as a tuple $(\mathsf{i}_\Phi, \mathsf{acc}, \mathsf{pf})$.
4. Run $\mathrm{E}_{\mathsf{HP}}^{(\tilde{\mathrm{P}}_{\mathsf{HP}}^{\rho_{\mathsf{AS}},\rho_{\mathsf{NARK}}}),\rho_{\mathsf{HP}}}$ to extract predicate witnesses $[\mathsf{qw}_{\mathsf{HP},i}]_{i=1}^{n}$ and accumulator witnesses $[\mathsf{acc}_{\mathsf{HP},j}.\mathbb{w}]_{j=1}^{m}$.
5. Run $[\beta_j, \mathsf{o}_j]_{j=1}^{n+m+1} \leftarrow \mathsf{Fork}^{(A^{\rho_{\mathsf{HP}},\rho_{\mathsf{NARK}}})}(\mathsf{pp}, 1^{n+m+1}, (\mathsf{q}, \rho(\mathsf{q})), \mathsf{o}, \mathsf{tr}_{\mathsf{q}}, r)$.
6. For each $j \in [n+m+1]$:
   a) parse the output o_j as a tuple $(\mathsf{i}_\Phi^{(j)}, \mathsf{acc}^{(j)}, \mathsf{pf}^{(j)})$, and the accumulator $\mathsf{acc}^{(j)}$ as $(\mathsf{acc}.\mathbb{x}^{(j)}, \mathsf{acc}.\mathbb{w}^{(j)})$;
   b) parse the accumulator instance $\mathsf{acc}.\mathbb{x}^{(j)}$ as $(C_{x,\star}^{(j)}, C_{A,\star}^{(j)}, C_{B,\star}^{(j)}, C_{C,\star}^{(j)}, \mathsf{acc}_{\mathsf{HP},\star}.\mathbb{x}^{(j)})$;
   c) parse the accumulator witness $\mathsf{acc}.\mathbb{w}^{(j)}$ as $(x_\star^{(j)}, s_\star^{(j)}, \sigma_{A,\star}^{(j)}, \sigma_{B,\star}^{(j)}, \sigma_{C,\star}^{(j)}, \mathsf{acc}_{\mathsf{HP},\star}.\mathbb{w}^{(j)})$.
7. Set M to be the Vandermonde matrix on $(1, \beta, \ldots, \beta^{m+n})$.
8. If M is not invertible, abort. Otherwise, compute

$$\begin{pmatrix} x_1 & s_1 & \sigma_{A,1} & \sigma_{B,1} & \sigma_{C,1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_k & s_k & \sigma_{A,k} & \sigma_{B,k} & \sigma_{C,k} \end{pmatrix} := M^{-1} \cdot \begin{pmatrix} x_\star^{(1)} & s_\star^{(1)} & \sigma_{A,\star}^{(1)} & \sigma_{B,\star}^{(1)} & \sigma_{C,\star}^{(1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_\star^{(k)} & s_\star^{(k)} & \sigma_{A,\star}^{(k)} & \sigma_{B,\star}^{(k)} & \sigma_{C,\star}^{(k)} \end{pmatrix},$$

   where $k := n + m + 1$.
9. For each $i \in [n]$, set $\mathsf{qw}_i = (s_i, \sigma_{A,i}, \sigma_{B,i}, \sigma_{C,i}, \mathsf{qw}_{\mathsf{HP},i})$.
10. For each $j \in [m]$, $\mathsf{acc}_j.\mathbb{w} = (x_{n+j}, s_{n+j}, \sigma_{A,n+j}, \sigma_{B,n+j}, \sigma_{C,n+j}, \mathsf{acc}_{\mathsf{HP},j}.\mathbb{w})$.
11. Output $\big(\mathsf{i}_\Phi, \mathsf{acc}, [(\mathsf{qx}_i, \mathsf{qw}_i)]_{i=1}^{n}, [(\mathsf{acc}_j.\mathbb{x}, \mathsf{acc}_j.\mathbb{w})]_{j=1}^{m}, \mathsf{pf}\big)$.

By the properties of Fork guaranteed in Lemma 3.6.1, and the extraction guarantee of $\mathsf{AS}_{\mathsf{HP}}$ (Theorem 3.7.2), $\mathrm{E}_{\tilde{\mathrm{P}}}$ runs in expected polynomial time and, except with probability $\mathrm{negl}(\lambda)$, the following event E holds:

$$\begin{aligned} &[\beta_j]_{j=1}^{n+m+1} \text{ are pairwise distinct} \\ &\text{and } \forall j \in [n+m+1],\ \mathsf{p}\big((\mathsf{pp}, \mathsf{pp}_\Phi, \mathsf{ai}), (\mathsf{q}, \beta_j), \mathsf{o}_j, \mathsf{tr}_{\mathsf{q}}\big) = 1, \\ &\text{and } \forall i \in [n],\ \Phi_{\mathsf{HP}}(\mathsf{pp}, \mathsf{pp}_\Phi, \mathsf{qx}_{\mathsf{HP},i}, \mathsf{qw}_{\mathsf{HP},i}) = 1, \\ &\text{and } \forall j \in [m],\ \mathrm{D}_{\mathsf{HP}}\big(\mathsf{dk}, (\mathsf{acc}_{\mathsf{HP},j}.\mathbb{x}, \mathsf{acc}_{\mathsf{HP},j}.\mathbb{w})\big) = 1. \end{aligned}$$

Moreover, since $\rho_{\mathsf{NARK}}(\mathsf{i}_\Phi^{(j)}) = \tau$ for all j, with all but negligible probability (over the randomness of ρ_{NARK}), $\mathsf{i}_\Phi^{(1)} = \cdots = \mathsf{i}_\Phi^{(n+m+1)}$. Hence we consider a single index $\mathsf{i}_\Phi = (A, B, C, \mathsf{n})$ for the remainder of the proof.

We complete the proof of knowledge soundness by showing two claims. Claim 3.8.6 shows that the extracted assignments s_i obey the correct linear relations with respect to the commitments output

by A. Claim 3.8.7 then uses the binding property of the commitment scheme and the guarantee of the Hadamard product extractor to show that these assignments satisfy the R1CS equation and the decider as appropriate. □

Claim 3.8.6. *Define the following:*

$$\forall\, i \in [n] \qquad s_A^{(i)} := A \begin{bmatrix} \mathsf{qx}_i.x \\ s_i \end{bmatrix} \qquad s_B^{(i)} := B \begin{bmatrix} \mathsf{qx}_i.x \\ s_i \end{bmatrix} \qquad s_C^{(i)} := C \begin{bmatrix} \mathsf{qx}_i.x \\ s_i \end{bmatrix}$$

$$\forall\, j \in [m] \qquad s_A^{(n+j)} := A \begin{bmatrix} x_{n+j} \\ s_{n+j} \end{bmatrix} \qquad s_B^{(n+j)} := B \begin{bmatrix} x_{n+j} \\ s_{n+j} \end{bmatrix} \qquad s_C^{(n+j)} := C \begin{bmatrix} x_{n+j} \\ s_{n+j} \end{bmatrix}$$

The event E implies that with overwhelming probability:

$$\begin{aligned}
\forall\, i \in [n] \qquad \mathsf{qx}_i.C_A + \gamma_i \cdot \mathsf{qx}_i.C'_A &= \mathsf{CM.Commit}(\mathsf{ck}, s_A^{(i)}; \sigma_{A,i}) \ , \\
\mathsf{qx}_i.C_B + \gamma_i \cdot \mathsf{qx}_i.C'_B &= \mathsf{CM.Commit}(\mathsf{ck}, s_B^{(i)}; \sigma_{B,i}) \ , \\
\mathsf{qx}_i.C_C + \gamma_i \cdot \mathsf{qx}_i.C'_C &= \mathsf{CM.Commit}(\mathsf{ck}, s_C^{(i)}; \sigma_{C,i}) \ , \\
\forall\, j \in [m] \qquad \mathsf{acc}_j.\mathbb{x}.C_x &= \mathsf{CM.Commit}(\mathsf{ck}, x_{n+j}; 0) \ , \\
\mathsf{acc}_j.\mathbb{x}.C_A &= \mathsf{CM.Commit}(\mathsf{ck}, s_A^{(n+j)}; \sigma_{A,n+j}) \ , \\
\mathsf{acc}_j.\mathbb{x}.C_B &= \mathsf{CM.Commit}(\mathsf{ck}, s_B^{(n+j)}; \sigma_{B,n+j}) \ , \\
\mathsf{acc}_j.\mathbb{x}.C_C &= \mathsf{CM.Commit}(\mathsf{ck}, s_C^{(n+j)}; \sigma_{C,n+j}) \ .
\end{aligned}$$

Proof. We prove the statements for A. The statements for B, C follow similarly.

Define the following $(n+m+1)$-entry vectors:

$$\begin{aligned}
\vec{C}_A &:= (\mathsf{qx}_1.C_A + \gamma_1 \cdot \mathsf{qx}_1.C'_A, \ldots, \mathsf{qx}_n.C_A + \gamma_n \cdot \mathsf{qx}_n.C'_A, \mathsf{acc}_1.\mathbb{x}.C_A, \ldots, \mathsf{acc}_m.\mathbb{x}.C_A, C_A^\star) \ , \\
\vec{C}_{A,\star} &:= (C_{A,\star}^{(1)}, \ldots, C_{A,\star}^{(n+m+1)}) \ .
\end{aligned}$$

Recall that if p holds then both the accumulation verifier and decider accept. Since the accumulation verifier accepts, it holds that $\vec{C}_{A,\star} = M\vec{C}_A$. Moreover, since the decider accepts $[(\mathsf{dk}, \mathsf{acc}^{(j)})]_{i=1}^{n+m+1}$, it holds for all $j \in [n+m+1]$ that $C_{A,\star}^{(j)} = \mathsf{CM.Commit}(\mathsf{ck}, A[x_\star^{(j)} \ \ s_\star^{(j)}]; \sigma_{A,\star}^{(j)})$. Using the homomorphic property of $\mathsf{CM.Commit}$ and that $M^{-1}\vec{C}_{A,\star} = \vec{C}_A$, we conclude that

$$\forall\, i \in [n] \ \ \mathsf{qx}_i.C_A + \gamma_i \cdot \mathsf{qx}_i.C'_A = \mathsf{CM.Commit}(\mathsf{ck}, A \begin{bmatrix} x_i \\ s_i \end{bmatrix}; \sigma_{A,i}) = \mathsf{CM.Commit}(\mathsf{ck}, s_A^{(i)}; \sigma_{A,i}) \ ,$$

$$\forall\, j \in [m] \ \ \mathsf{acc}_j.\mathbb{x}.C_A = \mathsf{CM.Commit}(\mathsf{ck}, A \begin{bmatrix} x_{n+j} \\ s_{n+j} \end{bmatrix}; \sigma_{A,n+j}) = \mathsf{CM.Commit}(\mathsf{ck}, s_A^{(n+j)}; \sigma_{A,n+j}) \ .$$

Finally, similarly to the above, since the accumulation verifier and decider accept, it holds that

$$\begin{aligned}
\forall\, i \in [n] \ \ \mathsf{CM.Commit}(\mathsf{qx}_i.x; 0) &= \mathsf{CM.Commit}(x_i; 0) \ , \\
\forall\, j \in [m] \ \ \mathsf{acc}_j.\mathbb{x}.C_x &= \mathsf{CM.Commit}(x_{n+j}; 0) \ .
\end{aligned}$$

Hence by the binding property of CM, $x_i = \mathsf{qx}_i.x$ for all $i \in [n]$ with all but negligible probability. □

Claim 3.8.7. *The event E implies that with overwhelming probability it holds that*

$$\forall\, i \in [n] \;\; A \begin{bmatrix} \mathsf{qx}_i.x \\ s_i \end{bmatrix} \circ B \begin{bmatrix} \mathsf{qx}_i.x \\ s_i \end{bmatrix} = C \begin{bmatrix} \mathsf{qx}_i.x \\ s_i \end{bmatrix} \;\; ,$$
$$\forall\, j \in [m] \;\; \mathrm{D}\big(\mathsf{dk}, (\mathsf{acc}_j.\mathbb{x}, \mathsf{acc}_j.\mathbb{w})\big) = 1 \;\; .$$

Proof. Fix $i \in [n]$ and write $\mathsf{qw}_{\mathsf{HP},i} = (a^{(i)}, b^{(i)}, \omega_1^{(i)}, \omega_2^{(i)}, \omega_3^{(i)})$. The event E implies that

$$C_1 = \mathsf{CM.Commit}(\mathsf{ck}, a^{(i)}; \omega_1^{(i)}) = \mathsf{CM.Commit}(\mathsf{ck}, s_A^{(i)}, \sigma_{A,i}) \;\; ,$$
$$C_2 = \mathsf{CM.Commit}(\mathsf{ck}, b^{(i)}; \omega_2^{(i)}) = \mathsf{CM.Commit}(\mathsf{ck}, s_B^{(i)}; \sigma_{B,i}) \;\; ,$$
$$C_3 = \mathsf{CM.Commit}(\mathsf{ck}, a^{(i)} \circ b^{(i)}; \omega_3^{(i)}) = \mathsf{CM.Commit}(\mathsf{ck}, s_C^{(i)}; \sigma_{C,i}) \;\; .$$

If it is not the case that $a^{(i)} = s_A^{(i)}$, $b^{(i)} = s_B^{(i)}$, and $a^{(i)} \circ b^{(i)} = s_C^{(i)}$, then the extractor breaks the binding property of CM, which can occur with only negligible probability. It follows that with all but negligible probability, $s_A^{(i)} \circ s_B^{(i)} = s_C^{(i)}$.

The event E also implies that E_{HP} produces witnesses $[\mathsf{acc}_{\mathsf{HP},j}.\mathbb{w}]_{j=1}^m$ such that the decider accepts: $\mathrm{D}_{\mathsf{HP}}\big(\mathsf{dk}, (\mathsf{acc}_{\mathsf{HP},j}.\mathbb{x}, \mathsf{acc}_{\mathsf{HP},j}.\mathbb{w})\big) = 1$ for all $j \in [m]$. Together with Claim 3.8.6 this shows that for all $j \in [m]$, $\mathrm{D}\big(\mathsf{dk}, (\mathsf{acc}_j.\mathbb{x}, \mathsf{acc}_j.\mathbb{w})\big) = 1$. □

3.9 Implementation

We contribute a generic and modular implementation of proof-carrying data based on accumulation schemes. Our implementation includes several components of independent interest.

Framework for accumulation. We design and implement a generic framework for accumulation schemes that supports arbitrary predicates/relations. The main interface is a Rust trait that defines the behavior of any (atomic or split) accumulation scheme. We implement this trait for several accumulation schemes:

- the atomic accumulation scheme $\mathsf{AS}_{\mathsf{AGM}}$ in [BCMS20] for the PC scheme $\mathsf{PC}_{\mathsf{AGM}}$;
- the atomic accumulation scheme $\mathsf{AS}_{\mathsf{IPA}}$ in [BCMS20] for the PC scheme $\mathsf{PC}_{\mathsf{IPA}}$;
- the split accumulation scheme $\mathsf{AS}_{\mathsf{PC}}$ in Section 3.11 for the polynomial commitment predicate Φ_{PC} (corresponding to the check algorithm of the trivial PC scheme $\mathsf{PC}_{\mathsf{Ped}}$);
- the split accumulation scheme $\mathsf{AS}_{\mathsf{HP}}$ in Section 3.7 for the Hadamard product predicate Φ_{HP};
- the split accumulation scheme $\mathsf{AS}_{\mathsf{R1CS}}$ for the zkNARK for R1CS in Section 3.8.

Our framework also provides a generic trait for defining R1CS constraints for the verifier of an accumulation scheme. We use this trait to implement R1CS constraints for all of these accumulation schemes.

PCD from accumulation. We provide a generic construction of PCD from accumulation, which simultaneously supports the case of atomic accumulation from [BCMS20] and the case of split accumulation from Section 3.5. Our code builds on and extends an existing PCD library that offers a generic "PCD" trait.[9] We instantiate this PCD trait via a modular construction, which takes as ingredients any NARK (as defined by an appropriate trait), accumulation scheme for that NARK that implements the accumulation trait (from above), and constraints for the accumulation verifier. We use our concrete instantiations of these ingredients to achieve recursion based on accumulation for each of $\mathsf{PC}_{\mathsf{AGM}}$, $\mathsf{PC}_{\mathsf{IPA}}$, Φ_{PC}, and Φ_{HP}. In particular, we obtain a simple construction of PCD based on the zkNARK for R1CS and its split accumulation from Section 3.8.1.

Cycles of elliptic curves. All PCD constructions in our implementation rely on the technique of cycles of elliptic curves [BCTV14]: PCD based on $\mathsf{PC}_{\mathsf{AGM}}$ uses cycles of pairing-friendly curves, while PCD based on $\mathsf{PC}_{\mathsf{IPA}}$, Φ_{PC}, and Φ_{HP} uses cycles of standard curves. For all of these, we rely on existing implementations from the `arkworks` ecosystem:[10] for pairing-friendly cycles we use the MNT cycle of curves (low security and high security variants), while for standard cycles we use the Pasta cycle of curves [Hop20].

Remark 3.9.1. Many of the aforementioned accumulation schemes compute linear combinations with respect to powers of a single challenge derived from the random oracle. In our implementation, when possible, we instead use linear combinations where the coefficients are multiple independent challenges obtained from the random oracle, because this leads to lower constraint costs for the accumulation verifier.

[9] `https://github.com/arkworks-rs/pcd`

[10] `https://github.com/arkworks-rs/curves`

This modification requires minor modifications in the security proofs. The knowledge extractor rewinds the prover several times to build a tree of accepting transcripts, and extraction succeeds if certain matrices constructed from the challenges of these transcripts are invertible. When using powers of challenges each matrix is a Vandermonde matrix, which is invertible precisely when the

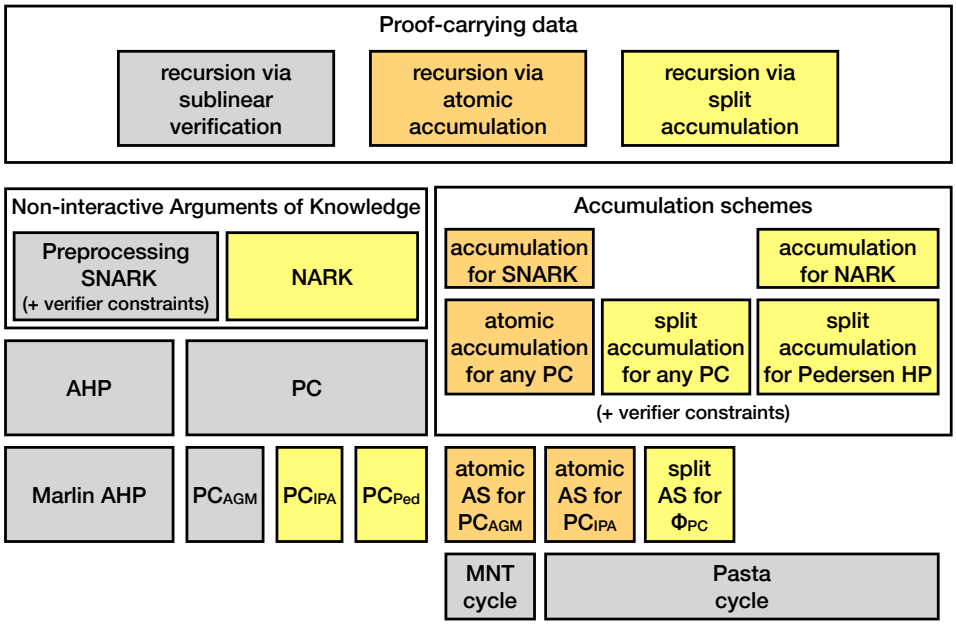

Figure 3.6: Diagram illustrating components in our implementation. The gray boxes denote components that exist in prior libraries; the orange boxes denote our implementation of components from [BCMS20]; and the yellow boxes denote our implementation of components contributed in this work.

3.10 Evaluation

We perform an evaluation focused on the discrete logarithm setting.[11] In Section 3.10.1 we describe the concrete costs of our zkNARK for R1CS and its split accumulation scheme; and in Section 3.10.2 we compare the costs of atomic versus split accumulation for PC schemes based on Pedersen commitments.

In Figure 3.7 we report the asymptotic cost of $|\mathrm{V}|$ (the constraint cost of V) in $\mathsf{AS}_{\mathsf{IPA}}$, $\mathsf{AS}_{\mathsf{PC}}$, and $\mathsf{AS}_{\mathsf{R1CS}}$.[12] Note that because these accumulation schemes share many common subcomponents (scalar multiplication, random oracle calls, non-native field arithmetic), any improvements would preserve the relative cost.

Experimental setup. All experiments are performed using a single thread on a machine with an Intel Xeon 6136 CPU at $3.0\,\mathrm{GHz}$. The reported numbers are for schemes instantiated over the 255-bit prime-order Pallas curve in the Pasta cycle [Hop20]; results for the Vesta curve in that cycle would be similar.

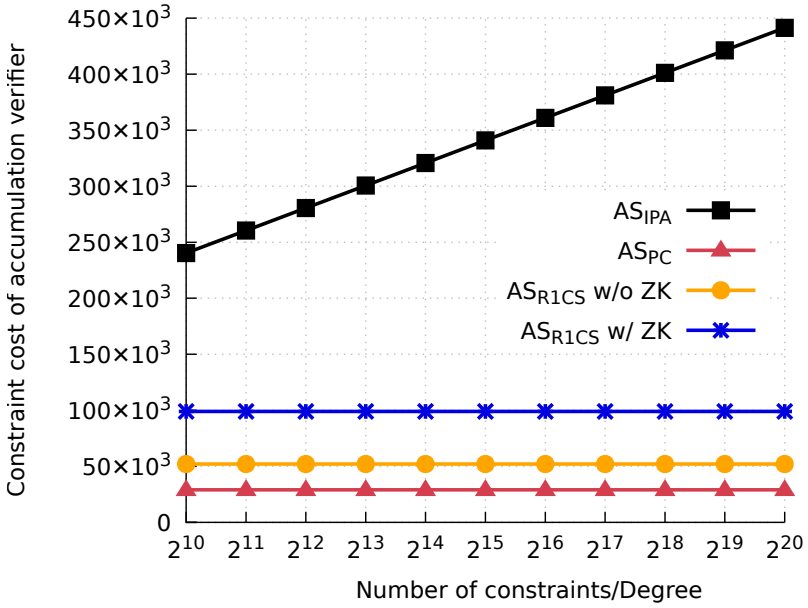

Figure 3.7: Comparison of the constraint cost of the accumulation verifier V in $\mathsf{AS}_{\mathsf{IPA}}$, $\mathsf{AS}_{\mathsf{PC}}$, and $\mathsf{AS}_{\mathsf{R1CS}}$ when varying the number of constraints (for $\mathsf{AS}_{\mathsf{R1CS}}$) or the degree of the accumulated polynomial (for $\mathsf{AS}_{\mathsf{IPA}}$ and $\mathsf{AS}_{\mathsf{PC}}$) from 2^{10} to 2^{20}. Note that the cost of accumulating $\mathsf{PC}_{\mathsf{IPA}}$ and $\mathsf{PC}_{\mathsf{Ped}}$ is a lower bound on the cost of accumulating any SNARK built atop those, and this enables comparing against the cost of $\mathsf{AS}_{\mathsf{R1CS}}$.

[11] The pairing setting is also part of our implementation, as described in Section 3.9, but we do not include an evaluation for it here.

[12] This comparison is meaningful because the cost of accumulating polynomial commitments provides a lower bound on the cost accumulating SNARKs that rely on these PC schemes.

3.10.1 Split accumulation for R1CS

In Tables 3.2 and 3.3, we compare the costs of our accumulation scheme for our zkNARK for R1CS for an illustrative number of constraints, with and without zero knowledge. We include the metric of lines of code (LoC) to highlight the simplicity of our constructions. We focus on the special case where the accumulation scheme is used to accumulate one new proof into one old accumulator to obtain a new accumulator (this corresponds to the case of IVC). We find that the cost in both cases is modest, and the overhead of zero knowledge is less than a factor of 2 in the number of constraints. Furthermore, the measured cost matches the expected asymptotic cost. In more detail, while the prover time and decider time are both linear in the number of constraints, the verifier cost (both wall-clock time and constraint cost) does not grow with the number of constraints. This latter point is illustrated in Fig. 3.7.

zk?	$\mathcal{P}$	$\mathcal{V}$	$\lvert\pi\rvert$	LoC
no	2.9 s	3.9 s	4.19 MB	618
yes	6.9 s	3.9 s	4.19 MB	

Table 3.2: Cost of proving and verifying a constraint system containing 2^{17} constraints.

zk?	P	V	D	$\lvert\mathsf{acc}\rvert$		$\lvert\mathrm{V}\rvert$	LoC	
				$\lvert\mathbb{x}\rvert$	$\lvert\mathbb{w}\rvert$		native	constraints
no	2.0 s	2 ms	6.0 s	392 B	8.4 MB	52×10^3	1258	1120
yes	8.1 s	3 ms	6.3 s	392 B	8.4 MB	99×10^3		

Table 3.3: Cost of accumulating a NARK proof and an old accumulator, for a constraint system of size 2^{17}.

3.10.2 Accumulation for polynomial commitments based on DL

We compare the costs of two accumulation schemes for two PC schemes:
- the atomic accumulation scheme $\mathsf{AS}_{\mathsf{IPA}}$ in [BCMS20] for the PC scheme $\mathsf{PC}_{\mathsf{IPA}}$;
- the split accumulation scheme $\mathsf{AS}_{\mathsf{PC}}$ in Section 3.11 for the predicate Φ_{PC} corresponding to $\mathsf{PC}_{\mathsf{Ped}}$.

In Section 3.10.2.1 we compare the two polynomial commitment schemes $\mathsf{PC}_{\mathsf{IPA}}$ and $\mathsf{PC}_{\mathsf{Ped}}$, and in Section 3.10.2.2 we compare the two corresponding accumulation schemes $\mathsf{AS}_{\mathsf{IPA}}$ and $\mathsf{AS}_{\mathsf{PC}}$.

3.10.2.1 Comparing polynomial commitments based on DL

We compare the performance of $\mathsf{PC}_{\mathsf{IPA}}$ and $\mathsf{PC}_{\mathsf{Ped}}$ in Table 3.4, reporting experiments for an illustrative choice of polynomial degree d. In both PC schemes all operations (commit, open, check) are linear in the degree d, though for $\mathsf{PC}_{\mathsf{Ped}}$ opening is concretely much cheaper than $\mathsf{PC}_{\mathsf{IPA}}$ (primarily because $\mathsf{PC}_{\mathsf{Ped}}$ has a trivial opening procedure). The main difference between the two PC schemes is that an evaluation proof in $\mathsf{PC}_{\mathsf{Ped}}$ is $O(d)$ field elements while an evaluation proof in $\mathsf{PC}_{\mathsf{IPA}}$ is $O(\log d)$ group elements; this asymptotic difference is apparent in the reported numbers (the proof size for

$\mathsf{PC}_{\mathsf{Ped}}$ is significantly larger than for $\mathsf{PC}_{\mathsf{IPA}}$). We also report lines of code to realize the same abstract PC scheme trait, to support the (intuitive) claim that $\mathsf{PC}_{\mathsf{Ped}}$ is a much simpler primitive than $\mathsf{PC}_{\mathsf{IPA}}$.

PC scheme	Commit	Open	Check	$\lvert C\rvert$	$\lvert\pi\rvert$	LoC
$\mathsf{PC}_{\mathsf{IPA}}$	8.0 s	106.6 s	8.2 s	33 B	1.4 kB $O(\log d)$ $\mathbb{G}$	1120
$\mathsf{PC}_{\mathsf{Ped}}$	8.1 s	0.43 s	8.3 s	33 B	33.5 MB $O(d)$ $\mathbb{F}$	608

Table 3.4: Comparison between the PC schemes $\mathsf{PC}_{\mathsf{IPA}}$ and $\mathsf{PC}_{\mathsf{Ped}}$ for polynomials of degree $d = 2^{20}$.

3.10.2.2 Comparing accumulation schemes based on DL

We compare the performance of $\mathsf{AS}_{\mathsf{IPA}}$ and $\mathsf{AS}_{\mathsf{PC}}$ in Table 3.5, reporting experiments for an illustrative choice of polynomial degree d. We focus on the special case where the accumulation scheme is used to accumulate one new polynomial evaluation claim into one old accumulator to obtain a new accumulator. Our experiments indicate that $\mathsf{AS}_{\mathsf{PC}}$ is cheaper than $\mathsf{AS}_{\mathsf{IPA}}$ across all metrics *except for accumulator size*, and more generally that performance is consistent with the asymptotic comparison from Table 3.1. In more detail:

- While prover time (per claim) in both $\mathsf{AS}_{\mathsf{IPA}}$ and $\mathsf{AS}_{\mathsf{PC}}$ are linear in the degree d, our experiments show that $\mathsf{AS}_{\mathsf{IPA}}$ is concretely much more expensive than $\mathsf{AS}_{\mathsf{PC}}$.
- Decider time in both $\mathsf{AS}_{\mathsf{IPA}}$ and $\mathsf{AS}_{\mathsf{PC}}$ are linear in the degree d, and our experiments show that the two schemes have similar concrete performance.
- Verifier time (per claim) in $\mathsf{AS}_{\mathsf{IPA}}$ is logarithmic while in $\mathsf{AS}_{\mathsf{PC}}$ it is constant, and our experiments confirm that $\mathsf{AS}_{\mathsf{IPA}}$ is concretely significantly more expensive than $\mathsf{AS}_{\mathsf{PC}}$.
- Verifier constraint cost is *much* higher for $\mathsf{AS}_{\mathsf{IPA}}$, even though both schemes use the same underlying constraint gadget libraries.
- The size of an atomic accumulator for $\mathsf{AS}_{\mathsf{IPA}}$ is logarithmic, and amounts to a few kilobytes; in contrast an accumulator for $\mathsf{AS}_{\mathsf{PC}}$ is much larger, but is split into a short instance part (106 bytes) and a long witness part (33.5 megabytes).

Overall the expensive parts of $\mathsf{AS}_{\mathsf{PC}}$ are exactly where intended (a large accumulation witness part) in exchange for a very cheap verifier and a very short accumulation instance part; all other metrics are comparable to (and concretely better than for) $\mathsf{AS}_{\mathsf{IPA}}$.

scheme	P	V	D	$\lvert\mathsf{acc}\rvert$		$\lvert\mathrm{V}\rvert$	LoC	
				$\lvert\mathbb{x}\rvert$	$\lvert\mathbb{w}\rvert$		native	constraints
$\mathsf{AS}_{\mathsf{IPA}}$	117.6 s	14 ms	8.3 s	1.58 kB	0	435×10^3	664	1232
$\mathsf{AS}_{\mathsf{PC}}$	25.2 s	2 ms	8.1 s	106 B	33.5 MB	30×10^3	571	395

Table 3.5: Comparison between the accumulation schemes $\mathsf{AS}_{\mathsf{IPA}}$ and $\mathsf{AS}_{\mathsf{PC}}$ for polynomials of degree $d = 2^{20}$, when accumulating one old accumulator and one evaluation claim into a new accumulator.

In Figure 3.7, we also compare $\lvert\mathrm{V}\rvert$ (the constraint cost of V) in both $\mathsf{AS}_{\mathsf{PC}}$ and $\mathsf{AS}_{\mathsf{IPA}}$ as we accumulate polynomial evaluation claims of degree d in the range 2^{10} to 2^{20}. As expected, the cost

for $\mathsf{AS}_{\mathsf{PC}}$ is a small constant, whereas the cost of $\mathsf{AS}_{\mathsf{IPA}}$ grows logarithmically (and is concretely much larger).

3.11 Split accumulation for Pedersen polynomial commitments

We construct a split accumulation scheme for Pedersen commitments to polynomials. We define the predicate we accumulate and then state our theorem. The remainder of the section is dedicated to proving the theorem.

Definition 3.11.1. *The* **(Pedersen) polynomial commitment predicate** Φ_{PC} *takes as input: (i) public parameters* $\mathsf{pp}_\Phi = \mathsf{pp}_{\mathsf{CM}}$ *for the Pedersen commitment scheme (for messages of some maximum length* $D+1$*); (ii) an index* $\mathsf{i}_\Phi = d$ *specifying a supported degree (at most* D*); (iii) an instance* $\mathsf{qx} = (C, z, v) \in \mathbb{G} \times \mathbb{F} \times \mathbb{F}$ *consisting of a commitment to a polynomial, a point at which it is evaluated, and the evaluation; (iv) a witness* $\mathsf{qw} = p \in \mathbb{F}^{\leq d}[X]$ *consisting of the committed polynomial. The predicate* Φ_{PC} *computes the Pedersen commitment key* $\mathsf{ck} := \mathsf{CM.Trim}(\mathsf{pp}_{\mathsf{CM}}, d+1)$ *for messages of length* $d+1$*, and checks that* $C = \mathsf{CM.Commit}(\mathsf{ck}, p)$*,* $p(z) = v$*, and* $\deg(p) \leq d$*.*

Theorem 3.11.2. *The scheme* $\mathsf{AS} = (\mathrm{G}, \mathrm{I}, \mathrm{P}, \mathrm{V}, \mathrm{D})$ *constructed in Section 3.11.1 is a split accumulation scheme in the random oracle model (assuming the hardness of the discrete logarithm problem) for the polynomial commitment predicate* Φ_{PC} *in Definition 3.11.1.* AS *achieves the efficiency stated below.*

- Generator: $\mathrm{G}(1^\lambda)$ *runs in time* $O(\lambda)$*.*
- Indexer: *The time of* $\mathrm{I}(\mathsf{pp}, \mathsf{pp}_\Phi, \mathsf{i}_\Phi = d)$ *is dominated by the time to run* $\mathsf{CM.Trim}$ *for messages of length* $d+1$*.*
- Accumulation prover: *The time of* $\mathrm{P}^\rho(\mathsf{apk}, [(\mathsf{qx}_i, \mathsf{qw}_i)]_{i=1}^n, [\mathsf{acc}_j]_{j=1}^m)$ *is dominated by the time to commit to* $n+m$ *polynomials of degree* d *(i.e,* $n+m$ *multi-scalar multiplications of size* $d+1$*).*
- Accumulation verifier: *The time of* $\mathrm{V}^\rho(\mathsf{avk}, [\mathsf{qx}_i]_{i=1}^n, [\mathsf{acc}_j.\mathbb{x}]_{j=1}^m, \mathsf{acc}.\mathbb{x}, \mathsf{pf})$ *is dominated by* $O(n+m)$ *field additions/multiplications and* $O(n+m)$ *group scalar multiplications.*
- Decider: *The time of* $\mathrm{D}(\mathsf{dk}, \mathsf{acc})$ *is dominated by the time to commit to a polynomial of degree at most* d*.*
- Sizes: *An accumulator* acc *consists of (a) an accumulator instance* $\mathsf{acc}.\mathbb{x}$ *consisting of a commitment and two field elements, and (b) an accumulator witness* $\mathsf{acc}.\mathbb{w}$ *consisting of a polynomial of degree less than* d*. An accumulation proof* pf *consists of* n *commitments and* $2n + 2m$ *field elements.*

Recall from Section 3.2.6 that the predicate Φ_{PC} can be seen as equivalent to checking an evaluation claim in the trivial polynomial commitment (PC) scheme $\mathsf{PC}_{\mathsf{Ped}}$: the evaluation proof is simply the original polynomial. This PC scheme is a drop-in replacement for PC schemes used in existing SNARKs [GWC19; CHMMVW20], and facilitates accumulation of the verifier for the resulting SNARKs.

3.11.1 Construction

We describe the accumulation scheme $\mathsf{AS} = (\mathrm{G}, \mathrm{I}, \mathrm{P}, \mathrm{V}, \mathrm{D})$ for the Pedersen polynomial commitment predicate Φ_{PC}. Predicate instances qx have the form (C, z, v), and predicate witnesses qw consist

of a polynomial p (allegedly, committed inside C and such that $p(z) = v$ and $\deg(p) < d$). An accumulator acc is split in two parts that are analogous to predicate instances and predicate witnesses. Jumping ahead, the decider D is equal to the predicate Φ_{PC}; therefore, there is no distinction between inputs and prior accumulators, and so it suffices to accumulate inputs only.

Generator. The generator G receives as input $\mathsf{pp} := 1^\lambda$ and outputs 1^λ. (In other words, G does not have to create additional public parameters beyond those used by Φ_{PC}.)

Indexer. On input the accumulator parameters pp, predicate parameters $\mathsf{pp}_\Phi = \mathsf{pp}_{\mathsf{PC}}$, and a predicate index $\mathsf{i}_\Phi = d$, the indexer I computes the commitment key $\mathsf{ck} := \mathsf{CM.Trim}(\mathsf{pp}_{\mathsf{PC}}, d+1)$, and then outputs the accumulator proving key $\mathsf{apk} := \mathsf{ck}$, the accumulator verification key $\mathsf{avk} := d$, and the decision key $\mathsf{dk} := \mathsf{ck}$.

Accumulation prover. On input the accumulation proving key apk and predicate instance-witness pairs $[(\mathsf{qx}_i, \mathsf{qw}_i)]_{i=1}^n$ (of the same form as split accumulators $[\mathsf{acc}_j]_{j=1}^m = [(\mathsf{acc}_j.\mathbb{x}, \mathsf{acc}_j.\mathbb{w})]_{j=1}^m$), P works as below.

$\mathrm{P}^\rho(\mathsf{apk} = \mathsf{ck}, [(\mathsf{qx}_i, \mathsf{qw}_i)]_{i=1}^n)$:

1. For each i in $[n]$:
   a) Parse the predicate instance qx_i as an evaluation claim $(C_i, z_i, v_i) \in \mathbb{G} \times \mathbb{F} \times \mathbb{F}$.
   b) Parse the predicate witness qw_i as a polynomial $p_i(X) \in \mathbb{F}^{\leq \mathsf{ck}.d}[X]$.
   c) Compute the witness polynomial $w_i(X) := \frac{p_i(X) - v_i}{X - z_i} \in \mathbb{F}[X]$.
   d) Compute a commitment to $w_i(X)$: $W_i := \mathsf{CM.Commit}(\mathsf{ck}, w_i) \in \mathbb{G}$.
2. Use the random oracle to compute the evaluation point $z_\star := \rho(d, [(C_i, z_i, v_i, W_i)]_{i=1}^n) \in \mathbb{F}$.
3. For each i in $[n]$, compute the evaluations $y_i := p_i(z_\star) \in \mathbb{F}$ and $y_i' := w_i(z_\star) \in \mathbb{F}$.
4. Use the random oracle to compute the challenge $\alpha := \rho(z_\star, [(y_i, y_i')]_{i=1}^n) \in \mathbb{F}$.
5. Compute the linear combination $p_\star(X) := \sum_{i=1}^n \alpha^{i-1} \cdot p_i(X) + \sum_{i=1}^n \alpha^{n+i-1} \cdot w_i(X) \in \mathbb{F}[X]$.
6. Compute the evaluation $v_\star := p_\star(z_\star) \in \mathbb{F}$.
7. Compute the linear combination $C_\star := \sum_{i=1}^n \alpha^{i-1} \cdot C_i + \sum_{i=1}^n \alpha^{n+i-1} \cdot W_i \in \mathbb{G}$.
8. Set the split accumulator $\mathsf{acc} := (\mathsf{acc}.\mathbb{x}, \mathsf{acc}.\mathbb{w})$ where $\mathsf{acc}.\mathbb{x} := (C_\star, z_\star, v_\star)$ and $\mathsf{acc}.\mathbb{w} := p_\star$.
9. Set the accumulation proof $\mathsf{pf} := [(W_i, y_i, y_i')]_{i=1}^n$.
10. Output $(\mathsf{acc}, \mathsf{pf})$.

Accumulation verifier. On input the accumulator verification key avk, predicate instances $[\mathsf{qx}_i]_{i=1}^n$ (of the same form as accumulator instances $[\mathsf{acc}_j.\mathbb{x}]_{j=1}^m$), a new accumulator instance $\mathsf{acc}.\mathbb{x}$, and an accumulation proof pf, V works as below.

$\mathrm{V}^\rho(\mathsf{avk} = d, [\mathsf{qx}_i]_{i=1}^n, \mathsf{acc}.\mathbb{x}, \mathsf{pf})$:

1. For each $i \in [n]$, parse qx_i as (C_i, z_i, v_i).
2. Parse $\mathsf{acc}.\mathbb{x}$ as $(C_\star, z_\star, v_\star)$, and pf as $[(W_i, y_i, y_i')]_{i=1}^n$.
3. Check that $z_\star = \rho(d, [(C_i, z_i, v_i, W_i)]_{i=1}^n)$.
4. For each $i \in [n]$, check that $y_i - v_i = y_i' \cdot (z_\star - z_i)$.
5. Compute $\alpha := \rho(z_\star, [(y_i, y_i')]_{i=1}^n)$.
6. Check that $v_\star = \sum_{i=1}^n \alpha^{i-1} \cdot y_i + \sum_{i=i}^n \alpha^{n+i-1} \cdot y_i'$.

7. Check that $C_\star = \sum_{i=1}^{n} \alpha^{i-1} \cdot C_i + \sum_{i=1}^{n} \alpha^{n+i-1} \cdot W_i$.

Decider. On input the decision key $\mathsf{dk} = \mathsf{ck}$ and an accumulator acc, D parses $\mathsf{acc}.\mathbb{x}$ as (C, z, v), parses $\mathsf{acc}.\mathbb{w}$ as p, and checks $C = \mathsf{CM.Commit}(\mathsf{ck}, p)$, $p(z) = v$, and $\deg(p) < |\mathsf{ck}|$.

3.11.2 Zero-finding games

The following lemma, due to [BCMS20], bounds the probability that applying the random oracle to a binding commitment to a polynomial yields a zero of that polynomial. We refer to this as a *zero-finding game*. Here we have adapted the lemma to expected-time adversaries; the proof is essentially unchanged.

The statement of the lemma involves the definition of a binding commitment scheme, given below. Even if in this work we focus on accumulation schemes based on Pedersen commitments, in the security proofs we need to invoke the lemma on binding commitment schemes that are related, *but not equal*, to Pedersen commitments. Hence we require this technical lemma with respect to a general binding commitment scheme.

Lemma 3.11.3 ([BCMS20])**.** *Let* $\mathsf{CM} = (\mathsf{Setup}, \mathsf{Trim}, \mathsf{Commit})$ *be a binding commitment scheme and* L *a message format for* CM*. Let* $F\colon \mathbb{N} \to \mathbb{N}$ *be a field size function,* $N \in \mathbb{N}$ *a number of variables, and* $D \in \mathbb{N}$ *a total degree bound. For every family of (possibly inefficient) functions* $\{f_{\mathsf{pp}}\colon \mathcal{M}_{\mathsf{pp}} \to \mathbb{F}_{\mathsf{pp}}^{\leq D}[X_1, \ldots, X_N]\}_{\mathsf{pp}}$ *mapping messages to polynomials of degree at most* D *over fields of size* $|\mathbb{F}_{\mathsf{pp}}| \geq F(\lambda)$ *and every* t*-query oracle algorithm* $\mathcal{A}$ *that runs in expected polynomial time, the following holds:*

$$\Pr\left[\begin{array}{l|r} \mathfrak{p} \in \mathcal{M}_{\mathsf{ck}} & \rho \leftarrow \mathcal{U}(\lambda) \\ \wedge\ \boldsymbol{z} \in \mathbb{F}_{\mathsf{pp}}^{N} & \mathsf{pp} \leftarrow \mathsf{CM.Setup}(1^\lambda, L) \\ \wedge\ p \not\equiv 0 & (\ell, \mathfrak{p}, \omega) \leftarrow \mathcal{A}^{\rho}(\mathsf{pp}) \\ \wedge\ p(\boldsymbol{z}) = 0 & \mathsf{ck} \leftarrow \mathsf{CM.Trim}(\mathsf{pp}, \ell) \\ & C \leftarrow \mathsf{CM.Commit}(\mathsf{ck}, \mathfrak{p}; \omega) \\ & \boldsymbol{z} \leftarrow \rho(C) \\ & p \leftarrow f_{\mathsf{pp}}(\mathfrak{p}) \end{array}\right] \leq \sqrt{\frac{(t+1) \cdot D}{F(\lambda)}} + \mathrm{negl}(\lambda) \ .$$

Remark 3.11.4. For Lemma 3.11.3 to hold, the algorithms of CM *must not* have access to the random oracle ρ used to generate the challenge point $\boldsymbol{z}$. The lemma is otherwise black-box with respect to CM, and so CM itself may use *other* oracles. The lemma continues to hold when $\mathcal{A}$ has access to these additional oracles. We use this fact later to justify the security of domain separation.

3.11.3 Proof of Theorem 3.11.2

We prove that the accumulation scheme constructed in the previous section satisfies the claimed efficiency properties and achieves completeness, and then, in Section 3.11.3.1, that it achieves knowledge soundness.

Efficiency. We now analyze the efficiency of our accumulation scheme.

- *Generator:* $\mathrm{G}(1^\lambda)$ outputs 1^λ, and hence runs in time $O(\lambda)$.
- *Indexer:* $\mathrm{I}^\rho(\mathsf{pp}, \mathsf{pp}_\Phi, \mathsf{i}_\Phi)$ invokes $\mathsf{CM.Trim}$, and hence runs in time $O_\lambda(d)$.
- *Accumulation prover:* $\mathrm{P}^\rho(\mathsf{apk}, [(\mathsf{qx}_i, \mathsf{qw}_i)]_{i=1}^n)$ computes a commitment to the degree $\deg(p_i) - 1$ witness polynomial w_i for each input $\mathsf{qx}_i = (C_i, z_i, v_i)$. The time to generate these n commitments dominates the running time of P.
- *Accumulation verifier:* $\mathrm{V}^\rho(\mathsf{avk}, [\mathsf{qx}_i]_{i=1}^n, \mathsf{acc}.\mathbb{x}, \mathsf{pf})$ computes a random linear combination between $2n$ commitments, and hence its running time is as claimed.
- *Decider:* $\mathrm{D}(\mathsf{dk}, \mathsf{acc})$ invokes $\mathsf{CM.Commit}$ and checks that the output matches the accumulator.
- *Sizes:* The accumulator instance $\mathsf{acc}.\mathbb{x}$ consists of a polynomial commitment C, an evaluation point z and an evaluation claim v. The accumulator witness $\mathsf{acc}.\mathbb{w}$ is a polynomial of degree d. The accumulation proof pf contains $O(n)$ group and field elements.

Completeness. Since we need only accumulate predicate inputs (as accumulators are split like predicate inputs and the decider equals the predicate being accumulated), it suffices to demonstrate that the simplified completeness property from Section 3.4.1 holds. Fix an (unbounded) adversary $\mathcal{A}$. For each $i \in [n]$, since

$$\Phi_{\mathsf{PC}}\big(\mathsf{pp}_\Phi, \mathsf{i}_\Phi, \mathsf{qx}_i = (C_i, z_i, v_i), \mathsf{qw}_i = p_i\big) = 1 \ ,$$

we know that $C_i = \mathsf{CM.Commit}(\mathsf{ck}, p_i)$ and $p_i(z_i) = v_i$; this implies that each witness polynomial $w_i(X) = \frac{p_i(X) - v_i}{X - z_i}$ is indeed a polynomial of degree $d - 1$.

Together with the fact that the accumulation prover P behaves honestly, the foregoing facts imply that C is a well-formed commitment to $p_\star = \sum_{i=1}^n \alpha^i p_i + \sum_{i=1}^n \alpha^{n+i} w_i$, and that $p_\star(z_\star) = v_\star$, as required.

3.11.3.1 Knowledge soundness

We need only accumulate predicate inputs (as accumulators are split like predicate inputs and the decider equals the predicate being accumulated), so it suffices to demonstrate that the simplified knowledge soundness property from Section 3.4.1 holds. We describe an extractor and then analyze why it satisfies the property.

Define the following algorithm:

$A^\rho\big((\mathsf{pp}, \mathsf{pp}_\Phi, \mathsf{ai})\big)$:

1. $(\mathsf{i}_\Phi = d, [\mathsf{qx}_i]_{i=1}^n, \mathsf{acc}, \mathsf{pf}) \leftarrow \tilde{\mathrm{P}}^\rho(\mathsf{pp}, \mathsf{pp}_\Phi, \mathsf{ai})$.
2. Parse, for each $i \in [n]$, qx_i as (C_i, z_i, v_i).
3. Parse acc as $(C, z, v; p)$, and pf as $[(W_i, y_i, y_i')]_{i=1}^n$.
4. Set $\mathsf{q} := (z, [y_i, y_i']_{i=1}^n)$.
5. Set $\mathsf{o} := (\mathsf{i}_\Phi, [\mathsf{qx}_i]_{i=1}^n, \mathsf{acc}, \mathsf{pf})$.

6. Query ρ at the points $(d, [(C_i, z_i, v_i, W_i)]_{i=1}^n)$ and q, if not already queried by $\tilde{\mathrm{P}}$.
7. Output (q, o).

Define the forking lemma predicate:

$\mathsf{p}\big((\mathsf{pp}, \mathsf{pp}_\Phi, \mathsf{ai}), (\mathsf{q}, \mathfrak{a}), \mathsf{o}, \mathsf{tr}\big)$:

1. Check that tr contains no collisions.
2. Parse the query q as $(z, [(y_i, y_i')]_{i=1}^n)$, and the output o as $(\mathsf{i}_\Phi = d, [\mathsf{qx}_i]_{i=1}^n, \mathsf{acc}, \mathsf{pf})$.
3. Compute $(\mathsf{apk}, \mathsf{avk}, \mathsf{dk}) := \mathrm{I}(\mathsf{pp}, \mathsf{pp}_\Phi, d)$.
4. Check that $\mathrm{V}(\mathsf{avk}, [\mathsf{qx}_i]_{i=1}^n, \mathsf{acc}.\mathbb{x}, \mathsf{pf})$ outputs 1 when answering its first query according to tr and its second query with $\mathfrak{a}$. (If the first query is outside the support of tr then output 0.)
5. Check that $\mathrm{D}(\mathsf{dk}, \mathsf{acc})$ outputs 1.

For the remainder of the proof we implicitly consider only the case that $\mathrm{V}^\rho(\mathsf{avk}, [\mathsf{qx}_i]_{i=1}^n, \mathsf{acc}.\mathbb{x}, \mathsf{pf}) = 1$ and $\mathrm{D}(\mathsf{dk}, \mathsf{acc}) = 1$ for $(\mathsf{i}_\Phi, [\mathsf{qx}_i]_{i=1}^n, \mathsf{acc}, \mathsf{pf}) \leftarrow \tilde{\mathrm{P}}^\rho(\mathsf{pp}, \mathsf{pp}_\Phi, \mathsf{ai})$ and $(\mathsf{apk}, \mathsf{avk}, \mathsf{dk}) := \mathrm{I}(\mathsf{pp}, \mathsf{pp}_\Phi, d)$; otherwise, the implication holds vacuously. The probability that V accepts when its first query is outside the support of tr_q, or that tr contains a collision, is $O(t^2/2^\lambda)$, and so the output of A fails to satisfy p with probability at most $\mathrm{negl}(\lambda)$. Let Fork be the algorithm given by applying Lemma 3.6.1 to the forking lemma predicate p.

$\mathrm{E}^{\tilde{\mathrm{P}}, \rho}(\mathsf{pp}, \mathsf{pp}_\Phi, \mathsf{ai}, r)$:

1. Run $(\mathsf{q}, \mathsf{o}; \mathsf{tr}) \leftarrow A^\rho\big((\mathsf{pp}, \mathsf{pp}_\Phi, \mathsf{ai}); r\big)$; parse o as $(\mathsf{i}_\Phi = d, [\mathsf{qx}_i]_{i=1}^n, \mathsf{acc}, \mathsf{pf})$.
2. Run $(\alpha_1, \mathsf{o}_1, \ldots, \alpha_{2n}, \mathsf{o}_{2n}) \leftarrow \mathsf{Fork}^A((\mathsf{pp}, \mathsf{pp}_\Phi, \mathsf{ai}), 1^{2n}, (\mathsf{q}, \rho(\mathsf{q})), \mathsf{o}, \mathsf{tr}_\mathsf{q}, r)$.
3. For $j \in [2n]$:
   - parse o_j as $(\mathsf{i}_\Phi^{(j)} = d, [\mathsf{qx}_i^{(j)}]_{i=1}^n, \mathsf{pf}^{(j)}, \mathsf{acc}^{(j)})$;
   - parse $\mathsf{acc}^{(j)}$ as $\mathsf{acc}^{(j)}.\mathbb{x} = (C_\star^{(j)}, z_\star^{(j)}, v_\star^{(j)}) \in \mathbb{G} \times \mathbb{F} \times \mathbb{F}$ and $\mathsf{acc}^{(j)}.\mathbb{w} = p_\star^{(j)} \in \mathbb{F}^{\leq d}[X]$.
4. Set $\vec{p}_\star := \begin{pmatrix} p_\star^{(1)} \\ \vdots \\ p_\star^{(2n)} \end{pmatrix}$, and set M to be the Vandermonde matrix on $(\alpha_1, \ldots, \alpha_{2n})$.
5. If M is invertible, compute $(\vec{p} \| \vec{w}) := M^{-1} \cdot \vec{p}_\star$; otherwise, abort.
6. Output $\big(\mathsf{i}_\Phi, [(\mathsf{qx}_i, p_i)]_{i=1}^n, \mathsf{acc}, \mathsf{pf}\big)$.

By the properties of Fork guaranteed in Lemma 3.6.1, $\mathrm{E}_{\tilde{\mathrm{P}}}$ runs in expected polynomial time and, moreover, except with probability $\mathrm{negl}(\lambda)$ the following event E holds:

$$\{\alpha_j\}_{j \in [2n]} \text{ are pairwise distinct and } \forall j \in [2n]\ \mathsf{p}\big((\mathsf{pp}, \mathsf{pp}_\Phi, \mathsf{ai}), (\mathsf{q}, \alpha_j), \mathsf{o}_j, \mathsf{tr}_\mathsf{q}\big) = 1.$$

Conditioned on E, we observe the following. First, since the α_j are distinct, M is invertible. Next, let $(z_\star, [(y_i, y_i')]_{i=1}^n) := \mathsf{q}$; note that $z_\star^{(j)} = z_\star$ and $(d^{(j)}, [\mathsf{qx}_i^{(j)}, W_i^{(j)}]_{i=1}^n) = \mathsf{tr}_\mathsf{q}^{-1}(z_\star) = (d, [\mathsf{qx}_i, W_i]_{i=1}^n)$ for all j, whence also $\mathsf{pf}^{(j)} = \mathsf{pf}$ for all j. The function $\mathsf{tr}_\mathsf{q}^{-1}$ is well-defined only because p requires that tr_q contains no collisions. For the remainder of the proof we therefore omit the superscripts on $d, z_\star, \mathsf{qx}_i = (C_i, z_i, v_i)$, and $\mathsf{pf} = [(W_i, y_i, y_i')]_{i=1}^n$.

We conclude the proof with two claims. In Claim 3.11.5 we argue that the extracted polynomials $\vec{p}, \vec{w}$ are openings of the corresponding commitments, and that their evaluations at $z_\star$ are as claimed. In Claim 3.11.6 we argue that the evaluations of the polynomials $\{p_i\}_{i\in[n]}$ on the original query points $\{z_i\}_{i\in[n]}$ are as claimed. Together these claims establish that, with all but negligible probability, for all $i \in [n]$ it holds that $\Phi_{\mathsf{PC}}(\mathsf{pp}_{\mathsf{CM}}, d, (C_i, z_i, v_i), p_i) = 1$, which completes the proof of knowledge soundness.

Claim 3.11.5. *The event E implies that for each $i \in [n]$:*

$$C_i = \mathsf{CM.Commit}(\mathsf{ck}, p_i)\ , \qquad \deg(p_i) \le d\ , \qquad p_i(z_\star) = y_i\ ,$$
$$W_i = \mathsf{CM.Commit}(\mathsf{ck}, w_i)\ , \qquad \deg(w_i) \le d\ , \qquad w_i(z_\star) = y'_i\ .$$

Proof. Define the following vectors:

$$\vec{C} := (C_1, \ldots, C_n)\ , \qquad \vec{W} := (W_1, \ldots, W_n)\ , \qquad \vec{C}_\star := (C_\star^{(1)}, \ldots, C_\star^{(2n)})\ ,$$
$$\vec{y} := (y_1, \ldots, y_n)\ , \qquad \vec{y'} := (y'_1, \ldots, y'_n)\ , \qquad \vec{v}_\star := (v_\star^{(1)}, \ldots, v_\star^{(2n)})\ .$$

Above, for each $j \in [2n]$, $C_\star^j$ and $v_\star^{(j)}$ are the commitment and claimed evaluation in $\mathsf{acc}^{(j)}.\mathbb{x}$.

By the definition of the forking lemma predicate p, the accumulation verifier V accepts $(\mathsf{avk}, [\mathsf{qx}_i]_{i=1}^n, \mathsf{acc}^{(j)}.\mathbb{x}, \mathsf{pf})$ for all $j \in [2n]$. By the polynomial evaluation check in Step 6 of V we obtain that $\vec{v} = M \cdot (\vec{y} \| \vec{y'})$, and by the commitment check in Step 7 we obtain that $\vec{C}_\star = M \cdot (\vec{C} \| \vec{W})$. Moreover, since the decider accepts $(\mathsf{avk}, \mathsf{acc}^{(j)})$ for all $j \in [2n]$, it holds for all j that

$$C_\star^{(j)} = \mathsf{CM.Commit}(\mathsf{ck}, p_\star^{(j)})\ ,\ p_\star^{(j)}(z_\star) = v_\star^{(j)}\ ,\ \deg(p_\star^{(j)}) \le d\ .$$

From this the degree bounds on p_i, w_i follow by linearity.

Since $(\vec{C} \| \vec{W}) = M^{-1} \cdot \vec{C}_\star$, and by the homomorphic property of $\mathsf{PC}_{\mathsf{Ped}}$, for each $i \in [n]$ we have that

$$C_i = \textstyle\sum_j M^{-1}_{i,j} C_\star^{(j)} = \mathsf{PC}_{\mathsf{Ped}}.\mathsf{Commit}(\mathsf{ck}, \sum_j M^{-1}_{i,j} p_\star^{(j)}) = \mathsf{PC}_{\mathsf{Ped}}.\mathsf{Commit}(\mathsf{ck}, p_i)\ .$$

Similarly, $W_i = \mathsf{PC}_{\mathsf{Ped}}.\mathsf{Commit}(\mathsf{ck}, w_i)$ for each $i \in [n]$.

In addition, since $(\vec{y}, \vec{y'}) = M^{-1} \cdot \vec{v}$, and $p_\star^{(j)}(z_\star) = v_\star^{(j)}$, we have that $p_i(z_\star) = \sum_j M^{-1}_{i,j} v_\star^{(j)} = y_i$, and $w_i(z_\star) = \sum_j M^{-1}_{n+i,j} v_\star^{(j)} = y'_i$. □

Claim 3.11.6. *With probability at least $1 - \mathrm{negl}(\lambda)$, it holds that E implies $p_i(z_i) = v_i$ for all $i \in [n]$.*

Proof. Consider a modification to $\mathrm{E}_{\tilde{\mathrm{P}}}$ that also outputs $\vec{w}$. By Claim 3.11.5, if E occurs then for each $i \in [n]$ the tuple (C_i, z_i, v_i, W_i) is a binding commitment to the polynomial $p_i(X) - v_i - w_i(X) \cdot (X - z_i)$ of degree at most $d + 1$, and further it holds that $p_i(z_i) = y_i$ and $w_i(z_i) = y'_i$. Since the verifier accepts the output of $\mathrm{E}_{\tilde{\mathrm{P}}}$, we have that $z_\star = \mathsf{tr}(d, [(C_i, z_i, v_i, W_i)]_{i=1}^n) = \rho(d, [(C_i, z_i, v_i, W_i)]_{i=1}^n)$, and that $\forall\, i \in [n]\ y_i - v_i = y'_i \cdot (z_\star - z_i)$. By Lemma 3.11.3, except with probability $\mathrm{negl}(\lambda)$, $p_i(X) - v_i - w_i(X) \cdot (X - z_i)$ is the zero polynomial, and so $p_i(z_i) = v_i$. □

Chapter 4

ZEXE: Enabling Decentralized Private Computations

This chapter presents ZEXE, a ledger-based system where users can execute offline computations and subsequently produce transactions, attesting to the correctness of these computations, that satisfy two main properties. First, transactions *hide all information* about the offline computations. Second, transactions can be *validated in constant time* by anyone, regardless of the offline computation.

The core of ZEXE is a construction for a new cryptographic primitive that we introduce, *decentralized private computation* (DPC) schemes. In order to achieve an efficient implementation of our construction, we leverage tools in the area of cryptographic proofs, including succinct zero knowledge proofs and recursive proof composition. Overall, transactions in ZEXE are 968 bytes *regardless of the offline computation*, and generating them takes less than a minute plus a time that grows with the offline computation.

We demonstrate how to use ZEXE to realize privacy-preserving analogues of popular applications: private decentralized exchanges for user-defined fungible assets and regulation-friendly private stablecoins.

This work was previously published in [BCGMMW20].

4.1 Introduction

Distributed ledgers are a mechanism that maintains data across a distributed system while ensuring that every party has the same view of the data, even in the presence of corrupted parties. Ledgers can provide an indisputable history of all "events" logged in a system, thereby offering a mechanism for multiple parties to collaborate with minimal trust (any party can ensure the system's integrity by auditing history). Interest in distributed ledgers has soared recently, catalyzed by their use in cryptocurrencies (peer-to-peer payment systems) and by their potential as a foundation for new forms of financial systems, governance, and data sharing. In this work we study two limitations of ledgers, one about *privacy* and the other about *scalability*.

A privacy problem. The main strength of distributed ledgers is also their main weakness: *the history of all events is available for anyone to read*. This severely limits a direct application of distributed ledgers.

For example, in ledger-based payment systems such as Bitcoin [Nak09], every payment transaction reveals the payment's sender, receiver, and amount. This not only reveals private financial details of individuals and businesses using the system,[1] but also violates fungibility, a fundamental economic property of money. This lack of privacy becomes more severe in smart contract systems like Ethereum [Woo17], wherein transactions not only contain payment details, but also embed function calls to specific applications. In these systems, every application's internal state is necessarily public, and so is the history of function calls associated to it.

This problem has motivated prior work to find ways to achieve meaningful privacy guarantees on ledgers. For example, the Zerocash protocol [Ben+14] provides privacy-preserving payments, and Hawk [KMSWP16] enables general state transitions with data privacy, that is, an application's data is hidden from third parties.

However, all prior work is limited to hiding the inputs and outputs of a state transition but not *which* transition function is being executed. That is, prior work achieves *data privacy* but not *function privacy*. In systems with a single transition function this is not a concern.[2] In systems with multiple transition functions, however, this leakage is problematic. For example, Ethereum currently supports thousands of separate ERC-20 "token" contracts [Eth18], each representing a distinct currency on the Ethereum ledger; even if these contracts each individually adopted a protocol such as Zerocash to hide details about token payments, the corresponding transactions would still reveal *which* token was being exchanged. Moreover, the leakage of this information would substantially reduce the anonymity set of those payments.

A scalability problem. Public auditability in the aforementioned systems (and many others) is achieved via direct verification of state transitions that re-executes the associated computation. This creates the following scalability issues. First, note that in a network consisting of devices with

[1] Even if payments merely contain *addresses* rather than, say, social security numbers, much information about individuals and businesses can be gleaned by analyzing the flow of money over time between addresses [RH11; RS13; AKRSC13; Mei+13; SMZ14; Kal+20]. There are even companies that offer analytics services on the information stored on ledgers [Ell13; Cha14].

[2] For example, in Zerocash the single transition function is the one governing cash flow of a single currency.

heterogeneous computing power, requiring every node to re-execute transactions makes the weakest node a bottleneck, and this effect persists even when the underlying ledger is "perfect", that is, it confirms every valid transaction immediately. To counteract this and to discourage denial-of-service attacks whereby users send transactions that take a long time to validate, current systems introduce mechanisms such as *gas* to make users pay more for longer computations. However, such mechanisms can make it unprofitable to validate legitimate but expensive transactions, a problem known as the "Verifier's Dilemma" [LTKS15]. These problems have resulted in Bitcoin forks [Bit15] and Ethereum attacks [Eth16].

In sum, there is a dire need for techniques that facilitate the use of distributed ledgers for rich applications, without compromising privacy (of data or functions) or relying on unnecessary re-executions. Prior works only partially address this need, as discussed in Section 4.1.2 below.

4.1.1 Our contributions

We design, implement, and evaluate Zexe (*Zero knowledge EXEcution*), a ledger-based system that enables users to execute offline computations and subsequently produce publicly-verifiable transactions that attest to the correctness of these offline executions. Zexe simultaneously provides two main security properties.

- **Privacy:** *a transaction reveals no information about the offline computation, except (an upper bound on) the number of consumed inputs and created outputs.*[3] One cannot link together multiple transactions by the same user or involving related computations, nor selectively censor transactions based on such information.
- **Succinctness:** *a transaction can be validated in time that is independent of the cost of the offline computation whose correctness it attests to.* Since all transactions are indistinguishable, and are hence equally cheap to validate, there is no "Verifier's Dilemma", nor a need for mechanisms like Ethereum's gas.

Zexe also offers rich functionality, as offline computations in Zexe can be used to realize state transitions of multiple applications (such as tokens, elections, markets) simultaneously running atop the *same* ledger. The users participating in applications do not have to trust, or even know of, one another. Zexe supports this functionality by exposing a simple, yet powerful, *shared execution environment* with the following properties.

- **Extensibility:** users may execute arbitrary functions of their choice, without seeking anyone's permission.
- **Isolation:** functions of malicious users cannot interfere with the computations and data of honest users.

[3] One can fix the number of inputs and outputs (say, fix both to 2), or carefully consider side channels that could arise from revealing bounds on the number of inputs and outputs.

- **Inter-process communication:** functions may exchange data with one another.

DPC schemes. The technical core of Zexe is a protocol for a new cryptographic primitive for performing computations on a ledger called *decentralized private computation* (DPC). Informally, a DPC scheme supports a simple, yet expressive, programming model in which units of data, which we call *records*, are bound to scripts (arbitrary programs) that specify the conditions under which a record can be created and consumed (this model is similar to the UTXO model; see Remark 4.2.3). The rules that dictate how these programs interact can be viewed as a "nano-kernel" that provides a shared execution environment upon which to build applications. From a technical perspective, DPC can be viewed as extending Zerocash [Ben+14] to the foregoing programming model, while still providing strong privacy guarantees, not only within a single application (which is a straightforward extension) but also across multiple co-existing applications (which requires new ideas that we discuss later on). The security guarantees of DPC are captured via an ideal functionality, which our protocol provably achieves.

Applications. To illustrate the expressivity of the RNK, we show how to use DPC schemes to construct privacy-preserving analogues of popular applications: private user-defined assets, private decentralized or non-custodial exchanges (DEXs), and private stablecoins. Our privacy guarantees in particular protect against vulnerabilities of current DEX designs such as front-running [BDJT17; BBDJLZ17; EMC19; Dai+20]. Moreover, we sketch how to use DPC to construct a privacy-preserving smart contract system. See Sections 4.2.3 and 4.6 for details.

Techniques for efficient implementation. We devise a set of techniques to achieve an efficient implementation of our DPC protocol, by drawing upon recent advances in zero knowledge succinct cryptographic proofs (namely, zkSNARKs) and in recursive proof composition (proofs attesting to the validity of other proofs).

Overall, transactions in Zexe with two input records and two output records are 968 bytes and can be verified in tens of milliseconds, *regardless of the offline computation*; generating these transactions takes less than a minute plus a time that grows with the offline computation (inevitably so). This implementation is achieved in a modular fashion via a collection of Rust libraries (see Fig. 4.15), in which the top-level one is `libzexe`. Our implementation also supports transactions with *any* number m of input records and n of output records; transactions size in this case is $32m + 32n + 840$ bytes (the transaction stores the serial number of each input record and the commitment of each output record).

Delegating transactions. While verifying succinct cryptographic proofs is cheap, producing them can be expensive. As the offline computation grows, the (time and space) cost of producing a cryptographic proof of its correctness also grows, which could become infeasible for a user.

To address this problem, we further obtain *delegable DPC*. The user communicates to an untrusted worker details about the desired transaction, then the worker produces the transaction, and finally the user authorizes it via a cheap computation (and in a way that does not violate indistinguishability of transactions). This feature is particularly relevant for prospective real-world deployments, because it enables support for weak devices, such as mobile phones or hardware tokens.

In fact, our delegable DPC protocol also extends to support *threshold transactions*, which can be used to improve operational security, and also to support *blind transactions*, which can be used to realize lottery tickets for applications such as micropayments.

All of these extensions are also part of our Rust library `libzexe`.

A perspective on costs. ZEXE is not a lightweight construction, but achieves, in our opinion, tolerable efficiency for the ambitious goals it sets out to achieve: *data and function privacy, and succinctness, with rich functionality, in a threat model that requires security against all efficient adversaries*. Relaxing any of these goals (assuming rational adversaries or hardware enclaves, or compromising on privacy) will lead to more efficient approaches.

The primary cost in our system is, unsurprisingly, the cost of generating the cryptographic proofs that are included in transactions. We have managed to keep this cost to roughly a minute plus a cost that grows with the offline computation. For the applications mentioned above, these additional costs are negligible. Our system thus supports applications of real-world interest today (e.g., private DEXs) with reasonable costs.

4.1.2 Related work

Avoiding naive re-execution. A number of proposals for improving the scalability of smart contract systems, such as TrueBit [TR17], Plasma [PB17], and Arbitrum [KGCWF18], avoid naive re-execution by having users report the results of their computations *without* any cryptographic proofs, and instead putting in place incentive mechanisms wherein others can challenge reported results. The user and challenger engage in a so-called *refereed game* [FK97; CRR11; CRR13; JSST16; Rei16], mediated by a smart contract acting as the referee, that efficiently determines which of the two was "telling the truth". In contrast, in this work correctness of computation is ensured by cryptography, regardless of any economic motives; we thus protect against all efficient adversaries rather than merely all rational and efficient ones. Also, unlike our DPC scheme, the above works do not provide formal guarantees of strong privacy (challengers must be able to re-execute the computation leading to a result and in particular must know its potentially private inputs).

Private payments. Zerocash [Ben+14], building on earlier work [MGGR13], showed how to use distributed ledgers to achieve payment systems with strong privacy guarantees. The Zerocash protocol, with some modifications, is now commercially deployed in several cryptocurrencies, including Zcash [Zcash]. Solidus [CZJKJS17] enables customers of financial institutions (such as banks) to transfer funds to one another in a manner that ensures that only the banks of the sender and receiver learn the details of the transfer; all other parties (all other customers and banks) only learn that a transfer occurred, and nothing else. zkLedger [NVV18] enables anonymous payments between a small number of distinguished parties via the use of homomorphic commitments and Schnorr proofs. None of these protocols support scripts (small programs that dictate how funds can be spent), let alone arbitrary state transitions as in ZEXE.

Privacy beyond payments. Hawk [KMSWP16], combining ideas from Zerocash and the notion of an evaluator-prover for multi-party computation, enables parties to conduct offline computations and then report their results via cryptographic proofs. Hawk's privacy guarantee protects the private

inputs used in a computation, but does not hide *which* computation was performed. That said, we view Hawk as complementary to our work: a user in our system could in particular be a semi-trusted manager that administers a multi-party computation and generates a transaction about its output. The privacy guarantees provided in this work would then additionally hide *which* computation was carried out offline.

Zether [BAZB20] is a system that enables *publicly known* smart contracts to reason about homomorphic commitments in zero knowledge, and in particular enables these to transact in a manner that hides transaction amounts; it does not hide the identities of parties involved in the transaction, beyond a small anonymity set. Furthermore, the cost of verifying a transaction scales linearly with the size of the anonymity set, whereas in Zexe this cost scales logarithmically with the size of anonymity set.

Succinct blockchains. Coda [MS18] uses arbitrary-depth recursive composition of SNARKs to enable blockchain nodes to verify the current blockchain state quickly. In contrast, Zexe uses depth-2 recursive composition to ensure that all blockchain transactions are equally cheap to verify (and are moreover indistinguishable from each other), regardless of the cost of the offline computation. In this respect, Coda and Zexe address orthogonal scalability concerns.

MPC with ledgers. Several works [ADMM14b; ADMM14a; KMB15; KB16; BKM17; RGJKM17] have applied ledgers to obtain secure multi-party protocols that have security properties that are difficult to achieve otherwise, such as *fairness*. These approaches are complementary to our work, as any set of parties wishing to jointly compute a certain function via one of these protocols could run the protocol "under" our DPC scheme in such a way that third parties would not learn any information that such a multi-party computation is happening.

Hardware enclaves. Kaptchuk et al. [KGM19] and Ekiden [Zha+20] combine ledgers with hardware enclaves, such as Intel Software Guard Extensions [McK+13], to achieve various integrity and privacy goals for smart contracts. Beyond ledgers, several systems explore privacy goals in distributed systems by leveraging hardware enclaves; see for example M2R [DSCOZ15], VC3 [Sch+15], and Opaque [ZDBPGS17]. All of these works are able to efficiently support rich and complex computations. In this work, we make no use of hardware enclaves, and instead rely entirely on cryptography. This means that on the one hand our performance overheads are more severe, while on the other hand we protect against a richer class of adversaries (all efficient ones). Moreover, the techniques above depend on a working *remote attestation capability*; we note that our techniques can be used to achieve stronger security guarantees, even in the face of a compromise in the remote attestation capabilities of an enclave system (as recently occurred with Intel SGX [Van+19]).

4.2 Techniques

We now summarize the main ideas behind our contributions. Our goal is to design a ledger-based system in which transactions attest to offline computations while simultaneously providing *privacy* and *succinctness*.

We first note that if privacy is not required, there is a straightforward folklore approach that provides succinctness and low verification cost: each user accompanies the result reported in a transaction with a succinct cryptographic proof (i.e., a SNARK) attesting to the result's correctness. Others who validate the transaction can simply verify the cryptographic proof, and do not have to re-execute the computation. Even this limited approach rules out a number of cryptographic directions, such as the use of Bulletproofs [BCCGP16; BBBPWM18] (which have verification time linear in the circuit complexity), but can be accomplished using a number of efficient SNARK techniques [GGPR13; BCTV14; BCS16; BCTV17]. In light of this, we shall first discuss how to achieve privacy, and then how to additionally achieve succinctness.

The rest of this section is organized as follows. In Sections 4.2.1 and 4.2.2 we explain why achieving privacy in our setting is challenging. In Section 4.2.3 we introduce the shared execution environment that we consider, and in Section 4.2.4 we introduce *decentralized private computation* (DPC), a cryptographic primitive that securely realizes it. In Section 4.2.5 we describe how we turn our ideas into an efficient implementation.

4.2.1 Achieving privacy for a single arbitrary function

Zerocash [Ben+14] is a protocol that achieves privacy for a specific functionality, namely, *value transfers within a single currency*. Therefore, it is natural to consider what happens if we extend Zerocash from this special case to the general case of a *single arbitrary function* that is known in advance to everybody.

Sketch of Zerocash. Money in Zerocash is represented via *coins*. The commitment of a coin is published on the ledger when the coin is created, and its serial number is published when the coin is consumed. Each transaction on the ledger attests that some "old" coins were consumed in order to create some "new" coins: it contains the serial numbers of the consumed coins, commitments of the created coins, and a zero knowledge proof attesting that the serial numbers belong to coins created in the past (without identifying which ones), and that the commitments contain new coins of the same total value. A transaction is private because it only reveals how many coins were consumed and how many were created, but no other information (each coin's value and owner address remain hidden). Also, revealing a coin's serial number ensures that a coin cannot be consumed more than once (the same serial number would appear twice). In sum, data in Zerocash corresponds to coin values, and state transitions are the single invariant that monetary value is preserved.

Extending to an arbitrary function. One way to extend Zerocash to a single arbitrary function Φ (known in advance to everybody) is to think of a coin as a *record* that stores some arbitrary data *payload*, rather than just some integer value. The commitment of a record would then be published on the ledger when the record is created, and its unique serial number would be published when the record is consumed. A transaction would then contain serial numbers of consumed records,

commitments of created records, and a proof attesting that invoking the function Φ on (the payload of) the old records produces (the payload of) the new records.

Data privacy holds because the ledger merely stores each record's commitment (and its serial number once consumed), and transactions only reveal that some number of old records were consumed in order to create some number of new records in a way that is consistent with Φ. Function privacy also holds but for trivial reasons: Φ is known in advance to everybody, and every transaction is about computations of Φ.

Note that Zerocash is indeed a special case of the above: it corresponds to fixing Φ to the particular (and publicly known) choice of a function $\Phi_{\$}$ that governs value transfers within a single currency. However the foregoing protocol supports only a single hard-coded function Φ, while instead we want to enable users to select their own functions, as we discuss next.

4.2.2 Difficulties with achieving privacy for user-defined functions

We want to enable users to execute functions of their choice concurrently on the same ledger without seeking permission from anyone. That is, when preparing a transaction, a user should be able to pick *any* function Φ of their choice for creating new records by consuming some old records. If function privacy is not a concern, then this is easy: just attach to the transaction a zero-knowledge proof that Φ was correctly evaluated offline. However, because this approach reveals Φ, we cannot use it because function privacy is a goal for us.

An approach that *does* achieve function privacy would be to modify the sketch in Section 4.2.1 by fixing a single function that is *universal*, and then interpreting data payloads as user-defined functions that are provided as inputs. Indeed, zero knowledge would ensure function privacy in this case. However merely allowing users to define their own functions does *not* by itself yield meaningful functionality, as we explain next.

The problem: malicious functions. A key challenge in this setting is that malicious users could devise functions to attack or disrupt other users' functions and data, so that a particular user would not know whether to trust records created by other users; indeed, due to function privacy, a verifier would not know what functions were used to create those records. For a concrete example, suppose that we wanted to realize the special case of value transfers within a single currency (i.e., Zerocash). One may believe that it would suffice to instruct users to pick the function $\Phi_{\$}$ (or similar). But this does *not* work: a user receiving a record claiming to contain, say, 1 unit of currency does not know if this record was created via the function $\Phi_{\$}$ from other such records and so on. A malicious user could have used a different function to create that record, for example, one that illegally "mints" records that appear valid to $\Phi_{\$}$, and thus enables arbitrary inflation of the currency. More generally, the lack of any enforced rules about how user-defined functions can interact precludes productive cooperation between users that are mutually distrustful. We stress that this challenge arises specifically due to the requirement that functions be private: if the function that created (the commitment of) a record was public knowledge, users could decide for themselves if records they receive were generated by "good" functions.

One way to address the foregoing problem is to augment records with a new attribute that identifies

the function that "created" the record, and then impose the restriction that in a valid transaction only records created by the same function may participate. This new attribute is contained within a hiding commitment and thus is never revealed publicly on the ledger (just like a record's payload); the zero knowledge proof is tasked with ensuring that records participating in the same transaction all have the same "type". This approach now *does* suffice to realize value transfers within a single currency, by letting users select the function $\Phi_{\$}$. More generally, this approach generalizes that in Section 4.2.1, and can be viewed as running multiple segregated "virtual ledgers" each with a fixed function. Function privacy holds because one cannot tell if a transaction belongs to one virtual ledger or another.

The problem: functions cannot communicate. The limitation of the above technique is that it forbids any "inter-process communication" between different functions, and so one cannot realize even simple functionalities like transferring value between different currencies on the same ledger. It also rules out more complex smart contract systems, as communication between contracts is a key part of such systems. It is thus clear that this crude "time sharing" of the ledger is too limiting.

4.2.3 The records nano-kernel: a minimalist shared execution environment

The approaches in Section 4.2.2 lie at opposite extremes: unrestricted inter-process interaction prevents the secure construction of even basic applications such as a single currency, while complete process segregation limits the ability to construct complex applications that interact with each other.

Balancing these extremes requires a shared execution environment: one can think of this as an *operating system* for a shared ledger. This operating system manages user-defined functions: it provides process isolation, determines data ownership, handles inter-process communication, and so on. Overall, processes must be able to concurrently share a ledger, without violating the integrity or confidentiality of one another.

However, function privacy (one of our goals) dictates that user-defined functions are hidden, which means that an operating system cannot be maintained publicly atop the ledger (as in current smart contract systems) but, instead, must be part of the statement proved in zero knowledge. This is unfortunate because designing an operating system that governs interactions across user-defined functions within a zero knowledge proof is not only a colossal design challenge but also entails many arbitrary design choices that we should not have to take.

In light of the above, we choose to take the following approach: we formulate a *minimalist* shared execution environment that imposes simple, yet expressive, rules on how records may interact, and enables programming applications in the UTXO model (see Remark 4.2.3 for why we make this choice). This execution environment can be viewed as a "nano-kernel" that enables users to manage records containing data by programming two boolean functions (or predicates) associated with each record. These predicates control the two defining moments in a record's life, namely creation (or "birth") and consumption (or "death"), and are hence called the record's *birth* and *death* predicates. A user can create and consume records in a transaction by satisfying the predicates of those records. In more detail,

> **The records nano-kernel (RNK)** is an execution environment that operates over units of data called records. A record contains a *data payload*, a *birth predicate* Φ_{b}, and a *death predicate* Φ_{d}. Records are created and consumed by *valid transactions*. These are transactions where the death predicates of all consumed records and the birth predicates of all created records are simultaneously satisfied when given as input the transaction's *local data* (see Fig. 4.3), which includes: (a) every record's contents (such as its payload and the identity of its predicates); (b) a piece of shared memory that is publicly revealed, called *transaction memorandum*; (c) a piece of shared memory that is kept hidden, called *auxiliary input*; and (d) other construction specifics.

The foregoing definition enables predicates to see the contents of the entire transaction and hence *to individually decide if the local data is valid according to its own logic*. This in turn enables predicates to communicate with each other securely without interference from malicious predicates. In more detail, a record $\mathbf{r}$ can protect itself from other records that contain "bad" birth or death predicates because the $\mathbf{r}$'s predicates could refuse to accept when they detect (from reading the local data) that they are in a transaction with records having bad predicates. At the same time, a record can interact with other records in the same transaction when its predicates decide to accept, providing the flexibility that we seek.

We briefly illustrate this via an example, *user-defined assets*, whereby one can use birth predicates to define and transact with their own assets, and also use death predicates to enforce custom access control policies over these assets.

Example 4.2.1 (user-defined assets)**.** Consider records whose payloads encode an asset identifier id, the initial asset supply $\mathbb{v}$, and a value v. Fix the birth predicate in all such records to be a *mint-or-conserve* function MoC that is responsible for creating the initial supply of a new asset, and then subsequently conserving the value of the asset across all transactions. In more detail, MoC can be invoked in one of two modes. In *mint mode*, given as input a desired initial supply $\mathbb{v}$, MoC deterministically derives (in a way that we discuss later) a fresh unique identifier id for a new asset and stores $(\mathsf{id}, \mathbb{v}, v = \mathbb{v})$ in a *genesis record*. In *conserve mode*, MoC inspects all records in a transaction whose birth predicates equal to MoC and whose asset identifiers equal the identifier of the current record, and ensures that among these records, the asset values are conserved.

Users can program death predicates of records to enforce conditions on how assets can be consumed, e.g., by realizing *conditional exchanges* with other counter-parties. Suppose that Alice wishes to exchange 100 units of an asset id_1 for 50 units of another asset id_2, but does not have a counter-party for the exchange. She creates a record $\mathbf{r}$ with 100 units of id_1 whose death predicate enforces that any transaction consuming $\mathbf{r}$ must also create another record, consumable by Alice, with 50 units of id_2. She then publishes out of band information about $\mathbf{r}$, and anyone can subsequently claim it by creating a transaction doing the exchange.

Since death predicates can be *arbitrary*, many different access policies can also be realized, e.g., to enforce that a transaction redeeming a record (a) must be authorized by two of three public keys, or (b) becomes valid only after a given amount of time, or (c) must reveal the pre-image of a hash function.

One can generalize this basic example to show how the RNK can realize a specific class of smart contract systems, namely those in which the transaction creator knows both the contract code being executed, as well as the (public and secret) state of the contract. At a high level, these contracts can be executed within a single transaction, or across multiple transactions, by storing suitable intermediate state/message data in record payloads, or by publishing that data in transaction memoranda (as plaintext or ciphertext as needed). We discuss in more detail below.

Example 4.2.2 (smart contracts with caller-known state)**.** At the highest level, smart contract systems operate over a set of individual contracts, each of which consists of a function (or collection of functions), some state variables, and some form of *address* that serves to uniquely identify the contract. The contract address ensures that the same code/functions can be deployed multiple times by different individuals, without two contracts inadvertently sharing state.[4] A standard feature of smart contract systems is that a contract can communicate with other contracts: that is, a contract can invoke a second smart contract as a subroutine, provided that the second contract provides an interface to allow this behavior. In our setting, we consider contracts in which the caller knows at least part of the state of each contract.

In this setting, one can use the records nano-kernel to realize basic smart contracts as follows. Each contract can be implemented as a function Φ_{sc}. The contract's state variables can be stored in one or more records such that each record $\mathbf{r}_i$ is labeled with Φ_{sc} as the birth and death predicate. Using this labeling, Φ_{sc} (via the RNK) can enforce that only it can update its state variables, thus fulfilling one requirement of a secure contract. Of course, while this serves to prevent *other functions* from updating the contract's state, it does not address the situation where multiple users wish to deploy different instances of the same function Φ_{sc}, each with isolated state. Fortunately (and validating our argument that the RNK realizes the *minimal* requirements needed for such a system), addressing this problem does not require changes to the RNK. Instead, one can devise the function Φ_{sc} so that it reasons over a unique contract address identifier id, which is recorded within the *payload* of every record.[5] The function Φ_{sc} can achieve contract state isolation by enforcing that each input and output state record considered by single execution of Φ_{sc} shares the same contract address.

To realize "inter-contract calls" between two functions Φ_{sc_1} and Φ_{sc_2}, one can use "ephemeral" records that communicate between the two functions. For example, if Φ_{sc_1} wishes to call Φ_{sc_2}, the caller may construct a record $\mathbf{r}_{\mathsf{e}}$ that contains the "arguments" to the called function Φ_{sc_2}, as well as the result of the function call. A transaction would then show that both Φ_{sc_1} and Φ_{sc_2} are satisfied.

The above example outlines how to implement a general smart contract system atop the RNK. We leave to future work the task of developing this outline into a full-fledged smart contract framework, and instead focus on constructing a scheme that implements the RNK, and on illustrating how to directly program the RNK to construct specific applications such as **user-defined assets**, **private**

[4] In concrete implementations such as Ethereum [Woo17], contract identification is accomplished through unique contract addresses, each of which can be bound to a possibly non-unique `codeHash` that identifies the code of the program implementing the contract.

[5] This identifier can be generated in a manner similar to the asset identifier in Example 4.2.1.

decentralized asset exchanges, and **regulation-friendly private stablecoins**. We discuss these applications in more detail in Section 4.6.

Remark 4.2.3 (working in the UTXO model)**.** In the records nano-kernel, applications update their state by consuming records containing the old state, and producing new records that contain the updated state. This programming model is popularly known as the "unspent transaction output" (UTXO) model. This is in contrast to the "account-based" model which is used by many other smart contract systems [Goo14; Woo17; Yak18]. At present, it is not known how to efficiently achieve strong privacy properties in this model even for the simple case of privacy-preserving payments among any number of users, as we explain below.

In the account-based model, application state is stored in a persistent location associated with the application's account, and updates to this state are applied in-place. A smart contract that implements a currency in this model would store user balances in a persistent table T that maps user account identifiers to user balances. Transactions from a user A to another user B would then decrement A's balance in T and increment B's balance by a corresponding amount. A straightforward way to make this contract data-private (i.e., to hide the transaction value and the identities of A and B) would be to replace the user balances in T with *hiding commitments* to these balances; transactions would then update these commitments instead of directly updating the balances. However, while this hides transaction values, it does not hide user identities; to further hide these, every transaction would have to update *all* commitments in T, which entails a cost that grows linearly with the number of users. This approach is taken by zkLedger [NVV18], which enables private payments between a small number of known users (among other things).

Even worse, achieving function privacy when running multiple applications in such a system would require each transaction to hide which application's data was being updated, which means that the transaction would have to update the data of *all* applications at once, again severely harming the efficiency of the system.

In sum, it is unclear how to efficiently achieve strong data and function privacy in the account-based model when users can freely join and leave the system without notifying other users. On the other hand, we show in this work that these properties can be achieved in the UTXO model at a modest cost.

4.2.4 Decentralized private computation

A new cryptographic primitive. We introduce a new cryptographic primitive called *decentralized private computation* (DPC) schemes, which capture the notion of a ledger-based system where privacy-preserving transactions attest to offline computations that follow the records nano-kernel. See Section 4.3 for the definition of DPC schemes, including the ideal functionality that we use to express security.

We construct a DPC scheme in Section 4.4, and prove it secure in Section 4.10. We take Zerocash [Ben+14] as a starting point, and then extend the protocol to support the records nano-kernel and also to facilitate proving security in the simulation paradigm relative to an ideal functionality (rather

than via a collection of separate game-based definitions as in [Ben+14]). Below we sketch the construction.

Construction sketch. Each transaction in the ledger consumes some old records and creates new records in a manner that is consistent with the records nano-kernel. To ensure privacy, a transaction only contains serial numbers of the consumed records, commitments of the created records, and a zero knowledge proof attesting that there exist records consistent with this information (and with the records nano-kernel). All commitments on the ledger are collected in a Merkle tree, which facilitates efficiently proving that a commitment appears on the ledger (by proving in zero knowledge the knowledge of a suitable authentication path). All serial numbers on the ledger are collected in a list that cannot contain duplicates. This implies that a record cannot be consumed twice because the same serial number is revealed each time a record is consumed. See Fig. 4.1.

The record data structure is summarized in Fig. 4.2. Each record is associated to an *address public key*, which is a commitment to a seed for a pseudorandom function acting as the corresponding *address secret key*; addresses determine ownership of records, and in particular consuming a record requires knowing its secret key. A *record* consists of an address public key, a data payload, a birth predicate, a death predicate, and a serial number nonce; a *record commitment* is a commitment to all of these attributes. The *serial number* of a record is the evaluation of a pseudorandom function, whose seed is the secret key for the record's address public key, evaluated at the record's serial number nonce. A record's commitment and serial number, which appear on the ledger when the record is created and consumed, reveal *no* information about the record attributes. This follows from the hiding properties of the commitment, and the pseudorandom properties of the serial number. The derivation of a record's serial number ensures that a user can create a record for another in such a way that its serial number is fully determined and yet cannot be predicted without knowing the other user's secret key.

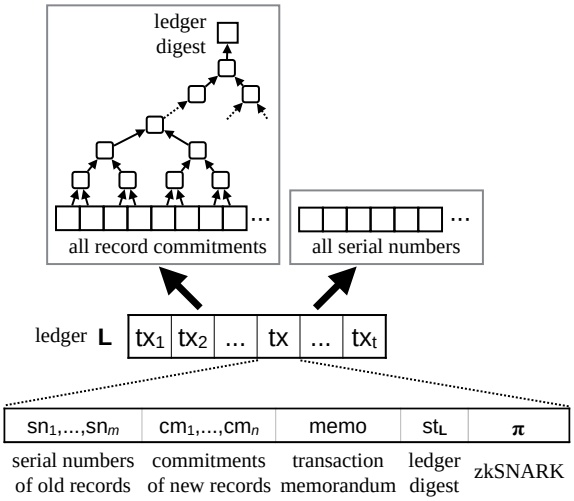

Figure 4.1: Construction of a transaction.

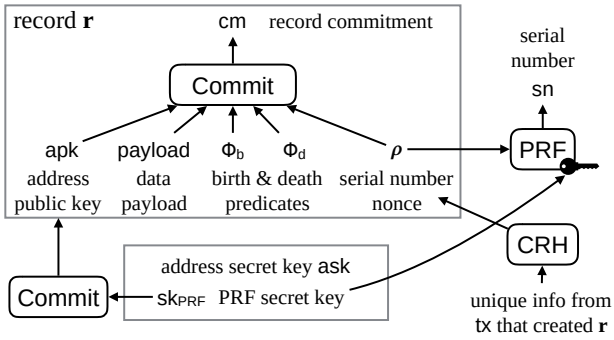

Figure 4.2: Construction of a record.

In order to produce a transaction, a user selects some previously-created records to consume, assembles some new records to create (including their payloads and predicates), and decides on other

aspects of the local data such as the transaction memorandum (shared memory seen by all predicates and published on the ledger) and the auxiliary input (shared memory seen by all predicates but not published on the ledger); see Fig. 4.3. If the user knows the secret keys of the records to consume and if all relevant predicates are satisfied (death predicates of old records and birth predicates of new predicates), then the user can produce a zero knowledge proof to append to the transaction. See Fig. 4.4 for a summary of the NP statement being proved.

In sum, a transaction only reveals the number of consumed records and number of created records, as well as any data that was deliberately revealed in the transaction memorandum (possibly nothing).[6]

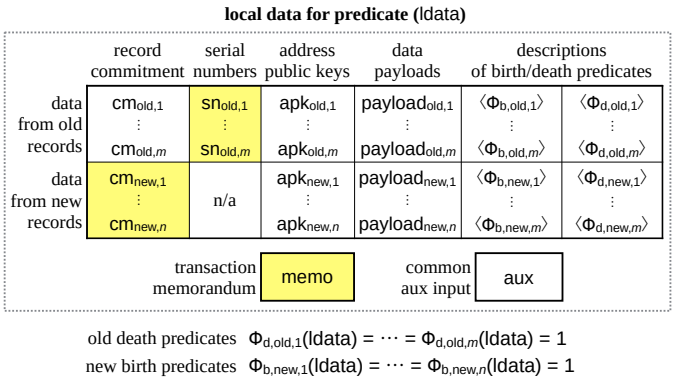

Figure 4.3: Predicates receive local data.

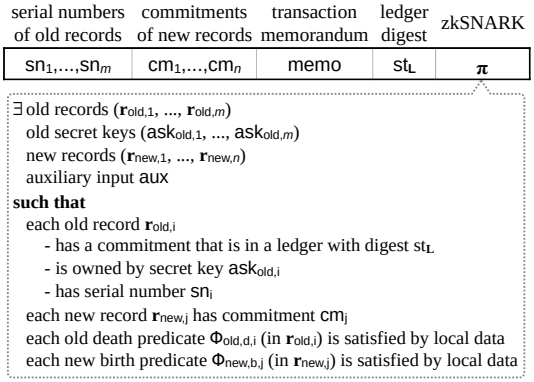

Figure 4.4: The execute statement.

Achieving succinctness. Our discussions so far have focused on achieving (data and function) privacy. However, we also want to achieve succinctness, namely, that a transaction can be validated in "constant time". This follows from a straightforward modification: we take the protocol that we have designed so far and use a zero knowledge *succinct* argument rather than just any zero knowledge proof. Indeed, the NP statement being proved (summarized in Fig. 4.4) involves attesting the satisfiability of all (old) death and (new) birth predicates, and thus we need to ensure that verifying the corresponding proof can be done in time that does not depend on the complexity of these predicates. While turning this idea into an efficient implementation requires more ideas (as we discuss in Section 4.2.5), the foregoing modification suffices from a theoretical point of view.

Delegation to an untrusted worker. In our DPC scheme, a user must produce, and include in the transaction, a zero knowledge succinct argument that, among other things, attests that death predicates of consumed records are satisfied and, similarly, that birth predicates of created records are satisfied. This implies that the cost of creating a transaction grows with the complexity (and number of) predicates involved in the transaction. Such a cost can quickly become infeasible for weak devices such as mobile phones or hardware tokens.

We address this problem by enabling a user to *delegate* to an untrusted worker, such as a remote server, the computation that produces a transaction. This notion, which we call a *delegable DPC*

[6] By supporting the use of dummy records, we can in fact ensure that only *upper bounds* on the foregoing numbers are revealed.

scheme, empowers weak devices to produce transactions that they otherwise could not have produced on their own.

The basic idea is to augment address keys in such a way that the secret information needed to produce the cryptographic proof is separate from the secret information needed to authorize a transaction containing that proof. Thus, the user can communicate to the worker the secrets necessary to generate a cryptographic proof, while retaining the remaining secrets for authorizing this (and future) transactions. In particular, the worker has no way to produce valid transactions that have not been authorized by the user.

We use randomizable signatures to achieve the foregoing functionality, without violating either privacy or succinctness. Informally, we modify a record's serial number to be an unlinkable randomization of (part of) the record's address public key, and a user's authorization of a transaction consists of signing the instance and proof relative to every randomized key (i.e., serial number) in that transaction. See Section 4.5 for details.

4.2.5 Achieving an efficient implementation

Our system ZEXE (*Zero knowledge EXEcution*) provides an implementation of two constructions: our "plain" DPC protocol, and its extension to a delegable DPC protocol. Achieving efficiency in our system required overcoming several challenges. Below we highlight some of these challenges, and explain how we addressed them; see Sections 4.7 and 4.8 for details. The discussions below equally apply to both types of DPC protocols.

Avoiding the cost of universality. The NP statement that we need to prove involves checking user-defined predicates, so it must support arbitrary computations that are not fixed in advance. However, state-of-the-art zkSNARKs for universal computations rely on expensive tools [BCGTV13; BCTV14; WSRBW15; BCTV17].

We address this problem by relying on one layer of *recursive proof composition* [Val08; BCCT13]. Instead of tasking the NP statement with directly checking user-defined predicates, we only task it with checking *succinct proofs* attesting to this. Checking these *inner* succinct proofs is a (relatively) inexpensive computation that is fixed for *all* predicates, which can be "hardcoded" in the statement. Since the single *outer* succinct proof produced does not reveal information about the inner succinct proofs attesting to predicates' satisfiability (thanks to zero knowledge), the inner succinct proofs do *not* have to hide what predicate was checked, so they can be for NP statements *tailored* to the computations of particular user-defined predicates.

A bespoke recursion. Recursive proof composition has been empirically demonstrated for pairing-based SNARKs [BCTV17]. We thus focus our attention on these, and explain the challenges that arise in our setting. Recall that if we instantiate a SNARK's pairing via an elliptic curve E defined over a prime field $\mathbb{F}_q$ and having a subgroup of prime order r, then (a) the SNARK supports NP statements expressed as arithmetic circuits over $\mathbb{F}_r$, while (b) proof verification involves arithmetic operations over $\mathbb{F}_q$. Being part of the NP statement, the SNARK verifier must also be expressed as an arithmetic circuit over $\mathbb{F}_r$, which is problematic because the verifier's "native" operations are over $\mathbb{F}_q$. Simulating $\mathbb{F}_q$ operations via $\mathbb{F}_r$ operations is expensive, and picking E such that

$q = r$ is impossible [BCTV17]. Prior work thus uses *multiple* curves [BCTV17]: a two-cycle of pairing-friendly elliptic curves, that is, two prime-order curves E_1 and E_2 such that the prime size of one's base field is the prime order of the other's group, and orchestrating SNARKs based on these so that fields "match up". However, known cycles are inefficient at 128 bits of security [BCTV17; CCW19].

We address this problem by noting that we merely need "a proof of a proof", and thus, instead of relying on a cycle, we can use the Cocks–Pinch method [FST10] to set up a bounded recursion [BCTV17]. First we pick a pairing-friendly elliptic curve that not only is suitable for 128 bits of security according to standard considerations but, moreover, is compatible with efficient SNARK provers in *both* levels of the recursion. Namely, letting p be the prime order of the base field and r the prime order of the group, we need that *both* $\mathbb{F}_r$ and $\mathbb{F}_p$ have multiplicative subgroups whose orders are large powers of 2. The condition on $\mathbb{F}_r$ ensures efficient proving for SNARKs over this curve, while the condition on $\mathbb{F}_p$ ensures efficient proving for SNARKs that verify proofs over this curve. In light of the above, we select a curve E_{BLS} from the Barreto–Lynn–Scott (BLS) family [BLS02; CLN11] with embedding degree 12. This family not only enables parameters that conservatively achieve 128 bits of security, but also enjoys properties that facilitate very efficient implementation [AFKMR12]. We ensure that both $\mathbb{F}_r$ and $\mathbb{F}_p$ have multiplicative subgroups of order 2^α for $\alpha \geq 40$, by a suitable condition on the parameter of the BLS family.

Next we use the Cocks–Pinch method to pick a pairing-friendly elliptic curve E_{CP} over a field $\mathbb{F}_q$ such that the curve group $E_{\mathsf{CP}}(\mathbb{F}_q)$ contains a subgroup of prime order p (the size of E_{BLS}'s base field). Since the method outputs a prime q that has about $2\times$ more bits than the desired p, and in turn p has about $1.5\times$ more bits than r (due to properties of the BLS family), we only need E_{CP} to have embedding degree of 6 in order to achieve 128 bits of security (as determined from the guidelines in [FST10]).

In sum, a SNARK over E_{BLS} is used to generate proofs of predicates' satisfiability; after that a zkSNARK over E_{CP} is used to generate proofs that these prior proofs are valid along with the remaining NP statement's checks. The matching fields between the two curves ensure that the former proofs can be efficiently verified.

Minimizing operations over E_{CP}. While the curve E_{CP} facilitates efficient checking of SNARK proofs over E_{BLS}, operations on it are at least $2\times$ more costly (in time and space) than operations over E_{BLS}, simply because E_{CP}'s base field is twice the size of E_{BLS}'s base field. This makes checks in the NP relation $\mathcal{R}_{\mathsf{e}}$ that are not related to proof checking unnecessarily expensive.

To avoid this, we split $\mathcal{R}_{\mathsf{e}}$ into two NP relations, $\mathcal{R}_{\mathsf{BLS}}$ and $\mathcal{R}_{\mathsf{CP}}$. The latter is responsible only for verifying proofs of predicates' satisfaction, while the former is responsible for all other checks. We minimize the number of E_{CP} operations by proving satisfaction of $\mathcal{R}_{\mathsf{BLS}}$ and $\mathcal{R}_{\mathsf{CP}}$ with zkSNARKs over E_{BLS} and E_{CP} respectively. A transaction now includes both proofs.

Optimizing the NP statement. We note that the remaining NP statement's checks can themselves be quite expensive, as they range from verifying authentication paths in a Merkle tree to verifying commitment openings, and from evaluating pseudorandom functions to evaluating collision resistant functions. Prior work realizing similar collections of checks required upwards of *four million gates* [Ben+14] to express such checks. This not only resulted in high latencies for producing transactions

(several minutes) but also resulted in large public parameters for the system (hundreds of megabytes).

Commitments and collision-resistant hashing can be expressed as very efficient arithmetic circuits if one opts for Pedersen-type constructions over suitable Edwards elliptic curves (and techniques derived from these ideas are now part of deployed systems [HBHW20]). To achieve this, we pick two Edwards curves, $E_{\mathsf{Ed/BLS}}$ over the field $\mathbb{F}_r$ (thereby matching the group order of E_{BLS}), and $E_{\mathsf{Ed/CP}}$ over the field $\mathbb{F}_p$ (thereby matching the group order of E_{CP}). This allows to realize very efficient circuits for various primitives used in our NP relations, including commitments, collision-resistant hashing, and randomizable signatures. Overall, we obtain highly optimized realizations of all checks in Fig. 4.4.

4.2.6 Deployment considerations

DPC schemes include a setup algorithm that specifies how to sample public parameters, which are used to produce transactions and to verify transactions. The setup algorithm in our DPC construction (see Section 4.4) simply consists of running the setup algorithms for the various cryptographic building blocks that we rely on: commitment schemes, collision-resistant hash functions, and zero knowledge proofs.

In practice, deploying cryptography that relies on setup algorithms (such as DPC schemes) can be challenging because the entity running the setup algorithm may be able to break certain security properties of the scheme, by abusing knowledge of the randomness used to produce the public parameters. On the other hand, *some* setup algorithm is typically inevitable. For example, non-interactive zero knowledge proofs without any setup exist only for languages decidable in polynomial time [GO94]. Nevertheless, one could still aim for a *transparent setup*, one that consists of public randomness, because in practice it is cheaper to realize.

Our construction of a DPC scheme has a transparent setup algorithm whenever the setup algorithms for the underlying cryptographic building blocks also have transparent setups. For example, this would hold if we instantiated our construction via Pedersen commitments, Pedersen hash functions, and transparent zkSNARKs (as obtained from probabilistic checking tools in the random oracle model [Mic00; BCS16]).

However, due to efficiency considerations described in Section 4.2.5, our implemented system relies on pairing-based zkSNARKs whose setup is *not* transparent. (We use the simulation-extractable zkSNARK of Groth and Maller [GM17].) We should thus discuss how one may deploy our implemented system, and in particular the effects of compromise in the trusted setup phase of these SNARKs. (All other primitives in our system use a transparent setup.)

Recall that prior zkSNARK deployments have used secure multiparty computation [BCGTV15; ZcashCmny; BGM17; BGG18; KMSV21], so that the sampled public parameters are guaranteed to be secure as long as even a single participating party is honest. One could leverage these same ideas to sample "master" parameters for proving/verifying the two NP relations $\mathcal{R}_{\mathsf{BLS}}$ and $\mathcal{R}_{\mathsf{CP}}$ (over the two elliptic curves E_{BLS} and E_{CP}) mentioned in Section 4.2.5. Note that these public parameters do *not* depend on any user-defined functions (birth or death predicates), and can thus be sampled once and for all regardless of which applications will be run over the system. Note also that these public

parameters must be trusted by *everyone*, because if they were compromised then the security (but not privacy) of *all* applications running over the system would be compromised as well.

The foregoing public parameters are not the only ones that need to be sampled in order to use our implemented system. Every (birth or death) predicate requires its own public parameters, because (the verification key contained in) these public parameters is part of the record that contains it, and is ultimately used to recursively check a proof of the predicate's satisfiability. Since an application relies only on the public parameters of certain predicates, we call such parameters as "application" parameters.

Unlike "master" parameters, "application" parameters do not have to be sampled at the start of the system's lifetime, and also do not have to be trusted by every user in the system. Indeed, interactions across records are overseen by the NP relations $\mathcal{R}_{\mathsf{BLS}}$ and $\mathcal{R}_{\mathsf{CP}}$ (which rely on the "master" parameters) and thus compromised parameters for one application will not affect (the security and privacy of) an application that does not rely on them. This means that a user only needs to trust the parameters that are relied upon by the applications that the user cares about. In turn this means that the sampling of application parameters can be viewed as an organic process, which occurs as applications are developed and deployed, and each application can be in charge of deciding whichever method is most suitable for securely sampling its own parameters.

Subsequent works [MBKM19; CFQ19; CHMMVW20; GWC19] have proposed pairing-based SNARKs that have a universal setup that can be used for *any* circuit. Once such SNARK constructions mature into efficient implementations, our system can be easily modified to use these instead of [GM17] to mitigate the above concerns, as both our construction and implementation make use of the underlying SNARKs in a modular and black-box manner.

4.3 Definition of decentralized private computation schemes

We define *decentralized private computation* (DPC) schemes, a cryptographic primitive in which parties with access to an ideal append-only ledger execute computations offline and subsequently post privacy-preserving, publicly-verifiable transactions that attest to the correctness of these offline executions. This primitive generalizes prior notions [Ben+14] that were limited to proving correctness of simple financial invariants.

Below we introduce the data structures, interface, and security requirements for a DPC scheme: Section 4.3.1 describes the main data structures of a DPC scheme, Section 4.3.2 defines the syntax of the DPC algorithms, and finally in Section 4.3.3 we describe the security requirements for DPC schemes via an ideal functionality. We note that our definition of DPC schemes focuses on (correctness and) privacy, because we leave succinctness as a separate efficiency goal that easily follows from suitable building blocks (see Remark 4.4.1).

4.3.1 Data structures

In a DPC scheme there are three main data structures: *records*, *transactions*, and the *ledger*.

Records. A *record*, denoted by the symbol $\mathbf{r}$, is a data structure representing a unit of data. Records can be created or consumed, and these events denote state changes in the system. For example, in a currency application, records store units of the currency, and state changes represent the flow of units in that currency.

In more detail, a record $\mathbf{r}$ has the following attributes (see Fig. 4.5): (a) a *commitment* cm, which binds together all other attributes of $\mathbf{r}$ while hiding all information about them; (b) an *address public key* apk, which specifies the record's owner; (c) a *payload* $\mathsf{payload}$ containing arbitrary application-dependent information; (d) a *birth predicate* Φ_{b} that must be satisfied when $\mathbf{r}$ is created; (e) a *death predicate* Φ_{d} that must be satisfied when $\mathbf{r}$ is consumed; and (f) other construction-specific information. Both Φ_{b} and Φ_{d} are arbitrary non-deterministic boolean-valued functions. The payload $\mathsf{payload}$ contains a designated subfield $\mathsf{isDummy}$ which denotes whether $\mathbf{r}$ is dummy or not.

Informally, the "life" of a (non-dummy) record $\mathbf{r}$ is marked by two events: *birth* and *death*. The record $\mathbf{r}$ is *born* (or is *created*) when its commitment cm is posted to the ledger as part of a transaction. Then the record $\mathbf{r}$ *dies* (or is *consumed*) when its serial number sn appears on the ledger as part of a later transaction. At each of these times (birth or death) the corresponding predicate (Φ_{b} or Φ_{d}) must be satisfied. Dummy records, on the other hand, can be created freely, but consuming them requires satisfaction of their death predicates. The purpose of dummy records is solely to enable the creation of new non-dummy records.

To consume $\mathbf{r}$, one must also know the address secret key ask corresponding to $\mathbf{r}$'s address public key apk because the serial number sn to be revealed can only be computed from $\mathbf{r}$ and ask. The ledger forbids the same serial number to appear more than once, so that: (a) a record cannot be consumed twice because it is associated to exactly one serial number; (b) others cannot prevent one from consuming a record because it is computationally infeasible to create two distinct records that share the same serial number sn but have distinct commitments cm and cm'.

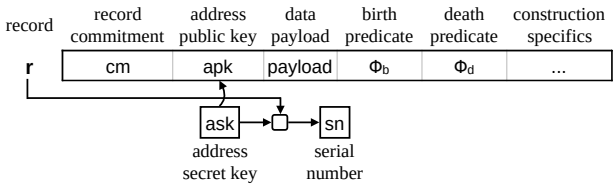

Figure 4.5: Diagram of a record.

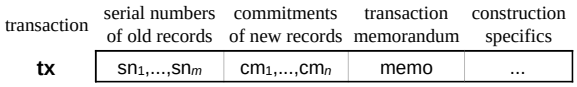

Figure 4.6: Diagram of a transaction.

Transactions. A transaction, denoted by the symbol tx, is a data structure representing a state change that involves the consumption and creation of records (see Fig. 4.6). It is a tuple $([\mathsf{sn}_i]_1^m, [\mathsf{cm}_j]_1^n, \mathsf{memo}, \star)$ where (a) $[\mathsf{sn}_i]_1^m$ is the list of serial numbers of the m old records, (b) $[\mathsf{cm}_j]_1^n$ is the list of commitments of the n new records, (c) memo is an arbitrary string associated with the transaction, and (d) $\star$ is other construction-specific information. The transaction tx reveals only the following information about old and new records: (i) the old records' serial numbers; (ii) the new records' commitments; and (iii) the fact that the death predicates of all consumed records and birth predicates of all new records were satisfied.

Anyone can assemble a transaction and append it to the ledger, provided that it is "valid" in the sense that (all records are well-formed and) the death predicates of any consumed records and the birth predicates of any created records are satisfied. Note that all transactions reveal the number of old records (m) and the number of new records (n), but not how many of these were dummy or not.

Ledger. We consider a model where all parties have access to an append-only ledger, denoted $\mathbf{L}$, that stores all published transactions. Our definitions (and constructions) are *agnostic* to how this ledger is realized (e.g., the ledger may be centrally managed or a distributed protocol). When an algorithm needs to interact with the ledger, we specify $\mathbf{L}$ in the algorithm's superscript. The ledger exposes the following interface.

- $\mathbf{L}.\mathsf{Len}$: Return the number of transactions currently on the ledger.
- $\mathbf{L}.\mathsf{Push}(\mathsf{tx})$: Append a (valid) transaction tx to the ledger.
- $\mathbf{L}.\mathsf{Digest} \rightarrow \mathsf{st}_{\mathbf{L}}$: Return a (short) digest of the current state of the ledger.
- $\mathbf{L}.\mathsf{ValidateDigest}(\mathsf{st}_{\mathbf{L}}) \rightarrow b$: Check that $\mathsf{st}_{\mathbf{L}}$ is a valid digest for some (past) ledger state.
- $\mathbf{L}.\mathsf{Contains}(\mathsf{tx}) \rightarrow b$: Determine if tx (or a subcomponent thereof) appears on the ledger or not.
- $\mathbf{L}.\mathsf{Prove}(\mathsf{tx}) \rightarrow \mathbb{w}_{\mathbf{L}}$: If a transaction tx (or a subcomponent thereof) appears on the ledger, return a proof of membership $\mathbb{w}_{\mathbf{L}}$ for it. If there are duplicates, return a proof for the lexicographically first one.
- $\mathbf{L}.\mathsf{Verify}(\mathsf{st}_{\mathbf{L}}, \mathsf{tx}, \mathbb{w}_{\mathbf{L}}) \rightarrow b$: Check that $\mathbb{w}_{\mathbf{L}}$ certifies that tx (or a subcomponent thereof) is in a ledger with digest $\mathsf{st}_{\mathbf{L}}$.

We stress that only "valid" transactions can be appended to the ledger. While the full definition of a valid transaction is implementation dependent, in all cases it must be that the commitments and serial numbers in a transaction (including any appearing in the $\star$ field of a transaction) do not already appear on the ledger.

4.3.2 Algorithms

A DPC scheme is a tuple of algorithms (some of which may read information from $\mathbf{L}$):

$$\mathsf{DPC} = (\mathsf{Setup}, \mathsf{GenAddress}, \mathsf{Execute}^{\mathbf{L}}, \mathsf{Verify}^{\mathbf{L}}) \ .$$

The syntax and semantics of these algorithms are informally described below.

- **Setup:** $\mathsf{DPC.Setup}(1^\lambda) \to \mathsf{pp}$.

  On input a security parameter 1^λ, DPC.Setup outputs public parameters pp for the system. A trusted party runs this algorithm once and then publishes its output; afterwards the trusted party is not needed anymore.

  For some constructions, the trusted party can be replaced by an efficient multiparty computation that securely realizes the DPC.Setup algorithm (see [BCGTV15; ZcashCmny; BGM17; BGG18] for how this has been done in some systems); in other constructions, the trusted party may not be needed, as the public parameters may simply consist of a random string of a certain length.

- **Create address:** $\mathsf{DPC.GenAddress}(\mathsf{pp}) \to (\mathsf{apk}, \mathsf{ask})$.

  On input public parameters pp, DPC.GenAddress outputs an address key pair $(\mathsf{apk}, \mathsf{ask})$. Any user may run this algorithm to create an address key pair. Each record is bound to an address public key, and the corresponding secret key is used to consume it.

- **Execute:** Any user may invoke DPC.Execute to consume records and create new ones.

$$\mathsf{DPC.Execute}^{\mathbf{L}}\left(\begin{array}{ll} \text{public parameters} & \mathsf{pp} \\ \text{old records} & [\mathbf{r}_i]_1^m \\ \text{old address secret keys} & [\mathsf{ask}_i]_1^m \\ \text{new address public keys} & [\mathsf{apk}_j]_1^n \\ \text{new record payloads} & [\mathsf{payload}_j]_1^n \\ \text{new record birth predicates} & [\Phi_{\mathsf{b},j}]_1^n \\ \text{new record death predicates} & [\Phi_{\mathsf{d},j}]_1^n \\ \text{auxiliary predicate input} & \mathsf{aux} \\ \text{transaction memorandum} & \mathsf{memo} \end{array}\right) \to \left(\begin{array}{ll} \text{new records} & [\mathbf{r}_j]_1^n \\ \text{transaction} & \mathsf{tx} \end{array}\right).$$

  Given as input a list of old records $[\mathbf{r}_i]_1^m$ with corresponding secret keys $[\mathsf{ask}_i]_1^m$, attributes for new records, private auxiliary input aux to birth and death predicates of new and old records respectively,[7] and an arbitrary transaction memorandum memo, DPC.Execute produces new records $[\mathbf{r}_j]_1^n$ and a transaction tx. The transaction attests that the input records' death predicates and the output records' birth predicates are all satisfied. The user subsequently pushes tx to the ledger by invoking $\mathbf{L}.\mathsf{Push}(\mathsf{tx})$.

- **Verify:** $\mathsf{DPC.Verify}^{\mathbf{L}}(\mathsf{pp}, \mathsf{tx}) \to b$.

  On input public parameters pp and a transaction tx, and given oracle access to the ledger $\mathbf{L}$, DPC.Verify outputs a bit b denoting whether the transaction tx is valid relative to the ledger $\mathbf{L}$.

[7] In addition to the "global" auxiliary input aux, each predicate may also take as input a "local" auxiliary input that is not (necessarily) shared with other predicates. For simplicity, we make these local inputs implicit.

4.3.3 Security

Informally, a DPC scheme achieves the following security goals.

- *Execution correctness.* Malicious parties cannot create valid transactions if the death predicate of some consumed record or the birth predicate of some created record is not satisfied.
- *Execution privacy.* Transactions reveal only the information revealed in the memorandum field, a bound on the number of consumed records, and a bound on the number of created records.[8] All other information is hidden, including the payloads and predicates of all involved records. For example, putting aside the information revealed in the memorandum (which is arbitrary), one cannot link a transaction that consumes a record with the prior transaction that created it.
- *Consumability.* Every record can be consumed at least once and at most once by parties that know its secrets. Thus, a malicious party cannot create two valid records for another party such that only one of them can be consumed. (This captures security against "faerie-gold" attacks [HBHW20].)
- *Transaction non-malleability.* Malicious parties cannot modify a transaction "in flight" to the ledger.

Formally, we prove *standalone* security against *static corruptions*, in a model where every party has private anonymous channels to all other parties [IKOS06].[9] (In Section 4.12 we discuss how to prove security under composition and against adaptive corruptions.) In more detail, we capture security of a DPC scheme via a *simulation-based* security definition that is akin to UC security [Can01], but restricted to a single execution.

Definition 4.3.1. *A DPC scheme* DPC *is* **secure** *if for every efficient real-world adversary* $\mathcal{A}$ *there exists an efficient ideal-world simulator* $\mathcal{S}_{\mathcal{A}}$ *such that for every efficient environment* $\mathcal{E}$ *the following are computationally indistinguishable:*

- *the output of* $\mathcal{E}$ *when interacting with the adversary* $\mathcal{A}$ *in a real-world execution of* DPC *in a model where parties can communicate with other parties via private anonymous channels; and*
- *the output of* $\mathcal{E}$ *when interacting with the simulator* $\mathcal{S}_{\mathcal{A}}$ *in an ideal-world execution with the ideal functionality* $\mathcal{F}_{\mathrm{DPC}}$ *specified in Fig. 4.7 (and further described below).*

We describe the data structures used by the ideal functionality $\mathcal{F}_{\mathrm{DPC}}$, the internal state of $\mathcal{F}_{\mathrm{DPC}}$, and the interface offered by $\mathcal{F}_{\mathrm{DPC}}$ to parties in the ideal-world execution.

Ideal data structures. The ideal functionality $\mathcal{F}_{\mathrm{DPC}}$ uses ideal counterparts of a DPC scheme's data structures. An *address public key* apk denotes the owner of an *ideal record* $\mathbb{r}$, which is a tuple $(\mathsf{cm}, \mathsf{apk}, \mathsf{payload}, \Phi_{\mathsf{b}}, \Phi_{\mathsf{d}})$, where cm is its commitment, apk is its address public key, $\mathsf{payload}$ is its payload, and Φ_{b} and Φ_{d} are its birth and death predicates. The record is also associated with a unique identifier (or *serial number*) sn. We require that apk, cm, and sn are "globally unique"; this means that there cannot be two different ideal records $\mathbb{r}$ and $\mathbb{r}'$ having the same commitments or serial numbers.

[8] And any information implied by knowing that the birth (resp., death) predicates of consumed (resp., created) records are satisfied.

[9] Parties can, e.g., use these channels to communicate the contents of newly created records to other parties.

The distribution of these components is specified by the simulator $\mathcal{S}$ as follows. Before the ideal execution begins, $\mathcal{S}$ specifies three functions $(\mathsf{SampleAddrPk}, \mathsf{SampleCm}, \mathsf{SampleSn})$ that, on input a random string, sample $(\mathsf{apk}, \mathsf{cm}, \mathsf{sn})$ respectively. When $\mathcal{F}_{\text{DPC}}$ needs to sample one of these, it invokes the respective functions. (Note that $\mathcal{F}_{\text{DPC}}$ cannot directly ask $\mathcal{S}$ to sample these because that would reveal to $\mathcal{S}$ when an honest party was invoking $\mathcal{F}_{\text{DPC}}.\mathsf{GenAddress}$ or $\mathcal{F}_{\text{DPC}}.\mathsf{Execute}$, and we cannot afford this leakage.)

Internal state. The ideal functionality $\mathcal{F}_{\text{DPC}}$ maintains several internal tables.

- Addr, which stores address public keys.
- AddrUsers, which maps an address public key to the set of parties that are authorized to use it.
- Records, which maps a record's commitment to that record's information (address public key, payload, birth predicate, and death predicates).
- RecUsers, which maps a record's commitment to the set of parties that are authorized to consume it. Note that, for a record $\mathbb{r}$, the set $\mathsf{RecUsers}[\mathbb{r}.\mathsf{cm}]$ can be different from the set in $\mathsf{AddrUsers}[\mathbb{r}.\mathsf{apk}]$, but a party $\mathcal{P}$ has to be in both sets to consume $\mathbb{r}$.
- SerialNumbers, which maps a record's commitment to that record's (unique) serial number.
- State, which maps a record's commitment to that record's state, either `alive` or `dead`.

Ideal algorithms. The ideal functionality $\mathcal{F}_{\text{DPC}}$ provides the following interface to parties.

- *Address generation:* $\mathcal{F}_{\text{DPC}}.\mathsf{GenAddress}$ outputs a new address public key apk.
- *Execution:* $\mathcal{F}_{\text{DPC}}.\mathsf{Execute}$ performs an execution that consumes old records and creates new records. All parties are notified that an execution has occurred, and learn the serial numbers of input records, commitments of output records, and the transaction memorandum memo. Concurrent $\mathcal{F}_{\text{DPC}}.\mathsf{Execute}$ calls are serialized arbitrarily.
- *Record consumption authorization:* $\mathcal{F}_{\text{DPC}}.\mathsf{ShareRecord}$ allows a party $\mathcal{P}$ to authorize another party $\mathcal{P}'$ to consume a record $\mathbb{r}$ (provided that $\mathcal{P}'$ is also authorized to use $\mathbb{r}$'s address public key).

Operation of honest parties. In both the real and ideal executions, the environment $\mathcal{E}$ can send instructions to honest parties. These instructions can be one of GenAddress, Execute, or ShareRecord. In the real world honest parties translate these instructions into corresponding invocations of DPC algorithms (or messages sent via private anonymous channels as in the case of ShareRecord), while in the ideal world they translate them into corresponding invocations of $\mathcal{F}_{\text{DPC}}$ algorithms. In both worlds, honest parties immediately invoke ShareRecord on records obtained from an Execute instruction. Finally, in the ideal world, when invoking $\mathcal{F}_{\text{DPC}}$, honest parties do not provide any inputs marked as optional; instead, they let $\mathcal{F}_{\text{DPC}}$ sample these.

Intuition. We explain how $\mathcal{F}_{\text{DPC}}$ enforces the informal security notions described at this section's beginning.

- *Execution correctness.* $\mathcal{F}_{\text{DPC}}.\mathsf{Execute}$ ensures that the death predicates of consumed records and birth predicates of created records are satisfied by the local data. Note that each predicate receives its own position as input so that it knows to which record in the local data it belongs.
- *Execution privacy.* Transactions contain serial numbers $[\mathsf{sn}_i]_1^m$ of consumed records, commitments $[\mathsf{cm}_j]_1^n$ of created records, and a memorandum memo. Serial numbers and commitments are sampled via SampleSn and SampleCm, so they are independent of the contents of any record, and

thus reveal no information about them. Transactions thus reveal no information (beyond what is contained in memo).

- *Consumability.* From the point of view of $\mathcal{F}_{\text{DPC}}$, two records are different if and only if they have different commitments. In such a case, both records can be consumed as long as their death predicates are satisfied. If a DPC scheme realizes $\mathcal{F}_{\text{DPC}}$, then it must satisfy this same requirement: if two valid records have distinct commitments, then they must both be consumable.
- *Transaction non-malleability.* The adversary has no power to modify the inputs to, or output of, an honest party's invocation of $\mathcal{F}_{\text{DPC}}$.Execute.

$\mathcal{F}_{\text{DPC}}.\mathsf{GenAddress}[\mathcal{P}]\left((\text{optional}) \text{ address public key } \mathsf{apk}\right)$

1. Sample randomness r for generating address public key.
2. If $\mathsf{apk} = \bot$ then $\mathsf{apk} \leftarrow \mathsf{SampleAddrPk}(r)$.
3. Check that apk is **unique**: $\mathsf{Addr}[\mathsf{apk}] = \bot$.
4. Set $\mathsf{Addr}[\mathsf{apk}] := r$.
5. If $\mathcal{P}$ is corrupted: set S to be the set of corrupted parties.
6. If $\mathcal{P}$ is honest: set $S := \{\mathcal{P}\}$.
7. Set $\mathsf{AddrUsers}[\mathsf{apk}] := \mathsf{AddrUsers}[\mathsf{apk}] \cup S$.
8. **Send to $\mathcal{P}$:** address public key apk.

$\mathcal{F}_{\text{DPC}}.\mathsf{ShareRecord}[\mathcal{P}]\begin{pmatrix}\text{record} & \mathbb{r} \\ \text{recipient party} & \mathcal{P}'\end{pmatrix}$

1. If $\mathsf{Records}[\mathbb{r}.\mathsf{cm}] \neq \bot$:
   a) Check that $\mathcal{P} \in \mathsf{RecUsers}[\mathbb{r}.\mathsf{cm}]$.
   b) Retrieve $((\mathsf{cm}, \mathsf{apk}, \mathsf{payload}, \Phi_{\mathsf{b}}, \Phi_{\mathsf{d}}), r) := \mathsf{Records}[\mathbb{r}.\mathsf{cm}]$.
2. If $\mathcal{P}'$ is corrupted: set S to be the set of corrupted parties.
3. If $\mathcal{P}'$ is honest: set $S := \{\mathcal{P}'\}$.
4. Set $\mathsf{RecUsers}[\mathbb{r}.\mathsf{cm}] := \mathsf{RecUsers}[\mathbb{r}.\mathsf{cm}] \cup S$.
5. If $\mathcal{P}$ is honest and $\mathcal{P}'$ isn't, **Send to $\mathcal{P}'$:** $(\texttt{RecordAuth}, (\mathbb{r}, r))$.
6. Else, **Send to $\mathcal{P}'$:** $(\texttt{RecordAuth}, \mathbb{r})$.

$\mathcal{F}_{\text{DPC}}.\mathsf{Execute}[\mathcal{P}]$

old records	$[\mathbb{r}_i]_1^m$
(optional) old serial numbers	$[\mathsf{sn}_i]_1^m$
(optional) new record commitments	$[\mathsf{cm}_j]_1^n$
new address public keys	$[\mathsf{apk}_j]_1^n$
new record payloads	$[\mathsf{payload}_j]_1^n$
new record birth predicates	$[\Phi_{\mathsf{b},j}]_1^n$
new record death predicates	$[\Phi_{\mathsf{d},j}]_1^n$
auxiliary predicate input	aux
transaction memorandum	memo

1. For each $i \in \{1, \ldots, m\}$:
   (a) Sample randomness r_i.
   (b) If $\mathsf{sn}_i = \bot$ then generate **serial number**: $\mathsf{sn}_i \leftarrow \mathsf{SampleSn}(r_i)$.
   (c) Check that sn_i is **unique**: $\mathsf{SerialNumbers}[\mathsf{sn}_i] = \bot$.
2. For each $j \in \{1, \ldots, n\}$:
   (a) Sample randomness r_j.
   (b) If $\mathsf{cm}_j = \bot$ then generate **commitment**: $\mathsf{cm}_j \leftarrow \mathsf{SampleCm}(r_j)$.
   (c) Check that cm_j is **unique**: $\mathsf{Records}[\mathsf{cm}_j] = \bot$.
   (d) **Construct record**: $\mathbb{r}_j := (\mathsf{cm}_j, \mathsf{apk}_j, \mathsf{payload}_j, \Phi_{\mathsf{b},j}, \Phi_{\mathsf{d},j})$.
3. Define the local data $\mathsf{ldata} := ([\mathbb{r}_i]_1^m, [\mathsf{sn}_i]_1^m, [\mathbb{r}_j]_1^n, \mathsf{aux}, \mathsf{memo})$.
4. For each $i \in \{1, \ldots, m\}$:
   a) Parse $\mathbb{r}_i$ as $(\mathsf{cm}_i, \mathsf{apk}_i, \mathsf{payload}_i, \Phi_{\mathsf{b},i}, \Phi_{\mathsf{d},i})$.
   b) Check that, for some randomness r_i, **old record** $\mathbb{r}_i$ **exists**: $((\mathsf{apk}_i, \mathsf{payload}_i, \Phi_{\mathsf{b},i}, \Phi_{\mathsf{d},i}), r_i) = \mathsf{Records}[\mathsf{cm}_i]$.
   c) Check that $\mathcal{P}$ **is authorized to use** apk_i: $\mathcal{P} \in \mathsf{AddrUsers}[\mathsf{apk}_i]$.
   d) If $\mathsf{payload}_i.\mathsf{isDummy} = 0$:
      i. Check that **record is unconsumed**: $\mathsf{State}[\mathbb{r}_i] = \texttt{alive}$.
      ii. Check that $\mathcal{P}$ **is authorized to consume** $\mathbb{r}_i$: $\mathcal{P} \in \mathsf{RecUsers}[\mathsf{cm}_i]$.
      iii. Check that $\mathcal{P}$ **is authorized to use** apk_i: $\mathcal{P} \in \mathsf{AddrUsers}[\mathsf{apk}_i]$.
   e) Check that **death predicate is satisfied**: $\Phi_{\mathsf{d},i}(i \| \mathsf{ldata}) = 1$.
   f) **Mark it as consumed**: $\mathsf{State}[\mathsf{cm}_i] := \texttt{dead}$.
5. For each $j \in \{1, \ldots, n\}$:
   a) Check that **birth predicate is satisfied**: $\Phi_{\mathsf{b},j}(j \| \mathsf{ldata}) = 1$.
   b) **Insert new record $\mathbf{r}_j$**: $\mathsf{Records}[\mathsf{cm}_j] := ((\mathsf{apk}_j, \mathsf{payload}_j, \Phi_{\mathsf{b},j}, \Phi_{\mathsf{d},j}), r_j)$.
   c) **Mark new record as unconsumed**: $\mathsf{State}[\mathsf{cm}_j] := \texttt{alive}$.
6. **Send to $\mathcal{P}$:** $([\mathbb{r}_j]_1^n)$.
7. **Send to all parties:** $(\texttt{Execute}, [\mathsf{sn}_i]_1^m, [\mathsf{cm}_j]_1^n, \mathsf{memo})$.

Figure 4.7: Ideal functionality $\mathcal{F}_{\text{DPC}}$ of a DPC scheme.

4.4 Construction of decentralized private computation schemes

We describe our construction of a DPC scheme. In Section 4.4.1 we introduce the building blocks that we use, and in Section 4.4.2 we describe each algorithm in the scheme. The security proof is provided in Section 4.10. We also describe some extensions of our construction, in functionality and in security, in Section 4.12.

4.4.1 Building blocks

CRHs. A collision-resistant hash function $\mathsf{CRH} = (\mathsf{Setup}, \mathsf{Eval})$ works as follows.
- *Setup:* on input a security parameter, $\mathsf{CRH.Setup}$ samples public parameters $\mathsf{pp}_{\mathsf{CRH}}$.
- *Hashing:* on input public parameters $\mathsf{pp}_{\mathsf{CRH}}$ and message m, $\mathsf{CRH.Eval}$ outputs a short hash h of m.

Given public parameters $\mathsf{pp}_{\mathsf{CRH}} \leftarrow \mathsf{CRH.Setup}(1^\lambda)$, it is computationally infeasible to find distinct inputs x and y such that $\mathsf{CRH.Eval}(\mathsf{pp}_{\mathsf{CRH}}, x) = \mathsf{CRH.Eval}(\mathsf{pp}_{\mathsf{CRH}}, y)$.

PRFs. A pseudorandom function family $\mathsf{PRF} = \{\mathsf{PRF}_x \colon \{0,1\}^* \to \{0,1\}^{O(|x|)}\}_x$, where x denotes the seed, is computationally indistinguishable from a random function family.

Commitments. A commitment scheme $\mathsf{CM} = (\mathsf{Setup}, \mathsf{Commit})$ enables a party to generate a (perfectly) hiding and (computationally) binding commitment to a given message.
- *Setup:* on input a security parameter, $\mathsf{CM.Setup}$ samples public parameters $\mathsf{pp}_{\mathsf{CM}}$.
- *Commitment:* on input public parameters $\mathsf{pp}_{\mathsf{CM}}$, message m, and randomness r_{cm}, $\mathsf{CM.Commit}$ outputs a commitment cm to m.

We also use a *trapdoor* commitment scheme $\mathsf{TCM} = (\mathsf{Setup}, \mathsf{Commit})$, with the same syntax as above. Auxiliary algorithms (beyond those in CM) enable producing a trapdoor and using it to open a commitment, originally to an empty string, to an arbitrary message. These algorithms are used only in the proof of security, and so we introduce them there (see Section 4.10).

NIZKs. Non-interactive zero knowledge arguments of knowledge enable a party, known as the *prover*, to convince another party, known as the *verifier*, about knowledge of the witness for an NP statement without revealing any information about the witness (besides what is already implied by the statement being true). This primitive is a tuple $\mathsf{NIZK} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ with the following syntax.
- *Setup:* on input a security parameter and the specification of an NP relation $\mathcal{R}$, $\mathsf{NIZK.Setup}$ outputs a set of public parameters $\mathsf{pp}_{\mathsf{NIZK}}$ (also known as a *common reference string*).
- *Proving:* on input $\mathsf{pp}_{\mathsf{NIZK}}$ and an instance-witness pair $(\mathbb{x}, \mathbb{w}) \in \mathcal{R}$, $\mathsf{NIZK.Prove}$ outputs a proof π.
- *Verifying:* on input $\mathsf{pp}_{\mathsf{NIZK}}$, instance $\mathbb{x}$, and proof π, $\mathsf{NIZK.Verify}$ outputs a decision bit.

Completeness states that honestly generated proofs make the verifier accept; *(computational) proof of knowledge* states that if the verifier accepts a proof for an instance then the prover "knows" a witness for it; and *perfect zero knowledge* states that honestly generated proofs can be perfectly simulated, when given a trapdoor to the public parameters. In fact, we require a strong form of (computational) proof of knowledge known as *simulation-extractability*, which states that proofs

continue to be proofs of knowledge even when the adversary has seen prior simulated proofs. For more details, see [Sah99; DDOPS01; Gro06].

Remark 4.4.1. If NIZK is additionally *succinct* (i.e., it is a simulation-extractable zkSNARK) then the DPC scheme constructed in this section is also *succinct*. This is the case in our implementation; see Section 4.8.

4.4.2 Algorithms

Pseudocode for our construction of a DPC scheme is in Fig. 4.8. The construction involves invoking zero knowledge proofs for the NP relation $\mathcal{R}_{\mathsf{e}}$ described in Fig. 4.9. The text below is a summary of the construction.

System setup. DPC.Setup is a wrapper around the setup algorithms of cryptographic building blocks. It invokes CM.Setup, TCM.Setup, CRH.Setup, and NIZK.Setup to obtain (plain and trapdoor) commitment public parameters $\mathsf{pp}_{\mathsf{CM}}$ and $\mathsf{pp}_{\mathsf{TCM}}$, CRH public parameters $\mathsf{pp}_{\mathsf{CRH}}$, and NIZK public parameters for the NP relation $\mathcal{R}_{\mathsf{e}}$ (see Fig. 4.9). It then outputs $\mathsf{pp} := (\mathsf{pp}_{\mathsf{CM}}, \mathsf{pp}_{\mathsf{TCM}}, \mathsf{pp}_{\mathsf{CRH}}, \mathsf{pp}_{\mathsf{e}})$.

Address creation. DPC.GenAddress constructs an address key pair as follows. The address secret key $\mathsf{ask} = (\mathsf{sk}_{\mathsf{PRF}}, r_{\mathsf{pk}})$ consists of a secret key $\mathsf{sk}_{\mathsf{PRF}}$ for the pseudorandom function PRF, and commitment randomness r_{pk}. The address public key apk is a perfectly hiding commitment to $\mathsf{sk}_{\mathsf{PRF}}$ with randomness r_{pk}.

Execution. DPC.Execute produces a transaction attesting that some old records $[\mathbf{r}_i]_1^m$ were consumed and some new records $[\mathbf{r}_j]_1^n$ were created, and that their death and birth predicates were satisfied. First, DPC.Execute computes a ledger membership witness and serial number for every old record. Then, DPC.Execute invokes the following auxiliary function to create record commitments for the new records.

$\mathsf{DPC.ConstructRecord}(\mathsf{pp}, \mathsf{apk}, \mathsf{payload}, \Phi_{\mathsf{b}}, \Phi_{\mathsf{d}}, \rho) \to (\mathbf{r}, \mathsf{cm})$

1. Sample new commitment **randomness** r.
2. Construct new record **commitment**: $\mathsf{cm} \leftarrow \mathsf{TCM.Commit}(\mathsf{pp}_{\mathsf{TCM}}, \mathsf{apk} \| \mathsf{payload} \| \Phi_{\mathsf{b}} \| \Phi_{\mathsf{d}} \| \rho;\ r)$.
3. Construct new **record** $\mathbf{r} := \begin{pmatrix} \text{address public key} & \mathsf{apk} & \text{payload} & \mathsf{payload} & \text{comm. rand.} & r \\ \text{serial number nonce} & \rho & \text{predicates} & (\Phi_{\mathsf{b}}, \Phi_{\mathsf{d}}) & \text{commitment} & \mathsf{cm} \end{pmatrix}$.
4. Output $(\mathbf{r}, \mathsf{cm})$.

Information about all records, secret addresses of old records, the desired transaction memorandum memo, and desired auxiliary predicate input aux are collected into the local data ldata (see Fig. 4.9).

Finally, DPC.Execute produces a proof that all records are well-formed and that several conditions hold.

- *Old records are properly consumed*, namely, for every old record $\mathbf{r}_i \in [\mathbf{r}_i]_1^m$:
  - (if $\mathbf{r}_i$ is not dummy) $\mathbf{r}_i$ *exists*, demonstrated by checking a ledger membership witness for $\mathbf{r}_i$'s commitment;
  - $\mathbf{r}_i$ *has not been consumed*, demonstrated by publishing $\mathbf{r}_i$'s serial number sn_i;

  - $\mathbf{r}_i$*'s death predicate* $\Phi_{\mathsf{d},i}$ *is satisfied*, demonstrated by checking that $\Phi_{\mathsf{d},i}(i \| \mathsf{ldata}) = 1$.

- *New records are property created*, namely, for every new record $\mathbf{r}_j \in [\mathbf{r}_j]_1^n$:
  - $\mathbf{r}_j$*'s serial number is unique*, achieved by setting nonce $\rho_j := \mathsf{CRH.Eval}(\mathsf{pp}_{\mathsf{CRH}}, j \| \mathsf{sn}_1 \| \ldots \| \mathsf{sn}_m)$;
  - $\mathbf{r}_j$*'s birth predicate* $\Phi_{\mathsf{b},j}$ *is satisfied*, demonstrated by checking that $\Phi_{\mathsf{b},j}(j \| \mathsf{ldata}) = 1$.

The serial number sn of a record $\mathbf{r}$ relative to an address secret key $\mathsf{ask} = (\mathsf{sk}_{\mathsf{PRF}}, r_{\mathsf{pk}})$ is derived by evaluating PRF at $\mathbf{r}$'s serial number nonce ρ with seed $\mathsf{sk}_{\mathsf{PRF}}$. This ensures that sn is pseudorandom even to a party that knows all of $\mathbf{r}$ but not ask (e.g., to a party that created the record for some other party). Note that each predicate receives its own position as input so that it knows to which record in the local data it belongs.

DPC.Setup

Input: security parameter 1^λ

Output: public parameters pp

1. Generate **commitment parameters**: $\mathsf{pp}_{\mathsf{CM}} \leftarrow \mathsf{CM.Setup}(1^\lambda)$, $\mathsf{pp}_{\mathsf{TCM}} \leftarrow \mathsf{TCM.Setup}(1^\lambda)$.
2. Generate **CRH parameters**: $\mathsf{pp}_{\mathsf{CRH}} \leftarrow \mathsf{CRH.Setup}(1^\lambda)$.
3. Generate **NIZK parameters for** $\mathcal{R}_{\mathsf{e}}$ (see Figure 4.9): $\mathsf{pp}_{\mathsf{e}} \leftarrow \mathsf{NIZK.Setup}(1^\lambda, \mathcal{R}_{\mathsf{e}})$.
4. Output $\mathsf{pp} := (\mathsf{pp}_{\mathsf{CM}}, \mathsf{pp}_{\mathsf{TCM}}, \mathsf{pp}_{\mathsf{CRH}}, \mathsf{pp}_{\mathsf{e}})$.

DPC.GenAddress

Input: public parameters pp

Output: address key pair $(\mathsf{apk}, \mathsf{ask})$

1. Sample secret key $\mathsf{sk}_{\mathsf{PRF}}$ for pseudorandom function PRF.
2. Sample randomness r_{pk} for commitment scheme CM.
3. Set **address public key** $\mathsf{apk} := \mathsf{CM.Commit}(\mathsf{pp}_{\mathsf{CM}}, \mathsf{sk}_{\mathsf{PRF}};\ r_{\mathsf{pk}})$.
4. Set **address secret key** $\mathsf{ask} := (\mathsf{sk}_{\mathsf{PRF}}, r_{\mathsf{pk}})$.
5. Output $(\mathsf{apk}, \mathsf{ask})$.

DPC.Execute$^{\mathbf{L}}$

Input:

- public parameters pp
- old $\begin{cases} \text{records } [\mathbf{r}_i]_1^m \\ \text{address secret keys } [\mathsf{ask}_i]_1^m \end{cases}$
- new $\begin{cases} \text{address public keys } [\mathsf{apk}_j]_1^n \\ \text{record payloads } [\mathsf{payload}_j]_1^n \\ \text{record birth predicates } [\Phi_{\mathsf{b},j}]_1^n \\ \text{record death predicates } [\Phi_{\mathsf{d},j}]_1^n \end{cases}$
- auxiliary predicate input aux
- transaction memorandum memo

Output: new records $[\mathbf{r}_j]_1^n$ and transaction tx

1. For each $i \in \{1, \ldots, m\}$, process the i-th old record as follows:
   a) Parse old record $\mathbf{r}_i$ as $\begin{pmatrix} \text{address public key} & \mathsf{apk}_i & \text{payload} & \mathsf{payload}_i & \text{comm. rand.} & r_i \\ \text{serial number nonce} & \rho_i & \text{predicates} & (\Phi_{\mathsf{b},i}, \Phi_{\mathsf{d},i}) & \text{commitment} & \mathsf{cm}_i \end{pmatrix}$.
   b) If $\mathsf{payload}_i.\mathsf{isDummy} = 1$, set **ledger membership witness** $\mathrm{w}_{\mathbf{L},i} := \bot$.
   If $\mathsf{payload}_i.\mathsf{isDummy} = 0$, compute **ledger membership witness** for commitment: $\mathrm{w}_{\mathbf{L},i} \leftarrow \mathbf{L}.\mathsf{Prove}(\mathsf{cm}_i)$.
   c) Parse address secret key ask_i as $(\mathsf{sk}_{\mathsf{PRF},i}, r_{\mathsf{pk},i})$.
   d) Compute **serial number**: $\mathsf{sn}_i \leftarrow \mathsf{PRF}_{\mathsf{sk}_{\mathsf{PRF},i}}(\rho_i)$.
2. For each $j \in \{1, \ldots, n\}$, construct the j-th new record as follows:
   a) Compute **serial number nonce**: $\rho_j := \mathsf{CRH.Eval}(\mathsf{pp}_{\mathsf{CRH}}, j \| \mathsf{sn}_1 \| \ldots \| \mathsf{sn}_m)$.
   b) Construct **new record**: $(\mathbf{r}_j, \mathsf{cm}_j) \leftarrow \mathsf{DPC.ConstructRecord}(\mathsf{pp}_{\mathsf{TCM}}, \mathsf{apk}_j, \mathsf{payload}_j, \Phi_{\mathsf{b},j}, \Phi_{\mathsf{d},j}, \rho_j)$.
3. Retrieve current **ledger digest**: $\mathsf{st}_{\mathbf{L}} \leftarrow \mathbf{L}.\mathsf{Digest}$.
4. Construct **instance** x_{e} **for** $\mathcal{R}_{\mathsf{e}}$: $\mathrm{x}_{\mathsf{e}} := (\mathsf{st}_{\mathbf{L}}, [\mathsf{sn}_i]_1^m, [\mathsf{cm}_j]_1^n, \mathsf{memo})$.
5. Construct **witness** w_{e} **for** $\mathcal{R}_{\mathsf{e}}$: $\mathrm{w}_{\mathsf{e}} := ([\mathbf{r}_i]_1^m, [\mathrm{w}_{\mathbf{L},i}]_1^m, [\mathsf{ask}_i]_1^m, [\mathbf{r}_j]_1^n, \mathsf{aux})$.
6. Generate **proof for** $\mathcal{R}_{\mathsf{e}}$: $\pi_{\mathsf{e}} \leftarrow \mathsf{NIZK.Prove}(\mathsf{pp}_{\mathsf{e}}, \mathrm{x}_{\mathsf{e}}, \mathrm{w}_{\mathsf{e}})$.
7. Construct **transaction**: $\mathsf{tx} := ([\mathsf{sn}_i]_1^m, [\mathsf{cm}_j]_1^n, \mathsf{memo}, \star)$, where $\star := (\mathsf{st}_{\mathbf{L}}, \pi_{\mathsf{e}})$.
8. Output $([\mathbf{r}_j]_1^n, \mathsf{tx})$.

DPC.Verify$^{\mathbf{L}}$

Input: public parameters pp and transaction tx

Output: decision bit b

1. Parse tx as $([\mathsf{sn}_i]_1^m, [\mathsf{cm}_j]_1^n, \mathsf{memo}, \star)$ and $\star$ as $(\mathsf{st}_{\mathbf{L}}, \pi_{\mathsf{e}})$.
2. Check that **there are no duplicate serial numbers**
   a) within the transaction tx: $\mathsf{sn}_i \neq \mathsf{sn}_j$ for every distinct $i, j \in \{1, \ldots, m\}$;
   b) on the ledger: $\mathbf{L}.\mathsf{Contains}(\mathsf{sn}_i) = 0$ for every $i \in \{1, \ldots, m\}$.
3. Check that **the ledger state is valid**: $\mathbf{L}.\mathsf{ValidateDigest}(\mathsf{st}_{\mathbf{L}}) = 1$.
4. Construct **instance for the relation** $\mathcal{R}_{\mathsf{e}}$: $\mathrm{x}_{\mathsf{e}} := (\mathsf{st}_{\mathbf{L}}, [\mathsf{sn}_i]_1^m, [\mathsf{cm}_j]_1^n, \mathsf{memo})$.
5. Check **proof for the relation** $\mathcal{R}_{\mathsf{e}}$: $\mathsf{NIZK.Verify}(\mathsf{pp}_{\mathsf{e}}, \mathrm{x}_{\mathsf{e}}, \pi_{\mathsf{e}}) = 1$.

Figure 4.8: Construction of a DPC scheme.

$$\mathbb{x}_{\mathsf{e}} = \begin{pmatrix} \text{ledger digest} & \mathsf{st}_{\mathbf{L}} \\ \text{old record serial numbers} & [\mathsf{sn}_i]_1^m \\ \text{new record commitments} & [\mathsf{cm}_j]_1^n \\ \text{transaction memorandum} & \mathsf{memo} \end{pmatrix} \quad \text{and} \quad \mathbb{w}_{\mathsf{e}} = \begin{pmatrix} \text{old records} & [\mathbf{r}_i]_1^m \\ \text{old record membership witnesses} & [\mathbb{w}_{\mathbf{L},i}]_1^m \\ \text{old address secret keys} & [\mathsf{ask}_i]_1^m \\ \text{new records} & [\mathbf{r}_j]_1^n \\ \text{auxiliary predicate input} & \mathsf{aux} \end{pmatrix}$$

where

- for each $i \in \{1, \ldots, m\}$, $\mathbf{r}_i = (\mathsf{apk}_i, \mathsf{payload}_i, \Phi_{\mathsf{b},i}, \Phi_{\mathsf{d},i}, \rho_i, r_i, \mathsf{cm}_i)$;
- for each $j \in \{1, \ldots, n\}$, $\mathbf{r}_j = (\mathsf{apk}_j, \mathsf{payload}_j, \Phi_{\mathsf{b},j}, \Phi_{\mathsf{d},j}, \rho_j, r_j, \mathsf{cm}_j)$.

Define the local data $\mathsf{ldata} := \begin{pmatrix} [\mathsf{cm}_i]_1^m & [\mathsf{apk}_i]_1^m & [\mathsf{payload}_i]_1^m & [\Phi_{\mathsf{d},i}]_1^m & [\Phi_{\mathsf{b},i}]_1^m & [\mathsf{sn}_i]_1^m & \mathsf{memo} \\ [\mathsf{cm}_j]_1^n & [\mathsf{apk}_j]_1^n & [\mathsf{payload}_j]_1^n & [\Phi_{\mathsf{d},j}]_1^n & [\Phi_{\mathsf{b},j}]_1^n & \mathsf{aux} & \end{pmatrix}$.

Then, a witness $\mathbb{w}_{\mathsf{e}}$ is valid for an instance $\mathbb{x}_{\mathsf{e}}$ if the following conditions hold:

1. For each $i \in \{i, \ldots, m\}$:
   - If $\mathbf{r}_i$ is not dummy, $\mathbb{w}_{\mathbf{L},i}$ proves that the commitment cm_i is in a ledger with digest $\mathsf{st}_{\mathbf{L}}$: $\mathbf{L}.\mathsf{Verify}(\mathsf{st}_{\mathbf{L}}, \mathsf{cm}_i, \mathbb{w}_{\mathbf{L},i}) = 1$.
   - The address public key apk_i and secret key ask_i form a valid key pair: $\mathsf{apk}_i = \mathsf{CM.Commit}(\mathsf{pp}_{\mathsf{CM}}, \mathsf{sk}_{\mathsf{PRF},i};\ r_{\mathsf{pk},i})$ and $\mathsf{ask}_i = (\mathsf{sk}_{\mathsf{PRF},i}, r_{\mathsf{pk},i})$.
   - The serial number sn_i is valid: $\mathsf{sn}_i = \mathsf{PRF}_{\mathsf{sk}_{\mathsf{PRF},i}}(\rho_i)$.
   - The old record commitment cm_i is valid: $\mathsf{cm}_i = \mathsf{TCM.Commit}(\mathsf{pp}_{\mathsf{TCM}}, \mathsf{apk}_i \| \mathsf{payload}_i \| \Phi_{\mathsf{b},i} \| \Phi_{\mathsf{d},i} \| \rho_i;\ r_i)$.
   - The death predicate $\Phi_{\mathsf{d},i}$ is satisfied by local data: $\Phi_{\mathsf{d},i}(i \| \mathsf{ldata}) = 1$.
2. For each $j \in \{1, \ldots, n\}$:
   - The serial number nonce ρ_j is computed correctly: $\rho_j = \mathsf{CRH.Eval}(\mathsf{pp}_{\mathsf{CRH}}, j \| \mathsf{sn}_1 \| \ldots \| \mathsf{sn}_m)$.
   - The new record commitment cm_j is valid: $\mathsf{cm}_j = \mathsf{TCM.Commit}(\mathsf{pp}_{\mathsf{TCM}}, \mathsf{apk}_j \| \mathsf{payload}_j \| \Phi_{\mathsf{b},j} \| \Phi_{\mathsf{d},j} \| \rho_j;\ r_j)$.
   - The birth predicate $\Phi_{\mathsf{b},j}$ is satisfied by local data: $\Phi_{\mathsf{b},j}(j \| \mathsf{ldata}) = 1$.

Figure 4.9: The execute NP relation $\mathcal{R}_{\mathsf{e}}$.

4.5 Delegating zero knowledge execution

The cost of creating a transaction in the DPC scheme from Section 4.4 grows with the complexity (and number of) predicates involved in the transaction. The user must produce, and include in the transaction, a cryptographic proof that, among other things, attests that death predicates of consumed records are satisfied and, similarly, that birth predicates of created records are satisfied. This implies that producing transactions on weak devices such as mobile phones or hardware tokens quickly becomes infeasible.

In Sections 4.5.1 to 4.5.3 we explain how to address this problem by enabling a user to *delegate* to an untrusted worker, such as a remote server, the computation that produces a transaction. This empowers weak devices to produce transactions that they otherwise could not have produced on their own. Then, in Section 4.5.4, we explain how the ideas that we use for delegating transactions also yield solutions for achieving *threshold transactions* and *blind transactions* in a DPC scheme, which are also valuable in applications. Techniques derived from these ideas are now part of deployed systems [HBHW20].

4.5.1 Approach

A naive approach is for the user to simply ask the worker to produce the cryptographic proof on its behalf, and then include this proof in the transaction. The intuition behind this idea is that the user can check that the proof received from the worker is valid, by simply running the proof verification procedure. Indeed, whenever the DPC scheme uses a succinct argument (see Remark 4.4.1), the verification procedure is *succinct*.

However, this approach is *insecure*, because the worker, in order to produce a proof, would have to learn not only the instance but also the secret witness for the NP statement being proved. Since the secret witness includes the user's address secret key, if the worker learns this information then the worker can impersonate the user, e.g., by producing further transactions that the user never *authorized*. This naive approach also fails in prior proof-based ledger protocols, including Zerocash [Ben+14]. New ideas are needed.

Taking our construction of a DPC scheme from Section 4.4 as a starting point, we explain how to enable a user to delegate the expensive proof computation to a worker in such a way that the worker *cannot* produce valid transactions that have not been authorized by the user; see Fig. 4.11. (Additional security goals, such as ensuring that the worker learns no information about the user, are left to future work.)

The basic idea is to augment address keys in such a way that the secret information needed to produce the cryptographic proof is separate from the secret information needed to authorize a transaction containing that proof. Thus, the user can communicate to the worker the secrets necessary to generate a cryptographic proof, while retaining the remaining secrets for authorizing this (and future) transactions. In particular, the worker has no way to produce valid transactions that have not been authorized by the user.

We stress that the simplistic solution in which the user authorizes the proof produced by the worker by signing it via a secret key not shared with the worker *does not work* because it violates

privacy. Indeed, others would have to use the same public key to verify signatures across multiple transactions containing signatures produced by the same secret key, thereby linking these transactions together.

The next two sub-sections explain how we achieve delegation: first, in Section 4.5.2, we describe a variant of randomizable signatures, which we use as a building block; then, in Section 4.5.3, we provide a high-level description of a *delegable* DPC scheme. The detailed construction is provided in Section 4.11.

4.5.2 Additional building block: randomizable signatures

A *randomizable* signature scheme is a tuple of algorithms $\mathsf{SIG} = (\mathsf{Setup}, \mathsf{Keygen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{RandPk}, \mathsf{RandSig})$ that enables a party to sign messages, while also allowing randomization of public keys and signatures to prevent linking across multiple signatures. We first discuss the syntax of the usual algorithms.

- *Setup:* on input a security parameter, $\mathsf{SIG.Setup}$ samples public parameters $\mathsf{pp}_{\mathsf{SIG}}$.
- *Key generation:* on input public parameters $\mathsf{pp}_{\mathsf{SIG}}$, $\mathsf{SIG.Keygen}$ samples a key pair $(\mathsf{pk}_{\mathsf{SIG}}, \mathsf{sk}_{\mathsf{SIG}})$.
- *Message signing:* on input public parameters $\mathsf{pp}_{\mathsf{SIG}}$, secret key $\mathsf{sk}_{\mathsf{SIG}}$, and message m, $\mathsf{SIG.Sign}$ produces a signature σ.
- *Signature verification:* on input public parameters $\mathsf{pp}_{\mathsf{SIG}}$, public key $\mathsf{pk}_{\mathsf{SIG}}$, message m, and signature σ, $\mathsf{SIG.Verify}$ outputs a bit b denoting whether σ is a valid signature for m under public key $\mathsf{pk}_{\mathsf{SIG}}$.

In addition to the usual algorithms, SIG has two algorithms for randomizing public keys and signatures.

- *Public key randomization:* $\mathsf{SIG.RandPk}(\mathsf{pp}_{\mathsf{SIG}}, \mathsf{pk}_{\mathsf{SIG}}, r_{\mathsf{SIG}})$ samples a randomized public key $\hat{\mathsf{pk}}_{\mathsf{SIG}}$.
- *Signature randomization:* $\mathsf{SIG.RandSig}(\mathsf{pp}_{\mathsf{SIG}}, \sigma, r_{\mathsf{SIG}})$ samples a randomized signature $\hat{\sigma}$.

The signature scheme SIG must satisfy the following security properties.

- *Existential unforgeability.* Given a public key $\mathsf{pk}_{\mathsf{SIG}}$, it is infeasible to produce a forgery under $\mathsf{pk}_{\mathsf{SIG}}$ or *under any randomization of* $\mathsf{pk}_{\mathsf{SIG}}$. This notion strengthens the standard unforgeability notion, and is similar to that of randomizable signatures in [FKMSSS16].
- *Unlinkability.* Given a public key $\mathsf{pk}_{\mathsf{SIG}}$ and a tuple $(\hat{\mathsf{pk}}_{\mathsf{SIG}}, m, \hat{\sigma})$ where $\hat{\sigma}$ is a valid signature for m under $\hat{\mathsf{pk}}_{\mathsf{SIG}}$, no efficient adversary can determine if $\hat{\mathsf{pk}}_{\mathsf{SIG}}$ is a fresh public key and $\hat{\sigma}$ a fresh signature, or if instead $\hat{\mathsf{pk}}_{\mathsf{SIG}}$ is a randomization of $\mathsf{pk}_{\mathsf{SIG}}$ and $\hat{\sigma}$ a randomization of a signature for $\mathsf{pk}_{\mathsf{SIG}}$. This property is a computational relaxation of the perfect unlinkability property of randomizable signatures in [FKMSSS16].
- *Injective randomization.* Randomization of public keys is *(computationally) injective* with respect to randomness. Informally, given public parameters $\mathsf{pp}_{\mathsf{SIG}}$, it is infeasible to find a public key $\mathsf{pk}_{\mathsf{SIG}}$ and $r_1 \neq r_2$ such that $\mathsf{SIG.RandPk}(\mathsf{pp}_{\mathsf{SIG}}, \mathsf{pk}_{\mathsf{SIG}}, r_1) = \mathsf{SIG.RandPk}(\mathsf{pp}_{\mathsf{SIG}}, \mathsf{pk}_{\mathsf{SIG}}, r_2)$.

4.5.3 A delegable DPC scheme

We describe how to construct a *delegable DPC scheme*, namely, a DPC scheme in which a user can delegate to an untrusted worker the expensive computations associated with producing a transaction. The security goal is that the worker should not be able to produce valid transactions that have not been authorized by the user. Below we assume familiarity with our "plain" DPC construction (see Section 4.4).

The user will maintain (among other things) a key pair $(\mathsf{pk}_{\mathsf{SIG}}, \mathsf{sk}_{\mathsf{SIG}})$ for a randomizable signature scheme SIG (see Section 4.5.2). The public key $\mathsf{pk}_{\mathsf{SIG}}$ will be embedded in the user's public key apk and also be used to derive the serial numbers of records "owned" by apk. In contrast, the secret key $\mathsf{sk}_{\mathsf{SIG}}$ will not be a part of any data structures, and will *only* be used to *authorize* transactions by signing the cryptographic proofs produced by untrusted workers.

In more detail, we first describe how addresses and records are generated (also see summary in Fig. 4.10).

- **Addresses.** In Section 4.4 an address public key apk was a commitment to a secret key $\mathsf{sk}_{\mathsf{PRF}}$ for a pseudorandom function PRF. Now apk is a commitment to this same information *as well as* the public key of a key pair $(\mathsf{pk}_{\mathsf{SIG}}, \mathsf{sk}_{\mathsf{SIG}})$ for SIG. The corresponding address secret key ask consists of all the committed information and the commitment randomness.

- **Records.** The structure of a record, including how a record commitment is computed, is as in Section 4.4. However, a record's serial number sn is now derived in a different way: while previously $\mathsf{sn} := \mathsf{PRF}_{\mathsf{sk}_{\mathsf{PRF}}}(\rho)$ now we set $\mathsf{sn} := \mathsf{SIG.RandPk}(\mathsf{pp}_{\mathsf{SIG}}, \mathsf{pk}_{\mathsf{SIG}}, \mathsf{PRF}_{\mathsf{sk}_{\mathsf{PRF}}}(\rho))$ where ρ is the record's serial number nonce. Namely, while before serial numbers were outputs of a pseudorandom function keyed by $\mathsf{sk}_{\mathsf{PRF}}$, now they are randomizations of the authorization public key $\mathsf{pk}_{\mathsf{SIG}}$ when using suitable pseudorandomness.

Note that the foregoing new derivation of serial numbers does not break important security properties.

– *Unlinkability of serial numbers:* serial numbers of different records that share the same authorization public $\mathsf{pk}_{\mathsf{SIG}}$ are computationally indistinguishable. This follows rather directly from the fact sn, being a randomization of $\mathsf{pk}_{\mathsf{SIG}}$, does not reveal information (to efficient distinguishers) about $\mathsf{pk}_{\mathsf{SIG}}$ itself.
– *No double spending:* a user cannot "spend" (i.e., consume) $\mathbf{r}$ in two different transactions by revealing different serial numbers because r_{SIG} (and thus sn) is generated deterministically from $\mathbf{r}$. Since SIG is randomness-injective in $\mathsf{SIG.RandPk}$, sn is (computationally) unique to $\mathbf{r}$.

Having described the modified data structures of addresses and serial numbers, we now explain how a user can task a worker to produce the cryptographic proofs that need to be included in a transaction. For simplicity, in this high-level discussion we focus on the case where the transaction involves only one input (old) record $\mathbf{r}$ and one output (new) record $\mathbf{r}'$. In this case, the transaction contains a serial number sn (supposedly corresponding to $\mathbf{r}$), and a commitment cm' (supposedly corresponding to $\mathbf{r}'$).

Previously, the user had to generate a proof π_{e} that sn is consistent with $\mathbf{r}$, that cm' can be opened to $\mathbf{r}'$, and that the death and birth predicates of $\mathbf{r}$ and $\mathbf{r}'$ respectively are satisfied. Now the user can delegate to a worker the generation of the proof π_{e} because the modified derivation of apk and sn allows the user to communicate to the worker only $\mathbf{r}$, $\mathbf{r}'$ and a *part* of the address secret key of $\mathbf{r}$. Namely, the user sends to the worker only the pseudorandom function key $\mathsf{sk}_{\mathsf{PRF}}$ and the commitment randomness r_{pk}. Crucially, the user does not have to communicate to the worker the authorization secret key $\mathsf{sk}_{\mathsf{SIG}}$.

After receiving the proof π_{e} from the worker, the user uses the authorization secret key $\mathsf{sk}_{\mathsf{SIG}}$ to sign π_{e} (along with the instance that π_{e} attests to), and then randomizes the resulting signature σ to obtain $\hat{\sigma}$. The final transaction tx not only includes the serial number sn (consuming the old record), the commitment cm' (creating the new record), and π_{e} (attesting to the correct state transition) as before, but also includes $\hat{\sigma}$. Transaction verification involves checking the proof π_{e} and also checking that $\hat{\sigma}$ is valid with respect to the randomized public key sn.

This completes our high-level description of our delegable DPC scheme; see Section 4.11 for details.

	Plain DPC	Delegable DPC
Address secret key	$(\mathsf{sk}_{\mathsf{PRF}}, r_{\mathsf{pk}})$	$(\mathsf{sk}_{\mathsf{SIG}}, \mathsf{sk}_{\mathsf{PRF}}, r_{\mathsf{pk}})$
Address public key	$\mathsf{apk} := \mathsf{CM.Commit}\left(\begin{smallmatrix}\mathsf{pp}_{\mathsf{CM}},\\ \mathsf{sk}_{\mathsf{PRF}}\end{smallmatrix}; r_{\mathsf{pk}}\right)$	$\mathsf{apk} := \mathsf{CM.Commit}\left(\begin{smallmatrix}\mathsf{pp}_{\mathsf{CM}},\\ \mathsf{pk}_{\mathsf{SIG}}\|\mathsf{sk}_{\mathsf{PRF}}\end{smallmatrix}; r_{\mathsf{pk}}\right)$
Serial number derivation	$\mathsf{sn} \leftarrow \mathsf{PRF}_{\mathsf{sk}_{\mathsf{PRF}}}(\rho)$	1. $r_{\mathsf{SIG}} \leftarrow \mathsf{PRF}_{\mathsf{sk}_{\mathsf{PRF}}}(\rho)$ 2. $\mathsf{sn} \leftarrow \mathsf{SIG.RandPk}(\mathsf{pp}_{\mathsf{SIG}}, \mathsf{pk}_{\mathsf{SIG}}, r_{\mathsf{SIG}})$
Transaction construction	$\mathsf{tx} := ([\mathsf{sn}_i]_1^m, [\mathsf{cm}_j]_1^n, \mathsf{memo}, \star)$, where $\star := (\mathsf{st}_{\mathbf{L}}, \pi_{\mathsf{e}})$.	1. **Sign transaction contents:** a) $\sigma_i \leftarrow \mathsf{SIG.Sign}(\mathsf{pp}_{\mathsf{SIG}}, \mathsf{sk}_{\mathsf{SIG},i}, \mathbb{x}_{\mathsf{e}}\|\pi_{\mathsf{e}})$. b) $\hat{\sigma}_i \leftarrow \mathsf{SIG.RandSig}(\mathsf{pp}_{\mathsf{SIG}}, \sigma_i, r_{\mathsf{SIG},i})$. 2. $\mathsf{tx} := ([\mathsf{sn}_i]_1^m, [\mathsf{cm}_j]_1^n, \mathsf{memo}, \star)$, where $\star := (\mathsf{st}_{\mathbf{L}}, \pi_{\mathsf{e}}, [\hat{\sigma}_i]_1^m)$.
Transaction verification	Check that serial numbers do not appear on ledger, that the ledger state digest is valid, and that the NIZK proof verifies.	As in plain DPC, but additionally check that each **signature verifies:** $\mathsf{SIG.Verify}(\mathsf{pp}_{\mathsf{SIG}}, \mathsf{tx.sn}_i, \mathbb{x}_{\mathsf{e}}\|\pi_{\mathsf{e}}, \sigma_i) = 1$.

Figure 4.10: Summary of differences between plain DPC and delegable DPC (highlighted).

4.5.4 Threshold transactions and blind transactions

We explain how the delegable DPC scheme described above can be modified, in a straightforward way, to achieve additional features: *threshold transactions* or *blind transactions*.

Threshold transactions. A DPC scheme has threshold transactions if the power to authorize transactions can be vested unto any t out of n parties, for any desired choice of t and n (as opposed to a single user as discussed thus far, which corresponds to the special case of $t = n = 1$); see Fig. 4.12. Threshold transactions are useful in many settings, e.g., to enhance operational security by realizing two-factor authentication.

We can achieve threshold transactions by simply using, in our delegable DPC scheme, a randomizable signature scheme SIG that also supports *threshold key generation* and *threshold signing algorithms* [DF91]. Such a *threshold signature scheme* distributes signing ability among n parties such that at least t of them are needed to authorize a signature. Threshold key generation would then be used to create an address, and threshold signing would be used to authorize a transaction by signing the corresponding cryptographic proof.

Blind transactions. A DPC scheme has blind transactions if there is a way for a user to authorize a transaction without learning of its contents; see Fig. 4.13. Blind transactions, in conjunction with prior techniques [CGLMMM17], can be used to construct efficient lottery tickets and thereby probabilistic micropayments.

We can achieve blind transactions by simply using, in our delegable DPC scheme, a randomizable signature scheme SIG that has a *blind signing algorithm*, which can then be used for signing the relevant cryptographic proof in order to authorize a transaction.

Instantiating randomizable threshold and blind signatures. As we explain in Section 4.11.1, we construct randomizable signature schemes by modifying Schnorr signatures. To further construct threshold or blind randomizable signatures, it is enough to note that public key and signature randomization occurs *after* the public key or signature has been created. Thus one can use existing protocols for threshold key-generation and signing [SS01; NKDM03; Dod07], and blind signing [PS00; SJ99] to obtain public keys and signatures, and then use the algorithms from Section 4.11.1 to randomize these. A nice feature of this approach is that all these types of delegated transactions (regular, threshold, blind) cannot be distinguished from one another.

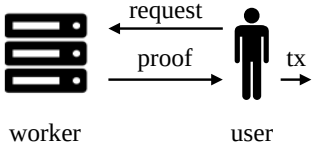

Figure 4.11: Delegable transactions.

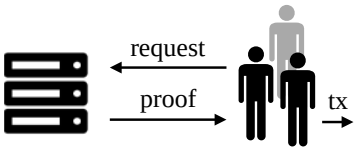

Figure 4.12: Threshold transactions.

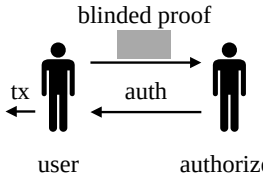

Figure 4.13: Blind transactions.

4.6 Applications

We describe example applications of DPC schemes, by showing how to "program" these within the records nano-kernel. We draw inspiration from current uses of smart contract systems (e.g., Ethereum), which largely focus on financial applications where privacy is an important goal. First, in Section 4.6.1 we describe how to enable users to privately create and transact with **custom assets** (expanding on Example 4.2.1). Second, in Section 4.6.2 we describe how to realize **private DEXs**, which enable users to privately trade assets while retaining custody of their assets. Finally, in Section 4.6.3, we describe how a central authority can issue assets with self-enforcing, *and updatable*, policies, and use these to realize regulation-friendly **private stablecoins**.

4.6.1 User-defined assets

One of the most basic applications of smart contract systems like Ethereum is the construction of *assets* (or *tokens*) that can be used for financial applications. For example, the Ethereum ERC20 specification [VB15] defines a general framework for such assets. These assets have two phases: asset minting (creation), and asset conservation (expenditure). We show below how to express such custom assets via the records nano-kernel.

We consider records whose payloads encode: an asset identifier id, the initial asset supply $\mathbb{v}$, a value v, and application-dependent data c (we will use this in Sections 4.6.2 and 4.6.3). We fix the birth predicate in all such records to be a *mint-or-conserve* function MoC that is responsible for asset minting and conservation. In more detail, the birth predicate MoC can be invoked in two modes, *mint mode* or *conserve mode*.

When invoked in mint mode, MoC creates the initial supply $\mathbb{v}$ of the asset in, say, a single output record, by deterministically deriving a fresh unique identifier id for the asset (see below for how), and storing the tuple $(\mathsf{id}, \mathbb{v}, \mathbb{v}, \bot)$ in the record's payload. The predicate MoC also ensures that the given transaction contains no input records or other output records (dummy records are allowed). If MoC is invoked in mint mode in other transactions, a *different* identifier id is created, ensuring that multiple assets can be distinguished even though anyone can use MoC as the birth predicate of a record.

When invoked in conserve mode, MoC inspects all records in a transaction whose birth predicates all equal MoC (i.e., all the transaction's user-defined assets) and whose asset identifiers all equal to the identifier of the current record. For these records it ensures that no new value is created: that is, the sum of the value across all output records is less than or equal to the sum of the value in all input records.

Below we provide pseudocode for MoC, making the informal discussion above more precise.

Mint-or-conserve predicate $\mathsf{MoC}(k, \mathsf{ldata}; \mathsf{mode})$ (mode is the private input of the predicate)

1. Parse ldata as $\begin{pmatrix} [\mathsf{cm}_i^{\mathsf{in}}]_1^2 & [\mathsf{apk}_i^{\mathsf{in}}]_1^2 & [\mathsf{payload}_i^{\mathsf{in}}]_1^2 & [\Phi_{\mathsf{d},i}^{\mathsf{in}}]_1^2 & [\Phi_{\mathsf{b},i}^{\mathsf{in}}]_1^2 & [\mathsf{sn}_i^{\mathsf{in}}]_1^2 & \mathsf{memo} \\ [\mathsf{cm}_j^{\mathsf{out}}]_1^2 & [\mathsf{apk}_j^{\mathsf{out}}]_1^2 & [\mathsf{payload}_j^{\mathsf{out}}]_1^2 & [\Phi_{\mathsf{d},j}^{\mathsf{out}}]_1^2 & [\Phi_{\mathsf{b},j}^{\mathsf{out}}]_1^2 & & \mathsf{aux} \end{pmatrix}$.
2. If $\mathsf{mode} = (\mathtt{mint}, \mathbb{v}, r)$, ensure that the first output record contains the initial supply of the asset:
   a) the index of the current output record is correct: $k = 1$.
   b) all other records are dummy: $\mathsf{payload}_1^{\mathsf{in}}.\mathsf{isDummy} = \mathsf{payload}_2^{\mathsf{in}}.\mathsf{isDummy} = \mathsf{payload}_2^{\mathsf{out}}.\mathsf{isDummy} = 1$.
   c) the asset identifier is derived correctly: $\mathsf{id} = \mathsf{CM.Commit}(\mathsf{pp}_{\mathsf{CM}}, \mathsf{sn}_1 \| \mathsf{sn}_2; r)$. (See explanation below.)
   d) the current output record's payload is correct: $\mathsf{payload}_1^{\mathsf{out}}.\mathsf{isDummy} = 0$ and $\mathsf{payload}_1^{\mathsf{out}} = (\mathsf{id}, \mathbb{v}, \mathbb{v}, \perp)$.
3. If $\mathsf{mode} = \mathtt{conserve}$, check that the value of the current asset is conserved:
   a) parse the current output record's payload $\mathsf{payload}_k^{\mathsf{out}}$ as $(\mathsf{id}^\star, \mathbb{v}^\star, v^\star, c^\star)$.
   b) for $i \in \{1, 2\}$, parse the i-th input record's payload $\mathsf{payload}_i^{\mathsf{in}}$ as $(\mathsf{id}_i^{\mathsf{in}}, \mathbb{v}_i^{\mathsf{in}}, v_i^{\mathsf{in}}, c_i^{\mathsf{in}})$.
   c) for $j \in \{1, 2\}$, parse the i-th output record's payload $\mathsf{payload}_j^{\mathsf{out}}$ as $(\mathsf{id}_j^{\mathsf{out}}, \mathbb{v}_j^{\mathsf{out}}, v_j^{\mathsf{out}}, c_j^{\mathsf{out}})$.
   d) initialize $v^{\mathsf{in}} := 0$ and $v^{\mathsf{out}} := 0$, representing the value of asset $\mathsf{id}^\star$ consumed and created (respectively).
   e) for $i \in \{1, 2\}$, if $\Phi_{\mathsf{b},i}^{\mathsf{in}} = \Phi_{\mathsf{b}}^\star$, $\mathsf{id}_i^{\mathsf{in}} = \mathsf{id}^\star$, $\mathsf{payload}_i^{\mathsf{in}}.\mathsf{isDummy} = 0$, set $v^{\mathsf{in}} := v^{\mathsf{in}} + v_i^{\mathsf{in}}$ and check that $\mathbb{v}_i^{\mathsf{in}} = \mathbb{v}^\star$.
   f) for $j \in \{1, 2\}$, if $\Phi_{\mathsf{b},j}^{\mathsf{out}} = \Phi_{\mathsf{b}}^\star$, $\mathsf{id}_j^{\mathsf{out}} = \mathsf{id}^\star$, $\mathsf{payload}_j^{\mathsf{out}}.\mathsf{isDummy} = 0$, set $v^{\mathsf{out}} := v^{\mathsf{out}} + v_j^{\mathsf{out}}$ and check that $\mathbb{v}_j^{\mathsf{out}} = \mathbb{v}^\star$.
   g) check that the value of asset $\mathsf{id}^\star$ is conserved: $v^{\mathsf{in}} = v^{\mathsf{out}}$.

Most of the lines above are self-explanatory, but for the line that derives a fresh unique identifier in the "mint" case (Step 2c), which deserves an explanation. Informally, the idea is to derive the identifier from the (globally unique) serial numbers of records consumed in the minting transaction. In more detail, we set the identifier to be a commitment to the serial numbers of consumed input records. To see why this works, first note that the commitment scheme's binding property guarantees that opening a commitment to two different messages is computationally difficult. Next, note that in our case these messages are the input records' serial numbers, and hence *are* different. Together, these facts imply that the identifier is globally unique (and hence non-repeating). A benefit of this method is that the commitment scheme's hiding property further guarantees that the identifier reveals no information about the underlying serial numbers, which in turn guarantees that the identifier hides all information about the initial minting transaction (given that r is random).

4.6.2 Decentralized exchanges

We describe how to use death predicates that enforce custom-access policies to build *privacy-preserving decentralized exchanges*. These allow users to exchange custom assets with strong privacy guarantees without requiring users to give up custody of these assets. We proceed by first providing background on centralized and decentralized exchanges. Then, we formulate desirable privacy properties for decentralized exchanges. Finally, we describe constructions that achieve these properties.

Motivation. Exchanging digital assets is a compelling use case of ledger-based systems. A straightforward method to exchange digital assets is via a *centralized exchange*: users entrust the exchange with custody of their assets via an on-chain transaction, and the exchange can then credit or debit assets to users' accounts according to off-chain trades without any on-chain activity; users can "exit" by requesting to withdraw assets, which generates another on-chain transaction

that transfers those assets from the exchange to the user. Examples of such exchanges include Coinbase [Coinbase] and Binance [Binance]. This architecture provides centralized exchanges with two attractive properties: (a) efficiency, namely, all trades occur in the exchange's off-chain database, resulting in low latency and high throughput for all users; and (b) privacy, namely, only the exchange knows the details of individual trades, and only asset deposits and withdrawals require on-chain activity; this activity can further be concealed by using private (Zerocash-style) transactions to realize deposits/withdrawals. However, this centralized architecture has a serious drawback: having given up custody of their assets, users are exposed to the risk of security breaches, fraud, or front-running at the exchange. These risks are not hypothetical: users have lost funds deposited at centralized exchanges [PA14; De18; Zha18; Cim18].

In light of the above, *decentralized exchanges* (DEXs) have been proposed as an alternative method for exchanging digital assets that enable users to retain custody of their assets. However, existing DEX constructions have poor efficiency and privacy guarantees. Below we describe how we can provide strong privacy for DEXs. (We leave improving the efficiency of DEXs to future work.)

DEX architectures. A DEX is a ledger-based application that enables users to trade digital assets without giving up custody of these assets to a third party. There are different DEX architectures with different trade-offs; see [Pro18] for a survey. In the following, we consider DEX architectures where the exchange has no state or maintains its state off-chain.[10] There are two main categories of such DEXs:

- *Intent-based DEX.* The DEX maintains an index, which is a table where makers publish their intention to trade (say, a particular asset pair) without committing any assets. A taker interested in a maker's intention to trade can directly communicate with the maker to agree on terms. They can jointly produce a transaction for the trade, to be broadcast for on-chain processing. An example of such a DEX is AirSwap [AirSwap].

  An attractive feature of intent-based DEXs is that they reduce exposure to front-running because the information required for front-running (like prices or identities of the involved parties) has been finalized by the time the transaction representing the trade is broadcast for processing. Note that the aforementioned lack of information also makes it difficult for the market to discover appropriate exchange rates because listings in the index cannot directly be linked with completed transactions.

- *Order-based DEX.* The DEX maintains an order book, which is a table where makers can publish orders by committing the funds for those orders up front. A taker can then interact with the order book to fill orders. In an *open-book DEX*, the taker manually picks an order from the order book, while in a *closed-book DEX*, the taker is matched off-chain with a maker's offer by the order book operator. An example of an open-book DEX is Radar Relay [Radar], and an example of a closed-book DEX is Paradex [Paradex].

  Note that order books (which are typically public) give more information about market activity than indexes, and hence enable better price discovery. However, existing constructions of order-based

[10] This is in contrast to DEX architectures that involve, say, a smart contract that stores on-chain the standing orders of all users.

DEXs also allow other parties to link a standing order with a transaction that fills the order before the transaction is finalized, enabling them to front-run the order. Which parties can front-run depends on the kind of order-based DEX: in the open-book variant, anyone can front-run, while in the closed-book variant, only the order book operator can front-run (as it is the sole entity that can invoke the trade smart-contract).

The architectures described above offer different trade-offs with respect to market price discovery and front-running exposure, and hence can be useful in different scenarios.

Privacy shortcomings and goals. While the foregoing DEX architectures offer attractive security and functionality, they do *not* provide strong privacy guarantees, as we now explain. First, each transaction reveals information about the corresponding trade, such as the assets and amounts that were exchanged. Prior work [BDJT17; BBDJLZ17; EMC19; Dai+20] shows that such leakage enables front-running that harms user experience and market transparency, and proposes mitigations that, while potentially useful, do not provide strong privacy guarantees. Even if one manages to hide these trade details, transactions in existing DEXs *also* reveal the identities of transacting parties. Onlookers can use this information to extract trading patterns and frequencies of users. This reduces the privacy of users, violates the fungibility of assets, and increases exposure to front-running, because onlookers can use the aforementioned patterns to infer when particular assets are being traded.

These shortcomings motivate the following privacy goals for DEXs. Throughout, we assume that an order is defined by a pair of assets (that are to be exchanged), and their exchange rates.

1. *Trade confidentiality:* No efficient adversary $\mathcal{A}$ should be able to learn the trade details of completed or cancelled trades. That is, a transaction that completes or cancels a trade should not reveal to $\mathcal{A}$ the asset pairs or amounts involved in the trade.
2. *Trade anonymity:* No efficient adversary $\mathcal{A}$ should be able to learn the identities of parties involved in a trade. That is, a transaction that completes or cancels a trade should not reveal to $\mathcal{A}$ any information about the maker or taker of the trade.

A protocol that achieves trade confidentiality and trade anonymity against an adversary $\mathcal{A}$ is secure against front-running by $\mathcal{A}$. The flip-side of this is that $\mathcal{A}$ cannot easily discover the rates used in successful trades, leading to poorer visibility into the trading market. We now describe constructions of intent-based and order-based DEXs that achieve trade confidentiality and anonymity.[11]

Record format. Recall from Section 4.6.1 that records representing units of an asset have payloads of the form $(\mathsf{id}, \mathbb{v}, v, c)$, where id is the asset identifier, $\mathbb{v}$ is the initial asset supply, v is the asset amount, and c is arbitrary auxiliary information. In the following, we make use of records that, in addition to the mint-or-conserve birth predicate MoC, have an *exchange-or-cancel* death predicate EoC described next. Informally, EoC allows a record $\mathbf{r}$ to be consumed either by exchanging it for $v^\star$ units of an asset with birth predicate $\Phi^\star_{\mathsf{b}}$ and identifier $\mathsf{id}^\star$ ($\mathsf{id}^\star$, $\Phi^\star_{\mathsf{b}}$ and $v^\star$ are specified in c), or by

[11] Throughout, we assume that users interact with index or-order book operators via anonymous channels. (If this is not the case, operators can use network information to link users across different interactions regardless of any cryptographic solutions used.).

"cancelling" the exchange and instead sending new records with $\mathbf{r}$'s asset identifier to an address $\mathsf{apk}^\star$ (also specified in c). The information required for the exchange includes the asset's birth predicate in addition to its identifier, as it enables users to interact with assets that have birth predicate different from MoC (such as the stablecoins in Section 4.6.3). The predicate is described below.

Exchange-or-cancel predicate $\mathsf{EoC}(k, \mathsf{ldata}; \mathsf{mode})$ (mode is the private input for the predicate.)

1. Parse ldata as $\begin{pmatrix} [\mathsf{cm}_i^{\mathsf{in}}]_1^2 & [\mathsf{apk}_i^{\mathsf{in}}]_1^2 & [\mathsf{payload}_i^{\mathsf{in}}]_1^2 & [\Phi_{\mathsf{d},i}^{\mathsf{in}}]_1^2 & [\Phi_{\mathsf{b},i}^{\mathsf{in}}]_1^2 & [\mathsf{sn}_i^{\mathsf{in}}]_1^2 & \mathsf{memo} \\ [\mathsf{cm}_j^{\mathsf{out}}]_1^2 & [\mathsf{apk}_j^{\mathsf{out}}]_1^2 & [\mathsf{payload}_j^{\mathsf{out}}]_1^2 & [\Phi_{\mathsf{d},j}^{\mathsf{out}}]_1^2 & [\Phi_{\mathsf{b},j}^{\mathsf{out}}]_1^2 & & \mathsf{aux} \end{pmatrix}$.
2. Recall that $k \in \{1, 2\}$ is the index of the current input record. Let $l \in \{1, 2\}$ denote the index of the other input record.
   (If $k = 1$ then set $l := 2$; if instead $k = 2$ then set $l := 1$.)
3. Parse the current input record's payload $\mathsf{payload}_k^{\mathsf{in}}$ as $(\mathsf{id}_k^{\mathsf{in}}, \mathbb{v}_k, v_k^{\mathsf{in}}, c_k^{\mathsf{in}})$, and the application data c_k^{in} as $(\Phi_{\mathsf{b},k}^\star, \mathsf{id}_k^\star, v_k^\star, \mathsf{apk}_k^\star)$.
4. Parse the other input record's payload $\mathsf{payload}_l^{\mathsf{in}}$ as $(\mathsf{id}_l^{\mathsf{in}}, \mathbb{v}_l, v_l^{\mathsf{in}}, c_l^{\mathsf{in}})$, and the application data c_l^{in} as $(\Phi_{\mathsf{b},l}^\star, \mathsf{id}_l^\star, v_l^\star, \mathsf{apk}_l^\star)$.
5. If $\mathsf{mode} = \mathtt{exch}$, ensure that the assets are correctly exchanged, by checking the following.
   a) the input records are not dummy: $\mathsf{payload}_1^{\mathsf{in}}.\mathsf{isDummy} = \mathsf{payload}_2^{\mathsf{in}}.\mathsf{isDummy} = 0$.
   b) the conditions of the trade are satisfied:
      i. the current input record has the expected identifier, birth predicate, and value: $\Phi_{\mathsf{b},k}^{\mathsf{in}} = \Phi_{\mathsf{b},l}^\star$, $\mathsf{id}_k^{\mathsf{in}} = \mathsf{id}_l^\star$, and $v_k^{\mathsf{in}} = v_l^\star$.
      ii. the other input record has the expected identifier, birth predicate, and value: $\Phi_{\mathsf{b},l}^{\mathsf{in}} = \Phi_{\mathsf{b},k}^\star$, $\mathsf{id}_l^{\mathsf{in}} = \mathsf{id}_k^\star$, and $v_l^{\mathsf{in}} = v_k^\star$.
      iii. the output records' birth predicates are correctly swapped: $\Phi_{\mathsf{b},1}^{\mathsf{in}} = \Phi_{\mathsf{b},2}^{\mathsf{out}}$ and $\Phi_{\mathsf{b},2}^{\mathsf{in}} = \Phi_{\mathsf{b},1}^{\mathsf{out}}$.
      iv. the output records have the correct asset identifier, initial supply, and value: $\mathsf{payload}_1^{\mathsf{out}} = (\mathsf{id}_2^{\mathsf{in}}, \mathbb{v}_2, v_2^{\mathsf{in}}, \bot)$ and $\mathsf{payload}_2^{\mathsf{out}} = (\mathsf{id}_1^{\mathsf{in}}, \mathbb{v}_1, v_1^{\mathsf{in}}, \bot)$.
      v. the output records are addressed correctly: $\mathsf{apk}_k^{\mathsf{out}} = \mathsf{apk}_k^\star$ and $\mathsf{apk}_l^{\mathsf{out}} = \mathsf{apk}_l^\star$.
6. Else if $\mathsf{mode} = \mathtt{cancel}$, ensure that the trade is cancelled by checking that the id_k-value is transferred to the specified "redemption" address public key $\mathsf{apk}_k^\star$, by checking the following.
   a) the current input record is non-dummy: $\mathsf{payload}_k.\mathsf{isDummy} = 0$.
   b) the other input record is dummy: $\mathsf{payload}_l.\mathsf{isDummy} = 1$.
   c) the output records are custom assets with identifier $\mathsf{id}_k^{\mathsf{in}}$:
      i. the output records have the correct birth predicate: $\Phi_{\mathsf{b},1}^{\mathsf{out}} = \Phi_{\mathsf{b},2}^{\mathsf{out}} = \Phi_{\mathsf{b},k}^{\mathsf{in}}$.
      ii. the output records have the correct asset identifier and initial supply: $\mathsf{payload}_1^{\mathsf{out}} = (\mathsf{id}_k^{\mathsf{in}}, \mathbb{v}_k, v_1^{\mathsf{out}}, \bot)$ and $\mathsf{payload}_2^{\mathsf{out}} = (\mathsf{id}_k^{\mathsf{in}}, \mathbb{v}_k, v_2^{\mathsf{out}}, \bot)$.
   d) the output records preserve $\mathsf{id}_k^{\mathsf{in}}$-value: $v_1^{\mathsf{out}} + v_2^{\mathsf{out}} = v_k^{\mathsf{in}}$.
   e) the address public key of the output records is correct: $\mathsf{apk}_1^{\mathsf{out}} = \mathsf{apk}_2^{\mathsf{out}} = \mathsf{apk}_k^\star$.

The case of intent-based DEXs. We describe an intent-based DEX that hides all information about orders and transacting parties.

1. A maker M can publish to the index an intention to trade, which is a tuple $(\mathsf{id}_A, \mathsf{id}_B, \mathsf{pk}_{\mathsf{M}})$ to be interpreted as: "I want to buy assets with identifier id_B in exchange for assets with identifier id_A. Please contact me using the encryption public key pk_{M} if you would like to discuss the terms."
2. A taker T who is interested in this offer can use pk_{M} to privately communicate with M and agree on the terms of the trade (the form of communication is irrelevant). If T and M do not reach an agreement, then T can always pursue other entries in the index. So suppose that T and M do

reach an agreement. For the sake of example, T will give 10 units of asset id_B to M and will receive 5 units of asset id_A from M.

3. The taker T creates a new record $\mathbf{r}$ with payload $(\mathsf{id}_B, \mathbb{v}_B, 10, c)$ for auxiliary data $c = (\mathsf{id}_A, 5, \mathsf{apk}_{\mathsf{new}})$, and with death predicate EoC. Then T sends $\mathbf{r}$ (along with the information necessary to redeem $\mathbf{r}$) to M.
4. If M possesses a record worth 5 units of asset id_A, he can use T's message to construct a DPC transaction that completes the exchange by consuming $\mathbf{r}$ and by producing appropriate new records for M and T. (This step deviates from existing intent-based DEXs in that it is the *maker* that broadcasts the trade transaction.)

The record $\mathbf{r}$ produced by the taker T can be redeemed by M only via an appropriate record in exchange. If M does not possess such a record, T can cancel the trade (at any time) and retrieve his funds by satisfying the "cancel" branch of the predicate EoC (which requires knowing the secret key corresponding to $\mathsf{apk}_{\mathsf{new}}$).

Note that regardless of whether the trade was successful or not, this protocol achieves trade anonymity and trade confidentiality against all parties (including the index operator). Indeed, the only information revealed in the final transaction is that some records were consumed and others created; no information is revealed about M, T, the assets involved in the trade (id_A and id_B), or the amounts exchanged.

The case of order-based DEXs. We describe private order-based DEXs, with open or closed books.

- *Open-book DEX:* The variant below hides all information about M and T, but reveals the assets and amounts involved. This implies achieving trade anonymity but not trade confidentiality.

  For the sake of example, assume again that the maker M will trade 5 units of asset id_A for 10 units of asset id_B. M constructs a record $\mathbf{r}$ with payload $(\mathsf{id}_B, \mathbb{v}_B, v = 10, c = (\mathsf{id}_A, 5, \mathsf{apk}))$ and death predicate EoC. He uses this to construct an order $o = (\mathbf{r}, \mathsf{info})$ consisting of the record and the information necessary to consume it, and publishes o to the order book. An interested taker T can then construct and publish a transaction tx that consumes $\mathbf{r}$ and creates new records with the appropriate values and asset identifiers.

  The transaction tx hides information about the maker M and taker T, but because it reveals $\mathbf{r}$'s serial number, it can be linked with its originating order o. This allows onlookers to learn the assets and amounts of tx. Hence, this protocol achieves trade anonymity, but not trade confidentiality.

- *Closed-book DEX:* The variant below hides all order information from everyone but the order book operator. Hence it achieves trade anonymity and confidentiality against everyone but the order book operator.

  The maker M creates a record $\mathbf{r}$ as above, and sends the record and its consumption information info to the order book. The order book does not publish these; it publishes only the terms of the order. Takers can publish orders of their own, and if two orders match then the order book operator constructs a transaction tx that consumes both records and produces new records, completing

the order. At no point does either party surrender custody of their funds, thus preserving the self-custodial nature of the exchange protocol.

The foregoing achieves trade anonymity and confidentiality against everyone but the order book operator because only the order book operator learns the details of the records consumed by the transaction tx, and tx itself (which once published anyone can see) does not reveal any information about these records. As a consequence, this protocol also protects against front-running by everyone but the order book.

Note that in our protocol, the maker acts as the taker's counterparty (and vice versa), while in non-private closed-book DEXs, only the order book operator can act as the counterparty for both the maker and the taker. Our protocol can be modified to support such a flow by straightforward modifications to EoC.

Operator fees. In the foregoing we have omitted a discussion of fees due to the operators of DEX infrastructure (such as index or order book operators). Support for such fees can be achieved, in a straightforward way, by the following small modifications to the exchange-or-cancel predicate EoC. First, one would need to increase the number of output records of DPC transactions to $n = 3$; the third record would be used to pay fees to the operator. Second, one would have to decide how these fees are calculated. This can be done, e.g., by hardcoding a fee percentage into the predicate or by allowing users to specify fees that they are willing to pay.

Remark 4.6.1 (preventing a denial-of-funds attack)**.** In the foregoing protocols, the maker M could refuse to provide the taker T with information about its output record, thus denying T the ability to consume its exchanged record. A simple approach to prevent this is to modify the exchange-or-cancel predicate to additionally enforce that the memorandum field of the created transaction contains an encryption of the output record information under a public key specified by T.

4.6.3 Stablecoins and centrally-managed assets

Recently there has been growing interest in custom assets that are managed by a central authority. These include stablecoins, which are assets whose value relative to an another is *fixed* (see [Har18] for an overview). Centrally-managed assets are more compatible with regulations like taxes or blacklists, because the central authority can enforce monetary policies that follow these regulations. Indeed, existing stablecoins like the Gemini dollar [Gemini] and the Paxos standard [Paxos] have mechanisms for reversing transactions or freezing funds in response to legal rulings. In this section, we show how to construct *private* centrally-managed assets that support arbitrary, and updatable, policies issued by the central authority; this in particular shows how to create and manage policies for *private stablecoins*. We stress that the ideas described below are compatible with applications that reason about other custom assets. For example, one can use DEXs from Section 4.6.2 to exchange units of a private stablecoin with units of any other user-defined asset (like one from Section 4.6.1).

We enforce policies by extending the mint-or-conserve predicate MoC from Section 4.6.1 into a *mint-or-enforce* predicate MoE_{Π} whose "enforce" mode enforces a desired policy Π. In more detail, say that a central authority A wishes to issue an asset satisfying policy Π (initially). To do

so, A generates a signature public key pk_A, and then invokes MoE_Π in *mint* mode. In this mode, MoE_Π, like MoC, generates the asset identifier id and creates the initial supply $\mathbb{v}$ of the asset in a single output record whose payload stores the tuple $(\mathsf{id}, \mathbb{v}, \mathbb{v}, \bot)$. Unlike MoC, MoE_Π binds id not only to the serial numbers of input records (to achieve uniqueness), but also to the public key pk_A that authorized Π. This means that, when receiving payments in such assets, the recipient can immediately deduce the asset's identifier and (authorized) policy.

In a transaction with multiple records, policies are applied and updated by the *enforce* mode of MoE_Π. In this mode, MoE_Π ensures that the new record's payload stores $(\mathsf{id}, \mathbb{v}, v, c)$, and that the policy Π is satisfied. To update a record $\mathbf{r}$ having policy Π to a record $\mathbf{r}'$ having policy Π', one can create a transaction that consumes $\mathbf{r}$ and creates $\mathbf{r}'$ such that $\mathbf{r}'$ has birth predicate $\mathsf{MoE}_{\Pi'}$. To ensure that this update is authorized by A, $\mathsf{MoE}_{\Pi'}$ checks that a signature over Π' with respect to pk_A has been provided, and that id has been correctly derived from pk_A. These checks ensure that every record with identifier id only has authorized policies.

Below we provide pseudocode for MoE_Π.

Mint-or-enforce predicate $\mathsf{MoE}_\Pi(k, \mathsf{ldata}; \mathsf{mode})$ (mode is the private input of the predicate)

1. Parse ldata as $\begin{pmatrix} [\mathsf{cm}_i^{\mathsf{in}}]_1^2 & [\mathsf{apk}_i^{\mathsf{in}}]_1^2 & [\mathsf{payload}_i^{\mathsf{in}}]_1^2 & [\Phi_{\mathsf{d},i}^{\mathsf{in}}]_1^2 & [\Phi_{\mathsf{b},i}^{\mathsf{in}}]_1^2 & [\mathsf{sn}_i^{\mathsf{in}}]_1^2 & \mathsf{memo} \\ [\mathsf{cm}_j^{\mathsf{out}}]_1^2 & [\mathsf{apk}_j^{\mathsf{out}}]_1^2 & [\mathsf{payload}_j^{\mathsf{out}}]_1^2 & [\Phi_{\mathsf{d},j}^{\mathsf{out}}]_1^2 & [\Phi_{\mathsf{b},j}^{\mathsf{out}}]_1^2 & & \mathsf{aux} \end{pmatrix}$.
2. If $\mathsf{mode} = (\texttt{mint}, \mathbb{v}, r, \mathsf{pk}_{\mathsf{SIG}}, \sigma_\Pi)$, ensure that the first output record contains the initial supply of the asset:
   a) the index of the current output record is correct: $k = 1$.
   b) all other records are dummy: $\mathsf{payload}_1^{\mathsf{in}}.\mathsf{isDummy} = \mathsf{payload}_2^{\mathsf{in}}.\mathsf{isDummy} = \mathsf{payload}_2^{\mathsf{out}}.\mathsf{isDummy} = 1$.
   c) the asset identifier is derived correctly: $\mathsf{id} = \mathsf{CRH}(\mathsf{pp}_{\mathsf{CRH}}, \mathsf{CM.Commit}(\mathsf{pp}_{\mathsf{CM}}, \mathsf{sn}_1 \| \mathsf{sn}_2; r) \| \mathsf{pk}_{\mathsf{SIG}})$.
   d) the policy Π is authorized by $\mathsf{pk}_{\mathsf{SIG}}$: $\mathsf{SIG.Verify}(\mathsf{pp}_{\mathsf{SIG}}, \mathsf{pk}_{\mathsf{SIG}}, \Pi, \sigma_\Pi) = 1$.
   e) the current output record's payload is correct: $\mathsf{payload}_1^{\mathsf{out}}.\mathsf{isDummy} = 0$ and $\mathsf{payload}_1^{\mathsf{out}} = (\mathsf{id}, \mathbb{v}, \mathbb{v}, c = \bot)$.
3. If $\mathsf{mode} = (\texttt{enforce}, \rho, \mathsf{pk}_{\mathsf{SIG}}, \sigma_\Pi)$, check that the policy Π is enforced:
   a) parse the current output record's payload $\mathsf{payload}_k^{\mathsf{out}}$ as $(\mathsf{id}^\star, \mathbb{v}^\star, v^\star, c)$.
   b) check that $\mathsf{pk}_{\mathsf{SIG}}$ is valid for the asset: $\mathsf{id}^\star = \mathsf{CRH}(\mathsf{pp}_{\mathsf{CRH}}, \rho \| \mathsf{pk}_{\mathsf{SIG}})$.
   c) check that the policy Π is authorized under pk: $\mathsf{SIG.Verify}(\mathsf{pp}_{\mathsf{SIG}}, \mathsf{pk}_{\mathsf{SIG}}, \Pi, \sigma_\Pi) = 1$.
   d) check that the policy Π is satisfied: $\Pi(k, \mathsf{ldata}) = 1$.

By way of example, we now show how a central authority A can use the mint-or-enforce predicate to construct a stablecoin that enforces a *blacklisting* (in addition to the default value-conservation policy). Namely, if an address is on a blacklist B of addresses, the address is not allowed to participate in transactions. To do so, A follows the above procedure to construct and publish a mint-or-enforce predicate MoE_{Π_B} implementing a policy Π_B that inspects the address public keys of consumed records, and ensures that none of them are in B. Now suppose that later on A wishes to update B into a new blacklist B' that includes a new address apk. It does so by publishing a corresponding updated predicate $\mathsf{MoE}_{\Pi_{B'}}$ for this new blacklist, and users can use the above update mechanism to move their records from policy Π_B to policy $\Pi_{B'}$. Now, any funds stored at the newly-blacklisted address apk cannot be moved to the new policy.

4.7 Implementation strategy

The straightforward approach to implement our construction of a DPC scheme (described in Section 4.4) is to instantiate the proof system via a simulation-extractable zkSNARK (e.g., [GM17]) and then select the other cryptographic building blocks so that the circuit (more precisely, constraint system) for deciding the NP relation $\mathcal{R}_{\mathsf{e}}$ has as small a size as possible. While the straightforward approach sounds promising, closer inspection reveals significant costs that we need to somehow reduce. In this section we discuss, in a "problem and solution" format, the challenges that we encountered and how we addressed them. (The implementation strategy for plain DPC schemes directly ports over to delegable DPC schemes so we do not discuss them.)

Problem 1: universality is expensive. The NP relation $\mathcal{R}_{\mathsf{e}}$ involves checking arbitrary predicates, which means that one must rely on proof systems for *universal* computations. However, checking universal computations via state-of-the-art zkSNARKs involves expensive tools for universal circuits/machines [BCGTV13; BCTV14; WSRBW15; BCTV17]. These tools would not only yield an expensive solution but would also penalize users who only produce transactions that attest to simple inexpensive predicates, because these users would have to incur the costs of using these "heavy duty" proof systems.

Solution 1: recursive proof verification. We address this problem by relying on one layer of *recursive proof composition* [Val08; BCCT13]. Instead of tasking $\mathcal{R}_{\mathsf{e}}$ with checking satisfiability of general predicates, we only task it with checking *succinct proofs* attesting to this. Checking succinct proofs is a (relatively) inexpensive computation that is universal for *all* predicates, which can be "hardcoded" in $\mathcal{R}_{\mathsf{e}}$. Crucially, since the "outer" succinct proofs produced for $\mathcal{R}_{\mathsf{e}}$ do not reveal information about the "inner" succinct proofs attesting to predicates' satisfiability (thanks to zero knowledge), the inner succinct proofs do *not* have to hide what predicate was checked, removing the need for expensive universal circuits; in fact, inner proofs do not even have to be zero knowledge. Rather, these inner succinct proofs can be for NP relations *tailored* to the computations needed by particular birth and death predicates. Furthermore, this approach ensures that a user only has to incur the cost of proving satisfiability of the specific predicates involved in his transactions, regardless of the complexity of predicates used by other users in their transactions.

In more detail, taking the case of one input and one output record as an example, we modify $\mathsf{DPC.Execute}$ to additionally take as input SNARK proofs π_{d} and π_{b}, and also modify the NP relation $\mathcal{R}_{\mathsf{e}}$ so that, instead of directly checking that Φ_{d} and Φ_{b} are satisfied, it instead checks that π_{d} *and* π_{b} *attest to the satisfaction of* Φ_{d} *and* Φ_{b}. That is, $\mathcal{R}_{\mathsf{e}}$ checks that $\mathsf{NIZK.Verify}(\mathsf{pp}_{\Phi_{\mathsf{d}}}, \mathbb{x}_{\mathsf{e}}, \pi_{\mathsf{d}}) = 1$ and $\mathsf{NIZK.Verify}(\mathsf{pp}_{\Phi_{\mathsf{b}}}, \mathbb{x}_{\mathsf{e}}, \pi_{\mathsf{b}}) = 1$, where $\mathsf{pp}_{\Phi_{\mathsf{d}}}$ are public parameters for the NP relation $\mathcal{R}_{\Phi_{\mathsf{d}}} := \{(\mathbb{x}_{\mathsf{e}}, \mathbb{w}_{\mathsf{e}}) \text{ s.t. } \Phi_{\mathsf{d}}(\mathbb{x}_{\mathsf{e}}, \mathbb{w}_{\mathsf{e}}) = 1\}$ and similarly for Φ_{b}. The public parameters $\mathsf{pp}_{\Phi_{\mathsf{d}}}$ and $\mathsf{pp}_{\Phi_{\mathsf{b}}}$ are stored in the record, in place of (a description of) the predicates.[12]

[12] More precisely, to verify a proof for a predicate Φ, the proof verifier does not need to read all of pp_{Φ}, which has size $O_\lambda(|\Phi|)$ in some zkSNARKs (i.e., it is large). Rather, the proof verifier only needs to read $O_\lambda(|\mathbb{x}_{\mathsf{e}}|)$ bits of pp_{Φ}, which are collectively known as the *verification key*. The record would then store this verification key (or a hash thereof) rather than pp_{Φ}.

More generally, we modify $\mathsf{DPC.Execute}$ to additionally take as input SNARK proofs $[\pi_{\mathsf{d},i}]_1^m$ attesting that the old records' death predicates are satisfied and SNARK proofs $[\pi_{\mathsf{b},j}]_1^n$ attesting that the new records' birth predicates are satisfied. Moreover, we similarly modify the NP relation $\mathcal{R}_{\mathsf{e}}$ to check that these proofs are valid, instead of directly checking that the relevant predicates are satisfied.

In sum, $\mathcal{R}_{\mathsf{e}}$ is not tasked with checking general predicates. Instead, it merely has to check SNARK proofs, a fixed computation of size $O_\lambda(m+n)$. Separately, a user wishing to prove that a predicate Φ is satisfied will invoke a SNARK on an NP statement of size $|\Phi|$ (tailored for Φ).[13] The approach described so far, however, hides additional costs that we need to overcome.

Problem 2: recursion is expensive. Recursive proof composition has so far been empirically demonstrated for pairing-based SNARKs [BCTV17], whose proofs are extremely short and cheap to verify. We thus focus our attention on these, and explain the efficiency challenges that we must overcome in our setting.

Recall that pairings are instantiated via elliptic curves of small embedding degree. If we instantiate a SNARK's pairing via an elliptic curve E defined over a prime field $\mathbb{F}_q$ and having a subgroup of large prime order r, then (a) the SNARK supports NP relations $\mathcal{R}$ expressed as arithmetic circuits over $\mathbb{F}_r$, while (b) proof verification involves arithmetic operations over $\mathbb{F}_q$. This means that we need to express $\mathcal{R}_{\mathsf{e}}$ via arithmetic circuits over $\mathbb{F}_r$. In turn, since the SNARK verifier is part of $\mathcal{R}_{\mathsf{e}}$, this means that we need to also express the verifier via an arithmetic circuit over $\mathbb{F}_r$, which is problematic because the verifier's "native" operations are over $\mathbb{F}_q$. Simulating $\mathbb{F}_q$ operations via $\mathbb{F}_r$ operations introduces significant overheads, and picking E such that $q = r$, in order to avoid simulation, is impossible [BCTV17].

Prior work thus suggests using *multiple* curves [BCTV17], such as a two-cycle of pairing-friendly elliptic curves, that is, two prime-order curves E_1 and E_2 such that the prime size of one's base field is the prime order of the other's group, and orchestrating SNARKs based on these so that fields always "match up". Unfortunately, known curves with these properties are inefficient at 128 bits of security [BCTV17; CCW19].

Solution 2: tailored set of curves. In our setting we merely need "a proof of a proof", with the latter proof not itself depending on further proofs.

This implies that we do not actually need a cycle of pairing-friendly elliptic curves (which enables recursion of arbitrary depth), but rather only a "two-chain" of two curves E_1 and E_2 such that the size of the base field of E_1 is the size of the prime order subgroup of E_2. We can use the Cocks–Pinch method [FST10] to set up such a bounded recursion [BCTV17]. We now elaborate on this.

First we pick a pairing-friendly elliptic curve E_1 that not only is suitable for 128 bits of security according to standard considerations (involving, e.g., its embedding degree and the ratio of the sizes of its base field and prime order group) but, moreover, is compatible with efficient SNARK provers

[13] An additional benefit of each predicate Φ having its own public parameters pp_Φ is flexible trust: users are not obliged to trust parameters used in each others' transactions and, moreover, if some parameters are known to be compromised, predicates can safely refuse to interact with records associated with them. We view this *isolation mechanism* as a novel and valuable feature in practice.

in *both* levels of the recursion. Namely, letting p be the prime order of the base field and r the prime order of the group, we need that *both* $\mathbb{F}_r$ and $\mathbb{F}_p$ have multiplicative subgroups whose orders are large powers of 2. The condition on $\mathbb{F}_r$ ensures efficient proving for SNARKs over E_1, while the condition on $\mathbb{F}_p$ ensures efficient proving for SNARKs that verify proofs over E_1. In light of the above, we set E_1 to be E_{BLS}, a curve from the Barreto–Lynn–Scott (BLS) family [BLS02; CLN11] with embedding degree 12. This family not only enables parameters that conservatively achieve 128 bits of security, but also enjoys properties that facilitate very efficient implementation [AFKMR12]. We ensure that both $\mathbb{F}_r$ and $\mathbb{F}_p$ have multiplicative subgroups of order 2^α for $\alpha \geq 40$, by choosing the parameter x of the BLS family to satisfy $x \equiv 1 \bmod 3 \cdot 2^\alpha$; indeed, for such a choice of x both $r(x) = x^4 - x^2 + 1$ and $p(x) = (x-1)^2 r(x)/3 + x$ are divisible by 2^α. This also ensures that $x \equiv 1 \bmod 3$, which ensures that there are efficient towering options for the relevant fields [Cos12].

Next we use the Cocks–Pinch method to pick a pairing-friendly elliptic curve $E_2 = E_{\mathsf{CP}}$ over a field $\mathbb{F}_q$ such that the curve group $E_{\mathsf{CP}}(\mathbb{F}_q)$ contains a subgroup of prime order p (the size of E_{BLS}'s base field). Since the method outputs a prime q that has about $2\times$ more bits than the desired p, and in turn p has about $1.5\times$ more bits than r (due to properties of the BLS family), we only need E_{CP} to have embedding degree 6 in order to achieve 128 bits of security (as determined from the guidelines in [FST10]).

In sum, proofs of predicates' satisfiability are produced via a SNARK over E_{BLS}, and proofs for the NP relation $\mathcal{R}_{\mathsf{e}}$ are produced via a zkSNARK over E_{CP}. The matching fields between the two curves ensure that the former proofs can be efficiently verified.

Problem 3: Cocks–Pinch curves are costly. While the curve E_{CP} was chosen to facilitate efficient checking of proofs over E_{BLS}, the curve E_{CP} is at least $2\times$ more expensive (in time and space) than E_{BLS} simply because E_{CP}'s base field has about twice as many bits as E_{BLS}'s base field. Checks in the NP relation $\mathcal{R}_{\mathsf{e}}$ that are not directly related to proof checking are now unnecessarily carried over a less efficient curve.

Solution 3: split relations across two curves. We split $\mathcal{R}_{\mathsf{e}}$ into two NP relations $\mathcal{R}_{\mathsf{BLS}}$ and $\mathcal{R}_{\mathsf{CP}}$ (see Fig. 4.14), with the latter containing just the proof check and the former containing all other checks. We can then use a zkSNARK over the curve E_{BLS} (an efficient curve) to produce proofs for $\mathcal{R}_{\mathsf{BLS}}$, and a zkSNARK over E_{CP} (the less efficient curve) to produce proofs for $\mathcal{R}_{\mathsf{CP}}$. This approach significantly reduces the running time of $\mathsf{DPC.Execute}$ (producing proofs for the checks in $\mathcal{R}_{\mathsf{BLS}}$ is more efficient over E_{BLS} than over E_{CP}), at the expense of a modest increase in transaction size (a transaction now includes a zkSNARK proof over E_{BLS} in addition to a proof over E_{CP}). An important technicality that must be addressed is that the foregoing split relies on certain secret information to be shared across the NP relations, namely, the identities of relevant predicates and the local data. We can store this information in suitable commitments that are part of the NP instances for the two NP relations (doing this efficiently requires some care as we discuss below).

Problem 4: the NP relations have many checks. Even using E_{CP} only for SNARK verification and E_{BLS} for all other checks does not suffice: the NP relations $\mathcal{R}_{\mathsf{BLS}}$ and $\mathcal{R}_{\mathsf{CP}}$ still have to perform expensive checks like verifying Merkle tree authentication paths and commitment openings, and evaluating pseudorandom functions and collision resistant functions. Similar NP relations, like the one in Zerocash [Ben+14], require upwards of *four million gates* to express such checks, resulting

in high latencies for producing transactions (several minutes) and large public parameters for the system (hundreds of megabytes).

Solution 4: efficient EC primitives. Commitments and collision-resistant hashing can be expressed as very efficient arithmetic circuits if one opts for Pedersen-type constructions over suitable Edwards elliptic curves (and techniques derived from these ideas are now part of deployed systems [HBHW20]). To do this, we pick two Edwards curves, $E_{\mathsf{Ed/BLS}}$ over the field $\mathbb{F}_r$ (matching the group order of E_{BLS}) and $E_{\mathsf{Ed/CP}}$ over the field $\mathbb{F}_p$ (matching the group order of E_{CP}). This enables us to achieve very efficient circuits for primitives used in our NP relations, including commitments, collision-resistant hashing, and randomizable signatures. (Note that $E_{\mathsf{Ed/BLS}}$ and $E_{\mathsf{Ed/CP}}$ do not need to be pairing-friendly as the primitives only rely on their group structure.)

Problem 5: sharing information between NP relations is costly. We have said that splitting $\mathcal{R}_{\mathsf{e}}$ into two NP relations $\mathcal{R}_{\mathsf{BLS}}$ and $\mathcal{R}_{\mathsf{CP}}$ relies on sharing secret information via commitments across NP statements; namely, a commitment cm_{Φ} to the identities of predicates and a commitment $\mathsf{cm}_{\mathsf{ldata}}$ to the local data. But if both relations open these commitments, we cannot make an efficient use of Pedersen commitments because the two NP relations are over different fields: $\mathcal{R}_{\mathsf{BLS}}$ is over $\mathbb{F}_r$, while $\mathcal{R}_{\mathsf{CP}}$ is over $\mathbb{F}_p$. For example, if we used a Pedersen commitment over the order-r subgroup of the Edwards curve $E_{\mathsf{Ed/BLS}}$, then: (a) opening a commitment in $\mathcal{R}_{\mathsf{BLS}}$ would be cheap, but (b) opening a commitment in $\mathcal{R}_{\mathsf{CP}}$ would involve expensive simulation of $\mathbb{F}_r$-arithmetic via $\mathbb{F}_p$-arithmetic. (And similarly if we used a Pedersen commitment over the order-p subgroup of the Edwards curve $E_{\mathsf{Ed/CP}}$.) To make matters worse, the predicate identities and the local data are large, so an inefficient solution for committing to these would add significant costs to $\mathcal{R}_{\mathsf{BLS}}$ and $\mathcal{R}_{\mathsf{CP}}$.

Solution 5: hash predicate verification keys and commit to local data. In a record, instead of storing predicate verification keys, we store collision-resistant hashes of these. This reduces the cost of producing the commitment cm_{Φ} in $\mathcal{R}_{\mathsf{BLS}}$ and $\mathcal{R}_{\mathsf{CP}}$, as cm_{Φ} contains hashes that are much smaller than verification keys. We realize cm_{Φ} via Blake2s, a boolean primitive of modest cost in $\mathbb{F}_r$ and $\mathbb{F}_p$. Crucially, *only* $\mathcal{R}_{\mathsf{CP}}$ needs to access the verification keys themselves, so we can efficiently use a Pedersen hash over the Edwards curve $E_{\mathsf{Ed/CP}}$ to let $\mathcal{R}_{\mathsf{CP}}$ check the keys (supplied as non-deterministic advice) against the hashes inside cm_{Φ}.

We realize the local data commitment $\mathsf{cm}_{\mathsf{ldata}}$ via a Pedersen commitment over $E_{\mathsf{Ed/BLS}}$, and assume that predicates take $\mathsf{cm}_{\mathsf{ldata}}$ as input rather than local data in the clear. Since both $\mathcal{R}_{\mathsf{BLS}}$ and the predicate relations are defined over the field $\mathbb{F}_r$ (the prime-order subgroup of the curve E_{BLS}), non-deterministically opening $\mathsf{cm}_{\mathsf{ldata}}$ is efficient in both relations. This approach significantly reduces costs because $\mathcal{R}_{\mathsf{CP}}$ no longer needs to reason about the contents of $\mathsf{cm}_{\mathsf{ldata}}$, and can simply pass $\mathsf{cm}_{\mathsf{ldata}}$ as input to the SNARK verifier.

The NP relation $\mathcal{R}_{\mathsf{BLS}}$ has instances $\mathbb{x}_{\mathsf{BLS}}$ and witnesses $\mathbb{w}_{\mathsf{BLS}}$ of the form

$$\mathbb{x}_{\mathsf{BLS}} = \begin{pmatrix} \text{ledger digest} & \mathsf{st}_{\mathbf{L}} \\ \text{old record serial numbers} & [\mathsf{sn}_i]_1^m \\ \text{new record commitments} & [\mathsf{cm}_j]_1^n \\ \text{predicate commitment} & \mathsf{cm}_{\Phi} \\ \text{local data commitment} & \mathsf{cm}_{\mathsf{ldata}} \\ \text{transaction memorandum} & \mathsf{memo} \end{pmatrix} \quad \text{and} \quad \mathbb{w}_{\mathsf{BLS}} = \begin{pmatrix} \text{old records} & [\mathbf{r}_i]_1^m \\ \text{old record membership witnesses} & [\mathbb{w}_{\mathbf{L},i}]_1^m \\ \text{old address secret keys} & [\mathsf{ask}_i]_1^m \\ \text{new records} & [\mathbf{r}_j]_1^n \\ \text{predicate comm. randomness} & r_{\Phi} \\ \text{local data randomness} & r_{\mathsf{ldata}} \\ \text{auxiliary predicate input} & \mathsf{aux} \end{pmatrix}$$

where
- for each $i \in \{1, \ldots, m\}$, $\mathbf{r}_i = (\mathsf{apk}_i, \mathsf{payload}_i, h_{\mathsf{b},i}, h_{\mathsf{d},i}, \rho_i, r_i, \mathsf{cm}_i)$;
- for each $j \in \{1, \ldots, n\}$, $\mathbf{r}_j = (\mathsf{apk}_j, \mathsf{payload}_j, h_{\mathsf{b},j}, h_{\mathsf{d},j}, \rho_j, r_j, \mathsf{cm}_j)$.

Define the local data $\mathsf{ldata} := \begin{pmatrix} [\mathsf{cm}_i]_1^m & [\mathsf{apk}_i]_1^m & [\mathsf{payload}_i]_1^m & [h_{\mathsf{d},i}]_1^m & [h_{\mathsf{b},i}]_1^m & [\mathsf{sn}_i]_1^m & \mathsf{memo} \\ [\mathsf{cm}_j]_1^n & [\mathsf{apk}_j]_1^n & [\mathsf{payload}_j]_1^n & [h_{\mathsf{d},j}]_1^n & [h_{\mathsf{b},j}]_1^n & \mathsf{aux} & \end{pmatrix}$.

A witness $\mathbb{w}_{\mathsf{BLS}}$ is valid for an instance $\mathbb{x}_{\mathsf{BLS}}$ if the following conditions hold:
1. For each $i \in \{i, \ldots, m\}$:
   - If $\mathbf{r}_i$ is not dummy, $\mathbb{w}_{\mathbf{L},i}$ proves that the commitment cm_i is in a ledger with digest $\mathsf{st}_{\mathbf{L}}$: $\mathbf{L}.\mathsf{Verify}(\mathsf{st}_{\mathbf{L}}, \mathsf{cm}_i, \mathbb{w}_{\mathbf{L},i}) = 1$.
   - The address public key apk_i and secret key ask_i form a valid key pair: $\mathsf{apk}_i = \mathsf{CM.Commit}(\mathsf{pp}_{\mathsf{CM}}, \mathsf{sk}_{\mathsf{PRF},i};\ r_{\mathsf{pk},i})$ and $\mathsf{ask}_i = (\mathsf{sk}_{\mathsf{PRF},i}, r_{\mathsf{pk},i})$.
   - The serial number sn_i is valid: $\mathsf{sn}_i = \mathsf{PRF}_{\mathsf{sk}_{\mathsf{PRF},i}}(\rho_i)$.
   - The old record commitment cm_i is valid: $\mathsf{cm}_i = \mathsf{TCM.Commit}(\mathsf{pp}_{\mathsf{TCM}}, \mathsf{apk}_i \| \mathsf{payload}_i \| h_{\mathsf{b},i} \| h_{\mathsf{d},i} \| \rho_i;\ r_i)$.
2. For each $j \in \{1, \ldots, n\}$:
   - The serial number nonce ρ_j is computed correctly: $\rho_j = \mathsf{CRH.Eval}(\mathsf{pp}_{\mathsf{CRH}}, j \| \mathsf{sn}_1 \| \ldots \| \mathsf{sn}_m)$.
   - The new record commitment cm_j is valid: $\mathsf{cm}_j = \mathsf{TCM.Commit}(\mathsf{pp}_{\mathsf{TCM}}, \mathsf{apk}_j \| \mathsf{payload}_j \| h_{\mathsf{b},j} \| h_{\mathsf{d},j} \| \rho_j;\ r_j)$.
3. The predicate commitment cm_{Φ} is valid: $\mathsf{cm}_{\Phi} = \mathsf{b2s}([h_{\mathsf{d},i}]_1^m \parallel [h_{\mathsf{b},j}]_1^n \parallel r_{\Phi})$.
4. The local data commitment $\mathsf{cm}_{\mathsf{ldata}}$ is valid: $\mathsf{cm}_{\mathsf{ldata}} = \mathsf{CM.Commit}(\mathsf{pp}_{\mathsf{CM}}, \mathsf{ldata}; r_{\mathsf{ldata}})$

The NP relation $\mathcal{R}_{\mathsf{CP}}$ has instances $\mathbb{x}_{\mathsf{CP}}$ and witnesses $\mathbb{w}_{\mathsf{CP}}$ of the form

$$\mathbb{x}_{\mathsf{CP}} = \begin{pmatrix} \text{predicate commitment} & \mathsf{cm}_{\Phi} \\ \text{local data commitment} & \mathsf{cm}_{\mathsf{ldata}} \end{pmatrix} \quad \text{and} \quad \mathbb{w}_{\mathsf{CP}} = \begin{pmatrix} \text{old death pred. ver. keys} & [\mathsf{vk}_{\mathsf{d},i}]_1^m \\ \text{old death pred. proofs} & [\pi_{\mathsf{d},i}]_1^m \\ \text{new birth pred. ver. keys} & [\mathsf{vk}_{\mathsf{b},j}]_1^n \\ \text{new birth pred. proofs} & [\pi_{\mathsf{b},j}]_1^n \\ \text{predicate comm. randomness} & r_{\Phi} \end{pmatrix}$$

A witness $\mathbb{w}_{\mathsf{CP}}$ is valid for an instance $\mathbb{x}_{\mathsf{CP}}$ if the following conditions hold:
1. For each $i \in \{i, \ldots, m\}$:
   - The death predicate hash $h_{\mathsf{d},i}$ is computed correctly: $h_{\mathsf{d},i} = \mathsf{CRH.Eval}(\mathsf{pp}_{\mathsf{CRH}}, \mathsf{vk}_{\mathsf{d},i})$.
   - The death predicate proof $\pi_{\mathsf{d},i}$ is valid: $\mathsf{NIZK.Verify}(\mathsf{vk}_{\mathsf{d},i}, i \| \mathsf{cm}_{\mathsf{ldata}}, \pi_{\mathsf{d},i})$.
2. For each $j \in \{1, \ldots, n\}$:
   - The birth predicate hash $h_{\mathsf{b},j}$ is computed correctly: $h_{\mathsf{b},j} = \mathsf{CRH.Eval}(\mathsf{pp}_{\mathsf{CRH}}, \mathsf{vk}_{\mathsf{b},j})$.
   - The birth predicate proof $\pi_{\mathsf{b},j}$ is valid: $\mathsf{NIZK.Verify}(\mathsf{vk}_{\mathsf{b},j}, j \| \mathsf{cm}_{\mathsf{ldata}}, \pi_{\mathsf{b},j})$.
3. The predicate commitment cm_{Φ} is valid: $\mathsf{cm}_{\Phi} = \mathsf{b2s}([h_{\mathsf{d},i}]_1^m \parallel [h_{\mathsf{b},j}]_1^n \parallel r_{\Phi})$.

Figure 4.14: Splitting the NP relation $\mathcal{R}_{\mathsf{e}}$ into two NP relations $\mathcal{R}_{\mathsf{BLS}}$ and $\mathcal{R}_{\mathsf{CP}}$, over $\mathbb{F}_r$ and $\mathbb{F}_p$ respectively.

4.8 System implementation

We implemented our "plain" DPC scheme (Section 4.4) and our delegable DPC scheme (Section 4.5), by following the strategy described in Section 4.7. The resulting system, named **ZEXE** (*Zero knowledge EXEcution*), consists of several Rust libraries: (a) a library for finite field and elliptic curve arithmetic, adapted from [Bow17b]; (b) a library for cryptographic building blocks, including zkSNARKs for constraint systems (using components from [Bow17a]); (c) a library with constraints for many of these building blocks; and (d) a library that realizes our constructions of plain and delegable DPC. Our code base, like our construction, is written in terms of abstract building blocks, which allows to easily switch between different instantiations of the building blocks. In the rest of this section we describe the efficient instantiations used in the experiments reported in Section 4.9.

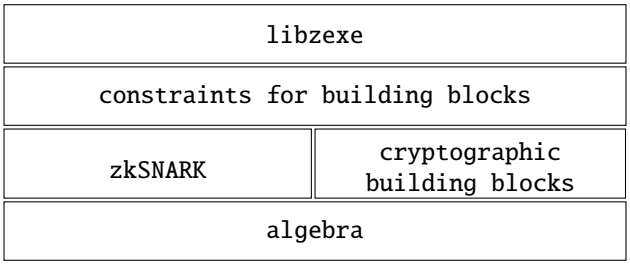

Figure 4.15: Stack of libraries comprising ZEXE.

Ledger. The ledger $\mathbf{L}$ in our prototype is simply an ideal ledger, i.e., an append-only log of valid transactions that is stored in memory. Of course, in a real-world deployment, this ideal ledger would be replaced by a distributed protocol that realizes (a suitable approximation of) an ideal ledger. Recall from Section 4.3.1 that we require the ledger $\mathbf{L}$ to provide a method to efficiently prove and verify membership of a transaction, or one of its subcomponents, in $\mathbf{L}$. For this, we maintain a Merkle tree [Mer87] atop the list of transactions, using the collision-resistant hash function CRH described below. This results in the following algorithms for $\mathbf{L}$.

- $\mathbf{L}.\mathsf{Push}(\mathsf{tx})$: Append tx to the transaction list and update the Merkle tree.
- $\mathbf{L}.\mathsf{Digest} \rightarrow \mathsf{st}_{\mathbf{L}}$: Return the root of the Merkle tree.
- $\mathbf{L}.\mathsf{Prove}(\mathsf{tx}) \rightarrow \mathbb{w}_{\mathbf{L}}$: Return the authentication path for tx in the Merkle tree.
- $\mathbf{L}.\mathsf{Verify}(\mathsf{st}_{\mathbf{L}}, \mathsf{tx}, \mathbb{w}_{\mathbf{L}}) \rightarrow b$: Check that $\mathbb{w}_{\mathbf{L}}$ is a valid authentication path for tx in a tree with root $\mathsf{st}_{\mathbf{L}}$.

Our prototype maintains the Merkle tree in memory, but a real-world deployment would have to maintain it via a distributed protocol. (Such data structures atop distributed ledgers are used in existing systems [Zcash].)

Pseudorandom function. Fixing key length and input length at 256 bits, we instantiate PRF using the Blake2s hash function [ANWW13]: $\mathsf{PRF}_k(x) := \mathsf{b2s}(k \| x)$ for $k, x \in \{0,1\}^{256}$.

Elliptic curves. Our implementation strategy (see Section 4.7) involves several elliptic curves: two pairing-friendly curves E_{BLS} and E_{CP}, and two "plain" curves $E_{\mathsf{Ed/BLS}}$ and $E_{\mathsf{Ed/CP}}$ whose base field respectively matches the prime-order subgroup of E_{BLS} and E_{CP}. Details about these curves are in

Figure 4.16; the parameter used to generate the BLS curve E_{BLS} is $x = 3 \cdot 2^{46} \cdot (7 \cdot 13 \cdot 499) + 1$ (see Section 4.7 for why).

name	curve type	embedding degree	size of prime-order subgroup	size of base field	size of compressed group elements in bytes	
					$\mathbb{G}_1$	$\mathbb{G}_2$
$E_{\mathsf{Ed/BLS}}$	twisted Edwards	—	s	r	32	—
E_{BLS}	BLS	12	r	p	48	96
$E_{\mathsf{Ed/CP}}$	twisted Edwards	—	t	p	48	—
E_{CP}	short Weierstrass	6	p	q	104	312

prime	value	size in bits	2-adicity
s	`0x4aad957a68b2955982d1347970dec005293a3afc43c8afeb95aee9ac33fd9ff`	251	1
r	`0x12ab655e9a2ca55660b44d1e5c37b00159aa76fed00000010a11800000000001`	253	47
t	`0x35c748c2f8a21d58c760b80d94292763445b3e601ea271e1d75fe7d6eeb84234066d10f5d893814103486497d95295`	374	2
p	`0x1ae3a4617c510eac63b05c06ca1493b1a22d9f300f5138f1ef3622fba094800170b5d44300000008508c00000000001`	377	46
q	`0x3848c4d2263babf8941fe959283d8f526663bc5d176b746af0266a7223ee72023d07830c728d80f9d78bab3596c8617c579252a3fb77c79c13201ad533049cfe6a399c2f764a12c4024bee135c065f4d26b7545d85c16dfd424adace79b57b942ae9`	782	3

Figure 4.16: The elliptic curves $E_{\mathsf{BLS}}, E_{\mathsf{CP}}, E_{\mathsf{Ed/BLS}}, E_{\mathsf{Ed/CP}}$.

NIZKs. We instantiate the NIZKs used for the NP relation $\mathcal{R}_{\mathsf{e}}$ via zero-knowledge *succinct* non-interactive arguments of knowledge (zkSNARKs), which makes our DPC schemes succinct (see Remark 4.4.1). Concretely, we rely on the simulation-extractable zkSNARK of Groth and Maller [GM17], used over the pairing-friendly elliptic curves E_{BLS} (for proving predicates' satisfiability) and E_{CP} (for proving validity of these latter proofs).

DLP-hard group. Several instantiations of cryptographic primitives introduced below rely on the hardness of extracting discrete logarithms in a prime order group. We generate these groups via a group generator $\mathsf{SampleGrp}$, which on input a security parameter λ (represented in unary), outputs a tuple $(\mathbb{G}, q, g)$ that describes a group $\mathbb{G}$ of prime order q generated by g. The discrete-log problem is hard in $\mathbb{G}$. In our prototype we fix $\mathbb{G}$ to be the largest prime-order subgroup of either $E_{\mathsf{Ed/BLS}}$ or $E_{\mathsf{Ed/CP}}$, depending on the context.

Commitments. We instantiate (plain and) trapdoor commitments via Pedersen commitments over $\mathbb{G}$, as defined in Figure 4.17; note that the setup algorithm takes as additional input the message length n. Pedersen commitments are perfectly hiding, and are computationally binding if the discrete-log problem is hard in $\mathbb{G}$.

Collision-resistant hashing. We instantiate CRH via a Pedersen hash function over $\mathbb{G}$, as specified in Figure 4.18; note that the setup algorithm takes as additional input the message length n. Collision resistance follows from hardness of the discrete-logarithm problem [MRK03].

Remark 4.8.1. Hopwood et al. [HBHW20] note that projecting a twisted Edwards curve point (x, y) to its x-coordinate is injective when the point is in the curve's largest prime-order subgroup. Our implementation uses this fact to reduce the output size of TCM and CRH by projecting their output to its x-coordinate.

$\mathsf{TCM.Setup}(1^\lambda, n) \to \mathsf{pp}_{\mathsf{TCM}}$:
1. Sample a group: $(\mathbb{G}, q, g) \leftarrow \mathsf{SampleGrp}(1^\lambda)$.
2. For $i \in \{1, \ldots, n\}$, sample generator h_i: $r_i \leftarrow \mathbb{Z}_q; h_i := g^{r_i}$.
3. Output $\mathsf{pp}_{\mathsf{TCM}} := (\mathbb{G}, q, g, [h_i]_1^n)$.
$\mathsf{TCM.Commit}(\mathsf{pp}_{\mathsf{TCM}}, m \in \{0,1\}^n; r_{\mathsf{cm}}) \to \mathsf{cm}$:
1. Parse $\mathsf{pp}_{\mathsf{TCM}}$ as $(\mathbb{G}, q, g, [h_i]_1^n)$.
2. Output $\mathsf{cm} := g^{r_{\mathsf{cm}}} \prod_{i=1}^n h_i^{m_i}$.

Figure 4.17: Pedersen commitment scheme.

$\mathsf{CRH.Setup}(1^\lambda, n) \to \mathsf{pp}_{\mathsf{CRH}}$:
1. Sample a group: $(\mathbb{G}, q, g_1) \leftarrow \mathsf{SampleGrp}(1^\lambda)$.
2. For $i \in \{2, \ldots, n\}$, sample generator g_i: $r_i \leftarrow \mathbb{Z}_q; g_i := g^{r_i}$.
3. Output $\mathsf{pp}_{\mathsf{CRH}} := (\mathbb{G}, q, [g_i]_1^n)$.
$\mathsf{CRH.Eval}(\mathsf{pp}_{\mathsf{CRH}}, m \in \{0,1\}^n) \to h$:
1. Parse $\mathsf{pp}_{\mathsf{CRH}}$ as $(\mathbb{G}, q, [g_i]_1^n)$.
2. Output $h := \prod_{i=1}^n g_i^{m_i}$.

Figure 4.18: Pedersen collision-resistant hash.

4.9 System evaluation

In Section 4.9.1 we evaluate individual cryptographic building blocks. In Section 4.9.2 we evaluate the cost of NP relations expressed as constraints, as required by the underlying zkSNARK. In Section 4.9.3 we evaluate the running time of DPC algorithms. In Section 4.9.4 we evaluate the sizes of DPC data structures. All reported measurements were taken on a machine with an Intel Xeon 6136 CPU at $3.0\,\mathrm{GHz}$ with $252\,\mathrm{GB}$ of RAM.

4.9.1 Cryptographic building blocks

We are interested in two types of costs associated with a given cryptographic building block: the *native execution cost*, which are the running times of certain algorithms on a CPU; and the *constraint cost*, which are the numbers of constraints required to express certain invariants, to be used by the underlying zkSNARK.

Native execution cost. The zkSNARK dominates native execution cost, and the costs of all other building blocks are negligible in comparison. Therefore we separately report only the running times of the zkSNARK, which in our case is a protocol due to Groth and Maller [GM17], abbreviated as GM17. When instantiated over the elliptic curve E_{BLS}, the GM17 prover takes $25\,\mu\mathrm{s}$ per constraint (with 12 threads), while the GM17 verifier takes $250n\,\mu\mathrm{s} + 9.5\,\mathrm{ms}$ on an input with n field elements (with 1 thread). When instantiated over the elliptic curve E_{CP}, the respective prover and verifier costs are $147\,\mu\mathrm{s}$ per constraint and $1.6n\,\mathrm{ms} + 34\,\mathrm{ms}$.

Constraint cost. There are three building blocks that together account for the majority of the cost of NP statements that we use. These are: (a) the Blake2s PRF, which requires 21792 constraints to map a 64-byte input to a 32-byte output; (b) the Pedersen collision-resistant hash, which requires $5n$ constraints for an input of n bits; and (c) the GM17 verifier, which requires $14n + 52626$ constraints for an n-bit input.

4.9.2 The execute NP relation

In many zkSNARK constructions, including the one that we use, one must express all the relevant checks in the given NP relation as (rank-1) *quadratic constraints* over a certain large prime field. The goal is to minimize the number of such constraints because the prover's costs grow (quasi)linearly in this number.

In our DPC scheme we use a zkSNARK for the NP relation $\mathcal{R}_{\mathsf{e}}$ in Fig. 4.9 and, similarly, in our delegable DPC scheme we use it for the NP relation $\mathcal{R}_{\mathsf{e}}^{\mathsf{del}}$ in Fig. 4.23. More precisely, for efficiency reasons explained in Section 4.7, we split $\mathcal{R}_{\mathsf{e}}$ into the two NP relations $\mathcal{R}_{\mathsf{BLS}}$ and $\mathcal{R}_{\mathsf{CP}}$ in Fig. 4.14, which we prove via zkSNARKs over the pairing-friendly curves E_{BLS} and E_{CP}, respectively. (We also similarly split $\mathcal{R}_{\mathsf{e}}^{\mathsf{del}}$.)

Table 4.3 reports the number of constraints that we use to express $\mathcal{R}_{\mathsf{BLS}}$, as a function of the number of input (m) and output (n) records, and additionally reports its primary contributors. Table 4.4 does the same for $\mathcal{R}_{\mathsf{CP}}$. These tables show that for each input record costs are dominated by verification of a Merkle tree path and the verification of a (death predicate) proof; while for each

output record costs are dominated by the verification of a (birth predicate) proof. We also report the cumulative number of constraints when setting $m := 2$ and $n := 2$ because this is a representative instantiation of m and n that enables useful applications.

4.9.3 DPC algorithms

In Table 4.1 we report the running times of algorithms in our plain DPC and delegable DPC implementations for two input and two output records (i.e., $m := 2$ and $n := 2$). Note that for Execute and Verify, we have excluded costs of ledger operations (such as retrieving an authentication path or scanning for duplicate serial numbers) because these depend on how a ledger is realized, which is orthogonal to our work. Also, we assume that Execute receives as inputs the application-specific SNARK proofs checked by the NP relation. Producing each of these proofs requires invoking the GM17 prover, over the elliptic curve E_{BLS}, for the relevant birth or death predicate; we describe the cost of doing so for representative applications in Section 4.9.5.

Observe that the overhead incurred by delegable DPC over plain DPC is negligible, and that, as expected, Setup and Execute are the most costly algorithms, as they invoke costly zkSNARK setup and proving algorithms. To mitigate these costs, Setup and Execute are executed on 12 threads; everything else is executed with 1 thread. Overall, we learn that Execute takes less than a minute, and Verify takes tens of milliseconds. Furthermore, both Setup and Execute consume less than $5\,\mathrm{GB}$ of RAM. These costs are comparable with those of similar systems such as Zerocash [Ben+14] and Hawk [KMSWP16].

4.9.4 DPC data structures

Addresses. An address public key in a DPC scheme is a point on the elliptic curve $E_{\mathsf{Ed/BLS}}$, which is 32 bytes when compressed (see Fig. 4.16); the corresponding secret key is 64 bytes and consists of a PRF seed (32 bytes) and commitment randomness (32 bytes). In a delegable DPC scheme, address public keys do not change, but address secret keys are 96 bytes, because they additionally contain the 32-byte secret key of a randomizable signature scheme over the elliptic curve $E_{\mathsf{Ed/BLS}}$ (see Fig. 4.10).

Transactions. A transaction in a DPC scheme, with two input and two output records, is 968 bytes. It contains two zkSNARK proofs: π_{BLS}, over the elliptic curve E_{BLS}, and π_{CP}, over the curve E_{CP}. Each proof consists of two $\mathbb{G}_1$ and one $\mathbb{G}_2$ elements from its respective curve, amounting to 192 bytes for π_{BLS} and 520 for π_{CP} (both in compressed form). In general, for m input records and n output records, transactions are $32m + 32n + 840$ bytes. In a delegable DPC scheme, a transaction additionally contains a 64-byte signature for each input record. See Table 4.2 for a detailed break down of all of these costs.

Record contents. We set a record's payload to be 32 bytes long; if a predicate needs longer data then it can set the payload to be the hash of this data, and use non-determinism to access the data. The foregoing choice means that all contents of a record add up to 224 bytes, since a record consists

of an address public key (32 bytes), the 32-byte payload, hashes of birth and death predicates (48 bytes each), a serial number nonce (32 bytes), and commitment randomness (32 bytes).

	Plain DPC	Delegable DPC
Setup	109.62 s	109.3 s
GenAddress	380 µs	780 µs
Execute	52.5 s	53.4 s
Verify	46 ms	47 ms

Table 4.1: Cost of DPC algorithms for 2 inputs and 2 outputs.

	Plain DPC	Delegable DPC
2 inputs and 2 outputs	968	1096
m inputs and n outputs	$32m + 32n + 840$	$96m + 32n + 840$
Per input record:		
Serial number	32	32
Signature	—	64
Per output record:		
Commitment	32	32
Memorandum	32	32
zkSNARK proof over E_{CP}	520	520
zkSNARK proof over E_{BLS}	192	192
Predicate commitment	32	32
Local data commitment	32	32
Ledger digest	32	32

Table 4.2: Size of a DPC transaction (in bytes).

4.9.5 Applications

We do not report total costs for producing transactions for the applications in Section 4.6 because the additional application-specific costs are *negligible* compared to the base cost reported in Table 4.1. This is because all application-specific proofs are produced over the efficient elliptic curve E_{BLS}, and moreover, for each application we consider, the heaviest computation checked by these proofs is the relatively lightweight one of opening the local data commitment; the remaining costs consist of a few cheap range and equality checks. Indeed, with two input and two output records, these applications require fewer than $35,000$ constraints (compared to over $350,000$ for $\mathcal{R}_{\mathsf{BLS}}$ and $\mathcal{R}_{\mathsf{CP}}$), and producing the corresponding proofs takes tens of milliseconds (compared to tens of seconds for the base cost of DPC.Execute).

		Plain DPC	Delegable DPC
Total with 2 inputs and 2 outputs		**387412**	**414339**
Below we provide a breakdown of the number of constraints with m input and n output records.			
Per input record	Total	117699	125401
	Enforce validity of:		
	Merkle tree path	81824	81824
	Address key pair	3822	8435
	Serial number computation	22301	25390
	Record commitment	9752	9752
Per output record	Total	15427	19523
	Enforce validity of:		
	Serial number nonce	5417	9513
	Record commitment	10010	10010
Other:	Enforce validity of:		
	Predicate commitment	$21792 \cdot \lceil \frac{3}{4}(m+n) + \frac{1}{2} \rceil$	$21792 \cdot \lceil \frac{3}{4}(m+n) + \frac{1}{2} \rceil$
	Local data commitment	$7168 \cdot m + 6144 \cdot n$	$8192 \cdot m + 6144 \cdot n$
	Miscellaneous	7368	8651

Table 4.3: Number of constraints for $\mathcal{R}_{\mathsf{BLS}}$.

		Plain DPC	Delegable DPC
Total with 2 inputs and 2 outputs		**439224**	**439476**
Below we provide a breakdown of the number of constraints with m input and n output records.			
Per input record	Total	87569	87569
	Enforce validity of:		
	Death predicate ver. key	45827	45827
	Death predicate proof	41742	41742
Per output record	Total	87569	87569
	Enforce validity of:		
	Birth predicate ver. key	45827	45827
	Birth predicate proof	41742	41742
Other	Enforce validity of:		
	Predicate commitment	$21792 \cdot \lceil \frac{3}{4}(m+n) + \frac{1}{2} \rceil$	$21792 \cdot \lceil \frac{3}{4}(m+n) + \frac{1}{2} \rceil$
	Miscellaneous	1780	2032

Table 4.4: Number of constraints for $\mathcal{R}_{\mathsf{CP}}$.

4.10 Proof of security for our DPC scheme

We prove that our DPC construction (see Section 4.4) satisfies the security definition in Section 4.3.3. To do this, for every real-world (efficient) adversary $\mathcal{A}$, we construct an ideal-world (efficient) simulator $\mathcal{S}$ such that the ideal-world and real-world executions are computationally indistinguishable with respect to any (efficient) environment $\mathcal{E}$. We proceed in three parts: in Section 4.10.1 we describe building blocks used to construct the simulator $\mathcal{S}$; in Section 4.10.2 we describe the simulator $\mathcal{S}$; in Section 4.10.3 we argue that the ideal-world and the real-world executions are computationally indistinguishable.

4.10.1 Building blocks for the simulator

We describe various algorithms that are used as sub-routines in the simulator $\mathcal{S}$.

Trapdoor commitments. Recall from Section 4.4.1 that a *trapdoor* commitment scheme is a commitment scheme with auxiliary algorithms $(\mathsf{SimSetup}, \mathsf{Equivocate})$ that enable one to open a commitment cm to any chosen message. Below we restrict cm to be a commitment to the empty string ε because this is sufficient for the proof of security of our DPC scheme.

- *Trapdoor setup:* on input a security parameter, $\mathsf{TCM.SimSetup}$ samples public parameters $\mathsf{pp}_{\mathsf{TCM}}$ and a trapdoor $\mathsf{td}_{\mathsf{TCM}}$ such that $\mathsf{pp}_{\mathsf{TCM}}$ is indistinguishable from public parameters sampled by $\mathsf{TCM.Setup}$.
- *Equivocation:* on input public parameters $\mathsf{pp}_{\mathsf{TCM}}$, trapdoor $\mathsf{td}_{\mathsf{TCM}}$, commitment cm to ε, commitment randomness r_{cm} (so that $\mathsf{TCM.Commit}(\mathsf{pp}_{\mathsf{TCM}}, \varepsilon; r_{\mathsf{cm}}) = \mathsf{cm}$), and target message m', $\mathsf{TCM.Equivocate}$ outputs commitment randomness r'_{cm} such that $\mathsf{TCM.Commit}(\mathsf{pp}_{\mathsf{TCM}}, m'; r'_{\mathsf{cm}}) = \mathsf{cm}$. Moreover, if r_{cm} is uniformly random then r'_{cm} is statistically close to uniformly random.

In Figure 4.19 we instantiate these algorithms for the Pedersen commitment scheme. Note that the real and simulated public parameters are identical; moreover, the trapdoor randomness r'_{cm} is the real randomness r_{cm} shifted by uniformly random field elements, and is hence statistically close to r_{cm}.

$\mathsf{TCM.SimSetup}(1^\lambda, n) \to (\mathsf{pp}_{\mathsf{TCM}}, \mathsf{td}_{\mathsf{TCM}})$
1. Sample a group: $(\mathbb{G}, q, g) \leftarrow \mathsf{SampleGrp}(1^\lambda)$.
2. For $i \in \{1, \ldots, n\}$:
   sample r_i uniformly from $\mathbb{Z}_q$, and set $h_i := g^{r_i}$.
3. Output $(\mathsf{pp}_{\mathsf{TCM}} := (\mathbb{G}, q, g, [h_i]_1^n), \mathsf{td}_{\mathsf{TCM}} := [r_i]_1^n)$.

$\mathsf{TCM.Equivocate}(\mathsf{pp}_{\mathsf{TCM}}, \mathsf{td}_{\mathsf{TCM}}, \mathsf{cm}, r_{\mathsf{cm}}, m' \in \{0,1\}^n) \to r'_{\mathsf{cm}}$
1. Parse $\mathsf{pp}_{\mathsf{TCM}}$ as $(\mathbb{G}, q, g, [h_i]_1^n)$.
2. Parse $\mathsf{td}_{\mathsf{TCM}}$ as $[r_i]_1^n$.
3. Output $r'_{\mathsf{cm}} := r_{\mathsf{cm}} - \sum_{i=1}^n r_i m'_i \bmod q$.

Figure 4.19: Simulated setup and equivocation algorithms for the Pedersen commitment scheme.

NIZKs. The scheme $\mathsf{NIZK} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ is a *simulation-extractable* non-interactive zero knowledge argument. Formally stating the properties of this scheme involves several auxiliary algorithms.

– *Trapdoor setup:* on input a security parameter and a description of an NP relation $\mathcal{R}$, $\mathsf{NIZK.SimSetup}$ outputs a set of public parameters $\mathsf{pp}_{\mathsf{NIZK}}$ and a trapdoor $\mathsf{td}_{\mathsf{NIZK}}$.

– *Simulation:* on input public parameters $\mathsf{pp}_{\mathsf{NIZK}}$, trapdoor $\mathsf{td}_{\mathsf{NIZK}}$, NP instance $\mathbb{x}$, and (optionally) auxiliary information aux, NIZK.Simulate outputs a simulated proof π.
– *Extraction:* on input public parameters $\mathsf{pp}_{\mathsf{NIZK}}$, trapdoor $\mathsf{td}_{\mathsf{NIZK}}$, NP instance $\mathbb{x}$, and proof π, NIZK.Extract outputs a witness $\mathbb{w}$ such that $(\mathbb{x}, \mathbb{w}) \in \mathcal{R}$ (allegedly).

We can now state the properties satisfied by NIZK.

- *Completeness:* for every NP relation $\mathcal{R}$ and instance-witness pair $(\mathbb{x}, \mathbb{w}) \in \mathcal{R}$,

$$\Pr\left[\mathsf{NIZK.Verify}(\mathsf{pp}_{\mathsf{NIZK}}, \mathbb{x}, \pi) = 1 \;\middle|\; \begin{array}{r} \mathsf{pp}_{\mathsf{NIZK}} \leftarrow \mathsf{NIZK.Setup}(1^\lambda, \mathcal{R}) \\ (\mathbb{x}, \pi) \leftarrow \mathsf{NIZK.Prove}(\mathsf{pp}_{\mathsf{NIZK}}, \mathbb{x}, \mathbb{w}) \end{array} \right] = 1 \ .$$

- *Perfect zero knowledge:* for every relation $\mathcal{R}$ and efficient adversary $\mathcal{A}$,

$$\Pr\left[\begin{array}{l} \mathsf{pp}_{\mathsf{NIZK}} \leftarrow \mathsf{NIZK.Setup}(1^\lambda, \mathcal{R}) \\ \mathcal{A}^{S_1(\cdot,\cdot)}(\mathsf{pp}_{\mathsf{NIZK}}, \mathsf{aux}) = 1 \end{array} \right] = \Pr\left[\begin{array}{l} (\mathsf{pp}_{\mathsf{NIZK}}, \mathsf{td}_{\mathsf{NIZK}}) \leftarrow \mathsf{NIZK.SimSetup}(1^\lambda, \mathcal{R}) \\ \mathcal{A}^{S_2(\cdot,\cdot)}(\mathsf{pp}_{\mathsf{NIZK}}, \mathsf{aux}) = 1 \end{array} \right]$$

  where the two oracles are defined as follows
  – $S_1(\mathbb{x}, \mathbb{w}) :=$ "if $(\mathbb{x}, \mathbb{w}) \in \mathcal{R}$ then $\mathsf{NIZK.Prove}(\mathsf{pp}_{\mathsf{NIZK}}, \mathbb{x}, \mathbb{w})$, else abort";
  – $S_2(\mathbb{x}, \mathbb{w}) :=$ "if $(\mathbb{x}, \mathbb{w}) \in \mathcal{R}$ then $\mathsf{NIZK.Simulate}(\mathsf{pp}_{\mathsf{NIZK}}, \mathsf{td}_{\mathsf{NIZK}}, \mathbb{x})$, else abort".
- *Simulation extractability:* for every relation $\mathcal{R}$ and efficient adversary $\mathcal{A}$,

$$\Pr\left[\begin{array}{l} (\mathbb{x}, \pi) \notin Q \\ (\mathbb{x}, \mathbb{w}) \notin \mathcal{R} \\ \mathsf{NIZK.Verify}(\mathsf{pp}_{\mathsf{NIZK}}, \mathbb{x}, \pi) = 1 \end{array} \;\middle|\; \begin{array}{r} (\mathsf{pp}_{\mathsf{NIZK}}, \mathsf{td}_{\mathsf{NIZK}}) \leftarrow \mathsf{NIZK.SimSetup}(1^\lambda, \mathcal{R}) \\ (\mathbb{x}, \pi) \leftarrow \mathcal{A}^{S(\cdot)}(\mathsf{pp}_{\mathsf{NIZK}}) \\ \mathbb{w} \leftarrow \mathsf{NIZK.Extract}(\mathsf{pp}_{\mathsf{NIZK}}, \mathsf{td}_{\mathsf{NIZK}}, \mathbb{x}, \pi) \end{array} \right] = \mathrm{negl}(\lambda) \ ,$$

  where $S(\mathbb{x}) := \mathsf{NIZK.Simulate}(\mathsf{pp}_{\mathsf{NIZK}}, \mathsf{td}_{\mathsf{NIZK}}, \mathbb{x})$ and Q is the set of query-answer pairs between the adversary $\mathcal{A}$ and the simulated-proof oracle S.

4.10.2 The ideal-world simulator

The ideal-word simulator $\mathcal{S}$ will interact with the ideal functionality $\mathcal{F}_{\text{DPC}}$ and with the environment $\mathcal{E}$. Note that for UC security it suffices to show security against a dummy real-world adversary $\mathcal{A}$ that simply forwards all instructions from the environment $\mathcal{E}$ [Can01]. Since our security definition is a special case of UC security, we inherit this simplification, and thus only consider such an adversary $\mathcal{A}$. The pseudocode for $\mathcal{S}$ is provided below; auxiliary subroutines are provided in Figure 4.20.

Setup.

1. Initialize an empty table $\mathcal{S}$.Records that maps record commitments to their contents.
2. Initialize an empty table $\mathcal{S}$.AddrPk that maps address public keys to their secret keys.
3. Initialize an empty transaction ledger $\mathbf{L}$.
4. Sample simulated public parameters and trapdoor: $(\mathsf{pp}, \mathsf{td}) \leftarrow \mathsf{DPC.SimSetup}(1^\lambda)$. (See Fig. 4.20.)

5. Define

$$\begin{aligned}\mathsf{SampleAddrPk}(\cdot) &:= \mathsf{CM.Commit}(\mathsf{pp}_{\mathsf{CM}}, \varepsilon; \cdot) \ , \\ \mathsf{SampleCm}(\cdot) &:= \mathsf{TCM.Commit}(\mathsf{pp}_{\mathsf{TCM}}, \varepsilon; \cdot) \ , \\ \mathsf{SampleSn}(\cdot) &:= \text{“sample uniformly random string of correct length”} \ .\end{aligned}$$

6. Start ideal-world execution with the above $(\mathsf{SampleAddrPk}, \mathsf{SampleCm}, \mathsf{SampleSn})$.

At this point, the simulator will receive messages notifying it of transactions and of messages sharing contents of newly-created records. The simulator handles each case separately.

Transaction notifications.

- **From environment.** When $\mathcal{E}$ instructs a corrupted party to invoke $\mathbf{L}.\mathsf{Push}(\mathsf{tx})$:
  1. If $\mathsf{DPC.Verify}^{\mathbf{L}}(\mathsf{pp}, \mathsf{tx}) \neq 1$, abort.
  2. Parse the real-world transaction tx as $([\mathsf{sn}_i]_1^m, [\mathsf{cm}_j]_1^n, \mathsf{memo}, \star)$.
  3. Compute $([\mathbf{r}_i]_1^m, [\mathsf{ask}_i]_1^m, [\mathbf{r}_j]_1^n, \mathsf{aux}) \leftarrow \mathsf{DPC.ExtractExecute}(\mathsf{pp}, \mathsf{td}, \mathsf{tx})$. [See Figure 4.20.]
  4. For every $i \in \{1, \ldots, m\}$:
     a) Parse the real-world record $\mathbf{r}_i$ as $(\mathsf{apk}_i, \mathsf{payload}_i, \Phi_{\mathsf{b},i}, \Phi_{\mathsf{d},i}, \rho_i, r_i, \mathsf{cm}_i)$.
     b) Parse the address secret key ask_i as $(\mathsf{sk}_{\mathsf{PRF},i}, r_{\mathsf{pk},i})$.
     c) If $\mathcal{S}.\mathsf{Records}[\mathsf{cm}_i] \neq \mathbf{r}_i$, abort. (***Note:*** *Captures binding property of the commitment.*)
     d) If $\mathbf{L}.\mathsf{Contains}(\mathsf{cm}_i) = 0$, abort. (***Note:*** *Captures existence of record.*)
     e) Create the ideal-world record $\mathbb{r}_i := (\mathsf{cm}_i, \mathsf{apk}_i, \mathsf{payload}_i, \Phi_{\mathsf{b},i}, \Phi_{\mathsf{d},i})$.
     f) If $\mathcal{S}.\mathsf{AddrPk}[\mathsf{apk}_i] = \perp$:
        i. Invoke $\mathcal{F}_{\text{DPC}}.\mathsf{GenAddress}(\mathsf{apk}_i)$.
        ii. Insert apk_i into $\mathcal{S}.\mathsf{AddrPk}$: $\mathcal{S}.\mathsf{AddrPk}[\mathsf{apk}_i] := \mathsf{ask}_i$.
     g) Else, if $\mathcal{S}.\mathsf{AddrPk}[\mathsf{apk}_i] \neq \mathsf{ask}_i$, abort. (***Note:*** *Captures uniqueness of secret key.*)
  5. For every $j \in \{1, \ldots, n\}$:
     a) Parse the real-world record $\mathbf{r}_j$ as $(\mathsf{apk}_j, \mathsf{payload}_j, \Phi_{\mathsf{b},j}, \Phi_{\mathsf{d},j}, \rho_j, r_j, \mathsf{cm}_j)$.
     b) If the serial number nonce ρ_j was seen in a prior extracted transaction, or if $\rho_j = \rho_k$ for $k \neq j$, abort. (***Note:*** *Captures uniqueness of nonce.*)
     c) Set $\mathcal{S}.\mathsf{Records}[\mathsf{cm}_j] := \mathbf{r}_j$.
  6. Construct instance for $\mathcal{R}_{\mathsf{e}}$: $\mathbb{x}_{\mathsf{e}} := (\mathsf{st}_{\mathbf{L}}, [\mathsf{sn}_i]_1^m, [\mathsf{cm}_j]_1^n, \mathsf{memo})$.
  7. Construct witness for $\mathcal{R}_{\mathsf{e}}$: $\mathbb{w}_{\mathsf{e}} := ([\mathbf{r}_i]_1^m, [\mathbb{w}_{\mathbf{L},i}]_1^m, [\mathsf{ask}_i]_1^m, [\mathbf{r}_j]_1^n, \mathsf{aux})$.
  8. If $(\mathbb{x}_{\mathsf{e}}, \mathbb{w}_{\mathsf{e}}) \notin \mathcal{R}_{\mathsf{e}}$, abort.
  9. Invoke $\mathcal{F}_{\text{DPC}}.\mathsf{Execute}\begin{pmatrix} [\mathbb{r}_i]_1^m & [\mathsf{meta}_i]_1^m & [\mathsf{sn}_i]_1^m & \mathsf{aux} & \mathsf{memo} \\ [\mathsf{cm}_j]_1^n & [\mathsf{apk}_j]_1^n & [\mathsf{payload}_j]_1^n & [\Phi_{\mathsf{b},j}]_1^n & [\Phi_{\mathsf{d},j}]_1^n \end{pmatrix}$.
  10. **Receive from $\mathcal{F}_{\text{DPC}}$:** $[\mathbb{r}_j]_1^n$.
  11. **Receive from $\mathcal{F}_{\text{DPC}}$:** $(\texttt{Execute}, [\mathsf{sn}_i]_1^m, [\mathsf{cm}_j]_1^n, \mathsf{memo})$.
  12. Append the real-world transaction tx to the ledger $\mathbf{L}$.
- **From ideal functionality.** When $\mathcal{F}_{\text{DPC}}$ broadcasts $(\texttt{Execute}, [\mathsf{sn}_i]_1^m, [\mathsf{cm}_j]_1^n, \mathsf{memo})$:
  1. Compute $([\mathbf{r}_j]_1^n, \mathsf{tx}) \leftarrow \mathsf{DPC.SimExecute}^{\mathbf{L}}(\mathsf{pp}, \mathsf{td}, [\mathsf{sn}_i]_1^m, [\mathsf{cm}_j]_1^n, \mathsf{memo})$. (See Fig. 4.20.)

2. For each $j \in \{1, \ldots, n\}$, set $\mathcal{S}.\mathsf{Records}[\mathsf{cm}_j] := \mathbf{r}_j$.
3. Append the real-world transaction tx to the ledger $\mathbf{L}$.

Record authorization notification.

- **From environment.** When $\mathcal{E}$ instructs a corrupted party to send $(\texttt{RecordAuth}, \mathbf{r}, \mathcal{P})$ to $\mathcal{P}$:
  1. Parse the real-world record $\mathbf{r}$ as $(\mathsf{apk}, \mathsf{payload}, \Phi_{\mathsf{b}}, \Phi_{\mathsf{d}}, \rho, r, \mathsf{cm})$.
  2. Invoke $\mathcal{F}_{\text{DPC}}.\mathsf{ShareRecord}(\mathbb{r}, \mathcal{P})$ with $\mathbb{r} := (\mathsf{cm}, \mathsf{apk}, \mathsf{payload}, \Phi_{\mathsf{b}}, \Phi_{\mathsf{d}})$.
- **From ideal functionality.** When $\mathcal{F}_{\text{DPC}}$ sends $(\texttt{RecordAuth}, \mathbb{r}, r)$:
  1. Parse the ideal record $\mathbb{r}$ as $(\mathsf{cm}, \mathsf{apk}, \mathsf{payload}, \Phi_{\mathsf{b}}, \Phi_{\mathsf{d}})$.
  2. Retrieve the real-world record $\mathbf{r} = \mathcal{S}.\mathsf{Records}[\mathsf{cm}]$, and set the serial number nonce $\rho := \mathbf{r}.\rho$.
  3. Define new record commitment message $m := (\mathsf{apk} \| \mathsf{payload} \| \Phi_{\mathsf{b}} \| \Phi_{\mathsf{d}} \| \rho)$.
  4. Compute new commitment randomness $r' \leftarrow \mathsf{TCM.Equivocate}(\mathsf{pp}_{\mathsf{TCM}}, \mathsf{td}_{\mathsf{TCM}}, \mathsf{cm}, r, m)$.
  5. Construct the new real-world record $\mathbf{r}' := (\mathsf{apk}, \mathsf{payload}, \Phi_{\mathsf{b}}, \Phi_{\mathsf{d}}, \rho, r', \mathsf{cm})$.
  6. Set $\mathcal{S}.\mathsf{Records}[\mathsf{cm}] := \mathbf{r}'$.
  7. **Send to $\mathcal{A}$:** $(\texttt{RecordAuth}, \mathbf{r}')$.

DPC.SimSetup
Input: security parameter 1^λ
Output: simulated public parameters pp and trapdoor td

1. Sample parameters for commitment: $\mathsf{pp}_{\mathsf{CM}} \leftarrow \mathsf{CM.Setup}(1^\lambda)$.
2. Sample simulated parameters for trapdoor commitment: $(\mathsf{pp}_{\mathsf{TCM}}, \mathsf{td}_{\mathsf{TCM}}) \leftarrow \mathsf{TCM.SimSetup}(1^\lambda)$.
3. Sample parameters for CRH: $\mathsf{pp}_{\mathsf{CRH}} \leftarrow \mathsf{CRH.Setup}(1^\lambda)$.
4. Sample simulated parameters for NIZK for $\mathcal{R}_{\mathsf{e}}$: $(\mathsf{pp}_{\mathsf{e}}, \mathsf{td}_{\mathsf{e}}) \leftarrow \mathsf{NIZK.SimSetup}(1^\lambda, \mathcal{R}_{\mathsf{e}})$.
5. Set $\mathsf{pp} := (\mathsf{pp}_{\mathsf{CM}}, \mathsf{pp}_{\mathsf{TCM}}, \mathsf{pp}_{\mathsf{CRH}}, \mathsf{pp}_{\mathsf{e}})$.
6. Set $\mathsf{td} := (\mathsf{td}_{\mathsf{TCM}}, \mathsf{td}_{\mathsf{e}})$.
7. Output $(\mathsf{pp}, \mathsf{td})$.

$\mathsf{DPC.SimExecute}^{\mathbf{L}}$
Input:
- public parameters pp and trapdoor td
- old serial numbers $[\mathsf{sn}_i]_1^m$
- new record commitments $[\mathsf{cm}_j]_1^n$
- transaction memorandum memo

Output: new records $[\mathbf{r}_j]_1^n$ and transaction tx

1. For $j \in \{1, \ldots, n\}$:
   a) Set new serial number nonce $\rho_j := \mathsf{CRH.Eval}(\mathsf{pp}_{\mathsf{CRH}}, j \| \mathsf{sn}_1 \| \ldots \| \mathsf{sn}_m)$.
   b) Set address public key, payload, predicates, and commitment randomness to be the empty string: $\mathsf{apk}_j, \mathsf{payload}_j, \Phi_{\mathsf{b},j}, \Phi_{\mathsf{d},j}, r_j := \varepsilon$.
   c) Construct dummy record: $\mathbf{r}_j := (\mathsf{apk}_j, \mathsf{payload}_j, \Phi_{\mathsf{b},j}, \Phi_{\mathsf{d},j}, \rho_j, r_j, \mathsf{cm}_j)$.
2. Retrieve current ledger digest: $\mathsf{st}_{\mathbf{L}} \leftarrow \mathbf{L}.\mathsf{Digest}$.
3. Construct instance for relation $\mathcal{R}_{\mathsf{e}}$: $\mathbb{x}_{\mathsf{e}} := (\mathsf{st}_{\mathbf{L}}, [\mathsf{sn}_i]_1^m, [\mathsf{cm}_j]_1^n, \mathsf{memo})$.
4. Generate simulated proof for $\mathcal{R}_{\mathsf{e}}$: $\pi_{\mathsf{e}} \leftarrow \mathsf{NIZK.Simulate}(\mathsf{pp}_{\mathsf{e}}, \mathsf{td}_{\mathsf{e}}, \mathbb{x}_{\mathsf{e}})$.
5. Construct transaction: $\mathsf{tx} := ([\mathsf{sn}_i]_1^m, [\mathsf{cm}_j]_1^n, \mathsf{memo}, \star)$, where $\star := (\mathsf{st}_{\mathbf{L}}, \pi_{\mathsf{e}})$.
6. Output $([\mathbf{r}_j]_1^n, \mathsf{tx})$.

DPC.ExtractExecute
Input:
- public parameters pp and trapdoor td
- transaction tx

Output:
- old $\begin{cases} \text{records } [\mathbf{r}_i]_1^m \\ \text{address secret keys } [\mathsf{ask}_i]_1^m \end{cases}$
- new records $[\mathbf{r}_j]_1^n$
- auxiliary predicate input aux

1. Parse tx as $([\mathsf{sn}_i]_1^m, [\mathsf{cm}_j]_1^n, \mathsf{memo}, \star)$ and $\star$ as $(\mathsf{st}_{\mathbf{L}}, \pi_{\mathsf{e}})$.
2. Construct instance for relation $\mathcal{R}_{\mathsf{e}}$: $\mathbb{x}_{\mathsf{e}} := (\mathsf{st}_{\mathbf{L}}, [\mathsf{sn}_i]_1^m, [\mathsf{cm}_j]_1^n, \mathsf{memo})$.
3. Obtain witness: $\mathbb{w}_{\mathsf{e}} \leftarrow \mathsf{NIZK.Extract}(\mathsf{pp}_{\mathsf{e}}, \mathsf{td}_{\mathsf{e}}, \mathbb{x}_{\mathsf{e}}, \pi_{\mathsf{e}})$.
4. Parse the witness $\mathbb{w}_{\mathsf{e}}$ as $([\mathbf{r}_i]_1^m, [\mathbb{w}_{\mathbf{L},i}]_1^m, [\mathsf{ask}_i]_1^m, [\mathbf{r}_j]_1^n, \mathsf{aux})$.
5. Output $([\mathbf{r}_i]_1^m, [\mathsf{ask}_i]_1^m, [\mathbf{r}_j]_1^n, \mathsf{aux})$.

Figure 4.20: Several subroutines used by the ideal-world simulator $\mathcal{S}$.

4.10.3 Proof of security by hybrid argument

We use a sequence of hybrids, each identified by a game $\mathcal{G}_i$, to prove that the outputs of the environment $\mathcal{E}$ when interacting with the real-world (dummy) adversary $\mathcal{A}$ and the ideal-world simulator $\mathcal{S}$ are computationally indistinguishable. We denote by $\mathsf{Output}_i(\mathcal{E})$ the output of $\mathcal{E}$ in game $\mathcal{G}_i$, and by $\mathcal{G}_0$ the real-world execution.

- $\mathcal{G}_1$ (sample parameters):
  This game is the real-world execution modified as follows.
  - $\mathcal{E}$ interacts with $\mathcal{S}$ instead of $\mathcal{A}$.
  - $\mathcal{S}$ uses DPC.Setup to generate public parameters pp, and gives these to $\mathcal{E}$.
  - $\mathcal{S}$ maintains the ledger $\mathbf{L}$ for $\mathcal{E}$ (it appends to $\mathbf{L}$ any pushed transaction passing the checks in DPC.Verify).
  - $\mathcal{S}$ forwards messages from $\mathcal{E}$ to $\mathbf{L}$ and other parties.
  - $\mathcal{S}$ forwards messages from other honest parties to $\mathcal{E}$.

  $\mathsf{Output}_1(\mathcal{E})$ is perfectly indistinguishable from $\mathsf{Output}_0(\mathcal{E})$ since $\mathcal{S}$ samples the public parameters honestly, maintains the ledger identically to the ideal ledger, and otherwise behaves like the dummy adversary.

- $\mathcal{G}_2$ (simulate setup):
  $\mathcal{S}$ invokes DPC.SimSetup instead of DPC.Setup. $\mathsf{Output}_2(\mathcal{E})$ is perfectly indistinguishable from $\mathsf{Output}_1(\mathcal{E})$ since NIZK is perfect zero knowledge.

- $\mathcal{G}_3$ (simulate proofs):
  In all honest party transactions, $\mathcal{S}$ replaces NIZK proofs with simulated proofs produced via NIZK.Simulate. $\mathsf{Output}_3(\mathcal{E})$ is perfectly indistinguishable from $\mathsf{Output}_2(\mathcal{E})$ since NIZK is perfect zero knowledge.

- $\mathcal{G}_4$ (simulate serial numbers):
  In all honest party transactions, $\mathcal{S}$ replaces all serial numbers with uniformly random elements sampled from PRF's codomain. Since PRF is a pseudorandom function, and $\mathcal{E}$ does not know the secret key used to compute it, $\mathsf{Output}_4(\mathcal{E})$ is computationally indistinguishable from $\mathsf{Output}_3(\mathcal{E})$.

- $\mathcal{G}_5$ (simulate commitments and equivocate commitment openings):
  In all honest party transactions, $\mathcal{S}$ replaces record commitments with commitments to the empty string ε. In all messages from honest parties to corrupted parties containing record contents, $\mathcal{S}$ replaces the actual commitment randomness with randomness produced by TCM.Equivocate. $\mathsf{Output}_5(\mathcal{E})$ is perfectly indistinguishable from $\mathsf{Output}_4(\mathcal{E})$ since TCM is perfectly hiding and equivocation produces commitment randomness that is statistically close to uniform.

- $\mathcal{G}_6$ (handle adversarial transactions):
  For every corrupted party transaction, $\mathcal{S}$ extracts an NP instance $\mathbb{x}_{\mathsf{e}}$ and witness $\mathbb{w}_{\mathsf{e}}$ for $\mathcal{R}_{\mathsf{e}}$ from the included proof and then proceeds as follows.

- If $(\mathbb{x}_{\mathsf{e}}, \mathbb{w}_{\mathsf{e}}) \notin \mathcal{R}$, $\mathcal{S}$ aborts. If NIZK is simulation-extractable, this occurs with negligible probability.
- For all $i \in \{1, \ldots, m\}$, if the contents of any $\mathbf{r}_i$ are different from those seen in any `RecordAuth` from an honest party or in the output of a previously extracted transaction, $\mathcal{S}$ aborts. If TCM is a binding commitment scheme, then this occurs with negligible probability.
- For all $i \in \{1, \ldots, m\}$, if the extracted secret key ask_i for apk_i differs from the secret key extracted for apk_i in a prior transaction, $\mathcal{S}$ aborts. If CM is a binding commitment scheme, then this occurs with negligible probability.
- For all $j \in \{1, \ldots, n\}$, if the serial number nonce ρ_j matches one extracted in a prior transaction, $\mathcal{S}$ aborts. If CRH is a collision-resistant hash, then this occurs with negligible probability because the serial number nonce is the output of CRH evaluated (in part) over the serial numbers of the input records. If this input is distinct across two different invocations of CRH, then collision resistance guarantees that a nonce collision happens with negligible probability. Now for the transaction to be valid, it must contain serial numbers not seen before on the ledger. Therefore, the inputs to CRH are never repeated.

$\mathsf{Output}_6(\mathcal{E})$ is therefore computationally indistinguishable from $\mathsf{Output}_5(\mathcal{E})$.

The final game is distributed identically to the operation of $\mathcal{S}$ from the point of view of $\mathcal{E}$. We have thus shown that $\mathcal{E}$'s advantage in distinguishing the interaction with $\mathcal{S}$ from the interaction with $\mathcal{A}$ is negligible.

4.11 Construction of a delegable DPC scheme

We provide more details on the delegable DPC scheme discussed in Section 4.5. First we give details on randomizable signatures (Section 4.11.1), and then give pseudocode for the DPC construction (Section 4.11.2).

4.11.1 Definition and construction of a randomizable signature scheme

A *randomizable* signature scheme is a tuple of algorithms $\mathsf{SIG} = (\mathsf{Setup}, \mathsf{Keygen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{RandPk}, \mathsf{RandSig})$ that enables a party to sign messages, while also allowing randomization of public keys and signatures to prevent linking across multiple signatures.

We have already described the syntax of the scheme's algorithms, and summarized its security properties, in Section 4.5.2. Now we discuss in more detail the security properties, and the construction used in our code.

Security properties. The signature scheme SIG satisfies the following security properties.

- *Existential unforgeability under randomization (EUR).* For every efficient adversary $\mathcal{A}$, the following probability is negligible:

$$\Pr\left[\begin{array}{l} \left(m^* \notin Q \text{ and } \mathsf{SIG.Verify}(\mathsf{pp_{SIG}}, \mathsf{pk_{SIG}}, m^*, \sigma^*)\right) \\ \text{or} \\ \left(m^* \notin Q \text{ and } \mathsf{SIG.Verify}(\mathsf{pp_{SIG}}, \hat{\mathsf{pk}}_{\mathsf{SIG}}, m^*, \sigma^*)\right) \end{array} \middle| \begin{array}{r} \mathsf{pp_{SIG}} \leftarrow \mathsf{SIG.Setup}(1^\lambda) \\ (\mathsf{pk_{SIG}}, \mathsf{sk_{SIG}}) \leftarrow \mathsf{SIG.Keygen}(\mathsf{pp_{SIG}}) \\ (m^*, \sigma^*, r^*_{\mathsf{SIG}}) \leftarrow \mathcal{A}^{S(\cdot)}(\mathsf{pp_{SIG}}, \mathsf{pk_{SIG}}) \\ \hat{\mathsf{pk}}_{\mathsf{SIG}} \leftarrow \mathsf{SIG.RandPk}(\mathsf{pp_{SIG}}, \mathsf{pk_{SIG}}, r^*_{\mathsf{SIG}}) \end{array}\right]$$

  where $S(m) := \mathsf{SIG.Sign}(\mathsf{pp_{SIG}}, \mathsf{sk_{SIG}}, m)$ and Q is the set of queries made by $\mathcal{A}$ to the signing oracle S.

- *Unlinkability.* Every efficient adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ has at most negligible advantage in guessing the bit b in the IND-RSIG game below.

$\mathsf{IND\text{-}RSIG}^{\mathsf{SIG}}_{\mathcal{A}}(1^\lambda)$:
1. Generate public parameters: $\mathsf{pp_{SIG}} \leftarrow \mathsf{SIG.Setup}(1^\lambda)$.
2. Generate key pair: $(\mathsf{pk_{SIG}}, \mathsf{sk_{SIG}}) \leftarrow \mathsf{SIG.Keygen}(\mathsf{pp_{SIG}})$.
3. Obtain message from adversary: $m \leftarrow \mathcal{A}_1^{\mathsf{SIG.Sign}(\mathsf{pp_{SIG}}, \mathsf{sk_{SIG}}, \cdot)}(\mathsf{pp_{SIG}}, \mathsf{pk_{SIG}})$.
4. Sample a bit b uniformly at random.
5. If $b = 0$:
   a) Sample new key pair: $(\mathsf{pk'_{SIG}}, \mathsf{sk'_{SIG}}) \leftarrow \mathsf{SIG.Keygen}(\mathsf{pp_{SIG}})$.
   b) Sign message: $\sigma \leftarrow \mathsf{SIG.Sign}(\mathsf{pp_{SIG}}, \mathsf{sk'_{SIG}}, m)$.
   c) Set $c := (\mathsf{pk'_{SIG}}, \sigma)$.
6. If $b = 1$:
   a) Sign message: $\sigma \leftarrow \mathsf{SIG.Sign}(\mathsf{pp_{SIG}}, \mathsf{sk_{SIG}}, m)$.
   b) Sample randomness r_{SIG}.
   c) Randomize public key: $\hat{\mathsf{pk}}_{\mathsf{SIG}} \leftarrow \mathsf{SIG.RandPk}(\mathsf{pp_{SIG}}, \mathsf{pk_{SIG}}, r_{\mathsf{SIG}})$.
   d) Randomize signature: $\hat{\sigma} \leftarrow \mathsf{SIG.RandSig}(\mathsf{pp_{SIG}}, \sigma, r_{\mathsf{SIG}})$.
   e) Set $c := (\hat{\mathsf{pk}}_{\mathsf{SIG}}, \hat{\sigma})$.
7. Output $\mathcal{A}_2^{\mathsf{SIG.Sign}(\mathsf{pp_{SIG}}, \mathsf{sk_{SIG}}, \cdot)}(\mathsf{pp_{SIG}}, \mathsf{pk_{SIG}}, c)$.

- *Injective randomization.* For every efficient adversary $\mathcal{A}$, the following probability is negligible:

$$\Pr\left[\begin{array}{c} r_1 \neq r_2 \\ \mathsf{SIG.RandPk}(\mathsf{pp_{SIG}}, \mathsf{pk_{SIG}}, r_1) = \mathsf{SIG.RandPk}(\mathsf{pp_{SIG}}, \mathsf{pk_{SIG}}, r_2) \end{array} \middle| \begin{array}{c} \mathsf{pp_{SIG}} \leftarrow \mathsf{SIG.Setup}(1^\lambda) \\ (\mathsf{pk_{SIG}}, r_1, r_2) \leftarrow \mathcal{A}(\mathsf{pp_{SIG}}) \end{array}\right].$$

Construction. In Fig. 4.21 we provide a modification of the Schnorr signature scheme [Sch91] that is randomizable. We briefly explain why this modification satisfies the security properties above.

- *Existential unforgeability under randomization (EUR).* Given an efficient adversary $\mathcal{A}$ that breaks EUR of randomizable Schnorr signatures, we construct an efficient adversary $\mathcal{A}'$ that breaks existential unforgeability of standard Schnorr signatures. In detail, $\mathcal{A}'$ forwards signature queries from $\mathcal{A}$ to its own signing oracle and returns the answers to $\mathcal{A}$ and then, when $\mathcal{A}$ outputs a tuple $(m^*, \sigma^*, r^*_{\mathsf{SIG}})$, $\mathcal{A}'$ outputs the tuple (m^*, σ) where σ is computed as follows. If σ^* is a valid signature for m^* under $\mathsf{pk_{SIG}}$ then $\sigma := \sigma^*$. Otherwise, $\mathcal{A}'$ "undoes" the randomization of $\sigma^* = (s, e)$ by setting $\sigma := (s + e \cdot r^*_{\mathsf{SIG}}, e)$; thus if $\mathcal{A}$ outputs a forgery for a randomization of $\mathsf{pk_{SIG}}$, $\mathcal{A}'$ translates it back into a forgery for $\mathsf{pk_{SIG}}$. In sum, since standard Schnorr signatures are secure in the random oracle model assuming hardness of discrete logarithms [PS00], so is the randomizable variant under the same assumptions.

- *Unlinkability of public keys.* Public keys are unlinkable because SIG.RandPk multiplies the public key pk (which is a group element) by a random group element; the result is statistically independent of pk.

- *Unlinkability of signatures.* The only part of a Schnorr signature that depends on the public or secret key is the scalar s. Since SIG.RandSig adds a random shift to s, the result is statistically independent of the signature's original key pair.

- *Injective randomization.* Fixing all inputs but for r_{SIG}, SIG.RandPk is a permutation over $\mathbb{G}$. Hence, finding collisions over the randomness is not possible.

$\mathsf{SIG.Setup}(1^\lambda) \rightarrow \mathsf{pp_{SIG}}$
1. Sample a group: $(\mathbb{G}, q, g) \leftarrow \mathsf{SampleGrp}(1^\lambda)$.
2. Sample cryptographic hash function H.
3. Output $\mathsf{pp_{SIG}} := (\mathbb{G}, q, g, H)$.

$\mathsf{SIG.Keygen}(\mathsf{pp_{SIG}}) \rightarrow (\mathsf{pk_{SIG}}, \mathsf{sk_{SIG}})$
1. Parse $\mathsf{pp_{SIG}}$ as $(\mathbb{G}, q, g, H)$.
2. Sample a scalar x uniformly from $\mathbb{Z}_q$.
3. Output $(\mathsf{pk_{SIG}}, \mathsf{sk_{SIG}}) := (g^x, x)$.

$\mathsf{SIG.Verify}(\mathsf{pp_{SIG}}, \mathsf{pk_{SIG}}, m, \sigma) \rightarrow b$
1. Parse $\mathsf{pp_{SIG}}$ as $(\mathbb{G}, q, g, H)$.
2. Parse σ as (s, e).
3. Set $r_v := g^s \mathsf{pk}^e_{\mathsf{SIG}} = g^{s+xe}$.
4. Set $e_v := H(r_v \| m)$.
5. Check if $e = e_v$.

$\mathsf{SIG.Sign}(\mathsf{pp_{SIG}}, \mathsf{sk_{SIG}}, m) \rightarrow \sigma$
1. Parse $\mathsf{pp_{SIG}}$ as $(\mathbb{G}, q, g, H)$.
2. Sample a scalar k uniformly from $\mathbb{Z}_q$.
3. Set $r := g^k$ and $e := H(r \| m)$.
4. Set $s := k - xe$.
5. Output $\sigma := (s, e)$.

$\mathsf{SIG.RandPk}(\mathsf{pp_{SIG}}, \mathsf{pk_{SIG}}, r_{\mathsf{SIG}}) \rightarrow \hat{\mathsf{pk}}_{\mathsf{SIG}}$
1. Parse $\mathsf{pp_{SIG}}$ as $(\mathbb{G}, q, g, H)$.
2. Output $\hat{\mathsf{pk}}_{\mathsf{SIG}} := \mathsf{pk_{SIG}} \cdot g^{r_{\mathsf{SIG}}}$.

$\mathsf{SIG.RandSig}(\mathsf{pp_{SIG}}, \sigma, r_{\mathsf{SIG}}) \rightarrow \hat{\sigma}$
1. Parse $\mathsf{pp_{SIG}}$ as $(\mathbb{G}, q, g, H)$.
2. Parse σ as (s, e).
3. Output $\hat{\sigma} := (s - e \cdot r_{\mathsf{SIG}}, e)$.

Figure 4.21: Construction of a randomizable signature scheme based on the Schnorr signature scheme [Sch91].

4.11.2 Construction of a delegable DPC scheme

Fig. 4.22 provides pseudocode that, together with the modified NP relation $\mathcal{R}_{\mathsf{e}}^{\mathsf{del}}$ given in Fig. 4.23, formalizes the high-level description of a delegable DPC scheme from Section 4.5.3. In both figures, we highlighted changes from the "plain" DPC scheme in Section 4.4.2. *The only step in* DPC.Execute *that must be performed by the delegator is Step 7a; all other steps can be performed by the worker without knowing the signature secret key.*

DPC.Setup

Input: security parameter 1^λ

Output: public parameters pp

1. Generate **commitment parameters**: $\mathsf{pp}_{\mathsf{CM}} \leftarrow \mathsf{CM.Setup}(1^\lambda)$, $\mathsf{pp}_{\mathsf{TCM}} \leftarrow \mathsf{TCM.Setup}(1^\lambda)$.
2. Generate **CRH parameters**: $\mathsf{pp}_{\mathsf{CRH}} \leftarrow \mathsf{CRH.Setup}(1^\lambda)$.
3. Generate **signature parameters**: $\mathsf{pp}_{\mathsf{SIG}} \leftarrow \mathsf{SIG.Setup}(1^\lambda)$.
4. Generate **NIZK parameters for** $\mathcal{R}_{\mathsf{e}}^{\mathsf{del}}$ (Fig. 4.23): $\mathsf{pp}_{\mathsf{e}} \leftarrow \mathsf{NIZK.Setup}(1^\lambda, \mathcal{R}_{\mathsf{e}}^{\mathsf{del}})$.
5. Output $\mathsf{pp} := (\mathsf{pp}_{\mathsf{CM}}, \mathsf{pp}_{\mathsf{TCM}}, \mathsf{pp}_{\mathsf{CRH}}, \mathsf{pp}_{\mathsf{SIG}}, \mathsf{pp}_{\mathsf{e}})$.

DPC.GenAddress

Input: public parameters pp

Output: address key pair $(\mathsf{apk}, \mathsf{ask})$

1. Generate **authorization key pair**: $(\mathsf{pk}_{\mathsf{SIG}}, \mathsf{sk}_{\mathsf{SIG}}) \leftarrow \mathsf{SIG.Keygen}(\mathsf{pp}_{\mathsf{SIG}})$.
2. Sample secret key $\mathsf{sk}_{\mathsf{PRF}}$ for pseudorandom function PRF.
3. Sample randomness r_{pk} for commitment scheme TCM.
4. Set **address public key** $\mathsf{apk} := \mathsf{CM.Commit}(\mathsf{pp}_{\mathsf{CM}}, \mathsf{pk}_{\mathsf{SIG}} \| \mathsf{sk}_{\mathsf{PRF}};\ r_{\mathsf{pk}})$.
5. Set **address secret key** $\mathsf{ask} := (\mathsf{sk}_{\mathsf{SIG}}, \mathsf{sk}_{\mathsf{PRF}}, r_{\mathsf{pk}})$.
6. Output $(\mathsf{apk}, \mathsf{ask})$.

DPC.Execute$^{\mathbf{L}}$

Input:

- public parameters pp
- old $\begin{cases} \text{records } [\mathbf{r}_i]_1^m \\ \text{address secret keys } [\mathsf{ask}_i]_1^m \end{cases}$
- auxiliary predicate input aux
- transaction memorandum memo
- new $\begin{cases} \text{address public keys } [\mathsf{apk}_j]_1^n \\ \text{record payloads } [\mathsf{payload}_j]_1^n \\ \text{record birth predicates } [\Phi_{\mathsf{b},j}]_1^n \\ \text{record death predicates } [\Phi_{\mathsf{d},j}]_1^n \end{cases}$

Output: new records $[\mathbf{r}_j]_1^n$ and transaction tx

1. For each $i \in \{1, \ldots, m\}$, process the i-th old record as follows:
   a) Parse old record $\mathbf{r}_i$ as $\mathbf{r}_i = \begin{pmatrix} \text{address public key} & \mathsf{apk}_i & \text{payload} & \mathsf{payload}_i & \text{comm. rand.} & r_i \\ \text{serial number nonce} & \rho_i & \text{predicates} & (\Phi_{\mathsf{b},i}, \Phi_{\mathsf{d},i}) & \text{commitment} & \mathsf{cm}_i \end{pmatrix}$.
   b) If $\mathsf{payload}_i.\mathsf{isDummy} = 1$, set **ledger membership witness** $\mathbb{w}_{\mathbf{L},i} := \perp$.
   If $\mathsf{payload}_i.\mathsf{isDummy} = 0$, compute **ledger membership witness** for commitment: $\mathbb{w}_{\mathbf{L},i} \leftarrow \mathbf{L.Prove}(\mathsf{cm}_i)$.
   c) Parse address secret key ask_i as $(\mathsf{sk}_{\mathsf{SIG},i}, \mathsf{sk}_{\mathsf{PRF},i}, r_{\mathsf{pk},i})$ and derive $\mathsf{pk}_{\mathsf{SIG},i}$ from $\mathsf{sk}_{\mathsf{SIG},i}$.
   d) Compute **signature randomness**: $r_{\mathsf{SIG},i} \leftarrow \mathsf{PRF}_{\mathsf{sk}_{\mathsf{PRF},i}}(\rho_i)$.
   e) Compute **serial number**: $\mathsf{sn}_i \leftarrow \mathsf{SIG.RandPk}(\mathsf{pp}_{\mathsf{SIG}}, \mathsf{pk}_{\mathsf{SIG},i}, r_{\mathsf{SIG},i})$.
2. For each $j \in \{1, \ldots, n\}$, construct the j-th new record as follows:
   a) Compute **serial number nonce**: $\rho_j := \mathsf{CRH.Eval}(\mathsf{pp}_{\mathsf{CRH}}, j \| \mathsf{sn}_1 \| \ldots \| \mathsf{sn}_m)$.
   b) Construct **new record**: $\mathbf{r}_j \leftarrow \mathsf{DPC.ConstructRecord}(\mathsf{pp}, \mathsf{apk}_j, \mathsf{payload}_j, \Phi_{\mathsf{b},j}, \Phi_{\mathsf{d},j}, \rho_j)$.
3. Retrieve current **ledger digest**: $\mathsf{st}_{\mathbf{L}} \leftarrow \mathbf{L}.\mathsf{Digest}$.
4. Construct **instance for relation** $\mathcal{R}_{\mathsf{e}}^{\mathsf{del}}$: $\mathbb{x}_{\mathsf{e}} := (\mathsf{st}_{\mathbf{L}}, [\mathsf{sn}_i]_1^m, [\mathsf{cm}_j]_1^n, \mathsf{memo})$.
5. Construct **witness for relation** $\mathcal{R}_{\mathsf{e}}^{\mathsf{del}}$: $\mathbb{w}_{\mathsf{e}} := ([\mathbf{r}_i]_1^m, [\mathbb{w}_{\mathbf{L},i}]_1^m, [\mathsf{sk}_{\mathsf{PRF},i}]_1^m, [\mathsf{pk}_{\mathsf{SIG},i}]_1^m, [\mathsf{meta}_i]_1^m, [r_{\mathsf{pk},i}]_1^m, [\mathbf{r}_j]_1^n, \mathsf{aux})$.
6. Generate **proof for relation** $\mathcal{R}_{\mathsf{e}}^{\mathsf{del}}$: $\pi_{\mathsf{e}} \leftarrow \mathsf{NIZK.Prove}(\mathsf{pp}_{\mathsf{e}}, \mathbb{x}_{\mathsf{e}}, \mathbb{w}_{\mathsf{e}})$.
7. For each $i \in \{1, \ldots, m\}$:
   a) **Sign message**: $\sigma_i \leftarrow \mathsf{SIG.Sign}(\mathsf{pp}_{\mathsf{SIG}}, \mathsf{sk}_{\mathsf{SIG},i}, \mathbb{x}_{\mathsf{e}} \| \pi_{\mathsf{e}})$.
   b) **Randomize signature**: $\hat{\sigma}_i \leftarrow \mathsf{SIG.RandSig}(\mathsf{pp}_{\mathsf{SIG}}, \sigma_i, r_{\mathsf{SIG},i})$.
8. Construct **transaction**: $\mathsf{tx} := ([\mathsf{sn}_i]_1^m, [\mathsf{cm}_j]_1^n, \mathsf{memo}, \star)$, where $\star := (\mathsf{st}_{\mathbf{L}}, \pi_{\mathsf{e}}, [\hat{\sigma}_i]_1^m)$.
9. Output $([\mathbf{r}_j]_1^n, \mathsf{tx})$.

DPC.Verify$^{\mathbf{L}}$

Input: public parameters pp and transaction tx

Output: decision bit b

1. Parse tx as $([\mathsf{sn}_i]_1^m, [\mathsf{cm}_j]_1^n, \mathsf{memo}, \star)$ and $\star$ as $(\mathsf{st}_{\mathbf{L}}, \pi_{\mathsf{e}}, [\hat{\sigma}_i]_1^m)$.
2. Check that **there are no duplicate serial numbers**
   a) within the transaction tx: $\mathsf{sn}_i \neq \mathsf{sn}_j$ for every distinct $i, j \in \{1, \ldots, m\}$;
   b) on the ledger: $\mathbf{L}.\mathsf{Contains}(\mathsf{sn}_i) = 0$ for every $i \in \{1, \ldots, m\}$.
3. Check that **the ledger state is valid**: $\mathbf{L}.\mathsf{ValidateDigest}(\mathsf{st}_{\mathbf{L}}) = 1$.
4. Construct **instance for the relation** $\mathcal{R}_{\mathsf{e}}^{\mathsf{del}}$: $\mathbb{x}_{\mathsf{e}} := (\mathsf{st}_{\mathbf{L}}, [\mathsf{sn}_i]_1^m, [\mathsf{cm}_j]_1^n, \mathsf{memo})$.
5. Check **proof for the relation** $\mathcal{R}_{\mathsf{e}}^{\mathsf{del}}$: $\mathsf{NIZK.Verify}(\mathsf{pp}_{\mathsf{e}}, \mathbb{x}_{\mathsf{e}}, \pi_{\mathsf{e}}) = 1$.
6. For every $i \in \{1, \ldots, m\}$, check that **signature verifies**: $\mathsf{SIG.Verify}(\mathsf{pp}_{\mathsf{SIG}}, \mathsf{sn}_i, \mathbb{x}_{\mathsf{e}} \| \pi_{\mathsf{e}}, \hat{\sigma}_i) = 1$.

Figure 4.22: Construction of a delegable DPC scheme. Highlights denote differences from Figure 4.8.

The NP relation $\mathcal{R}_{\mathsf{e}}^{\mathsf{del}}$ has instances $\mathbb{x}_{\mathsf{e}}$ and witnesses $\mathbb{w}_{\mathsf{e}}$ of the following form.

$$\mathbb{x}_{\mathsf{e}} = \begin{pmatrix} \text{ledger digest} & \mathsf{st}_{\mathbf{L}} \\ \text{old record serial numbers} & [\mathsf{sn}_i]_1^m \\ \text{new record commitments} & [\mathsf{cm}_j]_1^n \\ \text{transaction memorandum} & \mathsf{memo} \end{pmatrix} \quad \text{and} \quad \mathbb{w}_{\mathsf{e}} = \begin{pmatrix} \text{old records} & [\mathbf{r}_i]_1^m \\ \text{old record membership witnesses} & [\mathbb{w}_{\mathbf{L},i}]_1^m \\ \text{old record authorization public keys} & [\mathsf{sk}_{\mathsf{PRF},i}]_1^m \\ \text{old record serial number secret keys} & [\mathsf{pk}_{\mathsf{SIG},i}]_1^m \\ \text{old record address randomness} & [r_{\mathsf{pk},i}]_1^m \\ \text{new records} & [\mathbf{r}_j]_1^n \\ \text{auxiliary predicate input} & \mathsf{aux} \end{pmatrix}$$

where

- for each $i \in \{1, \ldots, m\}$, $\mathbf{r}_i = (\mathsf{apk}_i, \mathsf{payload}_i, \Phi_{\mathsf{b},i}, \Phi_{\mathsf{d},i}, \rho_i, r_i, \mathsf{cm}_i)$;
- for each $j \in \{1, \ldots, n\}$, $\mathbf{r}_j = (\mathsf{apk}_j, \mathsf{payload}_j, \Phi_{\mathsf{b},j}, \Phi_{\mathsf{d},j}, \rho_j, r_j, \mathsf{cm}_j)$.

Define the local data $\mathsf{ldata} := \begin{pmatrix} [\mathsf{cm}_i]_1^m & [\mathsf{apk}_i]_1^m & [\mathsf{payload}_i]_1^m & [\Phi_{\mathsf{d},i}]_1^m & [\Phi_{\mathsf{b},i}]_1^m & [\mathsf{sn}_i]_1^m & \mathsf{memo} \\ [\mathsf{cm}_j]_1^n & [\mathsf{apk}_j]_1^n & [\mathsf{payload}_j]_1^n & [\Phi_{\mathsf{d},j}]_1^n & [\Phi_{\mathsf{b},j}]_1^n & \mathsf{aux} & \end{pmatrix}$.

A witness $\mathbb{w}_{\mathsf{e}}$ is valid for an instance $\mathbb{x}_{\mathsf{e}}$ if the following conditions hold:

1. For each $i \in \{i, \ldots, m\}$:
   - If $\mathbf{r}_i$ is not dummy, $\mathbb{w}_{\mathbf{L},i}$ proves that the commitment cm_i is in a ledger with digest $\mathsf{st}_{\mathbf{L}}$: $\mathbf{L}.\mathsf{Verify}(\mathsf{st}_{\mathbf{L}}, \mathsf{cm}_i, \mathbb{w}_{\mathbf{L},i}) = 1$.
   - The address public key apk_i matches the authorization public key $\mathsf{pk}_{\mathsf{SIG},i}$ and the serial number secret key $\mathsf{sk}_{\mathsf{PRF},i}$: $\mathsf{apk}_i = \mathsf{CM}.\mathsf{Commit}(\mathsf{pp}_{\mathsf{CM}}, \mathsf{pk}_{\mathsf{SIG},i} \| \mathsf{sk}_{\mathsf{PRF},i}; r_{\mathsf{pk},i})$.
   - The serial number sn_i is valid: $r_{\mathsf{SIG},i} = \mathsf{PRF}_{\mathsf{sk}_{\mathsf{PRF},i}}(\rho_i)$ and $\mathsf{sn}_i = \mathsf{SIG}.\mathsf{RandPk}(\mathsf{pp}_{\mathsf{SIG}}, \mathsf{pk}_{\mathsf{SIG},i}, r_{\mathsf{SIG},i})$.
   - The old record commitment cm_i is valid: $\mathsf{cm}_i = \mathsf{TCM}.\mathsf{Commit}(\mathsf{pp}_{\mathsf{TCM}}, \mathsf{apk}_i \| \mathsf{payload}_i \| \Phi_{\mathsf{b},i} \| \Phi_{\mathsf{d},i} \| \rho_i;\ r_i)$.
   - The death predicate $\Phi_{\mathsf{d},i}$ is satisfied by the local data: $\Phi_{\mathsf{d},i}(i \| \mathsf{ldata}) = 1$.
2. For each $j \in \{1, \ldots, n\}$:
   - The serial number nonce ρ_j is computed correctly: $\rho_j = \mathsf{CRH}.\mathsf{Eval}(\mathsf{pp}_{\mathsf{CRH}}, j \| \mathsf{sn}_1 \| \ldots \| \mathsf{sn}_m)$.
   - The new record commitment cm_j is valid: $\mathsf{cm}_j = \mathsf{TCM}.\mathsf{Commit}(\mathsf{pp}_{\mathsf{TCM}}, \mathsf{apk}_j \| \mathsf{payload}_j \| \Phi_{\mathsf{b},j} \| \Phi_{\mathsf{d},j} \| \rho_j;\ r_j)$.
   - The birth predicate $\Phi_{\mathsf{b},j}$ is satisfied by the local data: $\Phi_{\mathsf{b},j}(j \| \mathsf{ldata}) = 1$.

Figure 4.23: The NP relation $\mathcal{R}_{\mathsf{e}}^{\mathsf{del}}$. Highlights denote differences from Figure 4.9.

4.12 Extensions in functionality and in security

We summarize some natural extensions of our DPC construction that give richer functionality, as well as methods to prove security notions beyond standalone non-adaptive security.

Storing data in addresses. For some applications it can be useful to verifiably associate address public keys with additional metadata meta. One can easily modify our construction to achieve this by using the address public key commitment to additionally commit to meta. To prove that a given address public key is bound to the metadata string meta, one can use a standard non-interactive zero knowledge proof of knowledge.[14]

With such a mechanism in hand, we can realize various useful functionality like *on-ledger encryption*: a user stores an encryption public key in the metadata of one of her addresses, and others can later use this public key to encrypt information about records created for her, and store the resulting ciphertext in the transaction's memorandum. This method, used for example in Zerocash [Ben+14], gives users the option to not use other out-of-band secure communication channels.

Selective disclosure. For compliance purposes, it may be useful to selectively reveal information about a transaction to certain parties. Our implementation can be extended to support this by changing how hashes of predicate verification keys are committed to in a transaction: instead of committing all the verification keys together, one can instead commit to them in separate commitments. To disclose the predicates that were invoked in a transaction, a user can then simply open the relevant commitments.

Ledger position. In some applications it may be useful to know the unique ledger position of a record, i.e., to have this information be part of the local data ldata given as input to predicates. For example, one can use a record's ledger position to implement a "time lock" that prevents the record's consumption until a pre-specified amount of time has passed since the record's creation. However, the ledger interface we described in Section 4.3.1 does not expose this functionality: $\mathbf{L}.\mathsf{Prove}$ only returns a proof that a transaction (or a subcomponent thereof) appears on the ledger, and *not its position*. One can augment $\mathbf{L}.\mathsf{Prove}$ to instead output the transaction's ledger position $\mathsf{pos}_{\mathbf{L}}$, and a proof that $\mathsf{pos}_{\mathbf{L}}$ is the transaction's position on the ledger. Our instantiation of the ledger with a Merkle tree supports this augmentation inherently: the path to the transaction in the Merkle tree is also its position the tree.

Composable security. The security definition in Section 4.3.3 is a restriction of UC security definitions to a single execution at any given time. We can avoid this restriction and prove our construction UC-secure by replacing our simulation-extractable NIZKs with UC-secure NIZKs. The remainder of the proof would go through unchanged, and this would achieve composition of multiple protocol instances.

Adaptive security. We can prove adaptive security, with a minor modification to our protocol in Section 4.4. The barrier to proving security against adaptive corruptions (even in a standalone setting) is a lack of forward-secure privacy. Namely, when the adversary corrupts a party $\mathcal{P}$, it gets

[14] We do not need these NIZK proofs to be simulation-extractable since we do not extract from them. In fact, in our implementation we can even use specific sigma-protocols designed to prove knowledge of openings of Pedersen commitments.

access to $\mathcal{P}$'s state, which includes contents of records held by $\mathcal{P}$ and address secret keys belonging to $\mathcal{P}$. The adversary can then use this information to break unlinkability of $\mathcal{P}$'s transactions by deriving the serial numbers of consumed records and matching these against those present on the ledger.

In the proof, this problem is reflected in how the simulator $\mathcal{S}$ handles serial numbers in honest party transactions (see Section 4.10.2). For honest party transactions, serial numbers are sampled uniformly at random via $\mathsf{SampleSn}$. When the environment $\mathcal{E}$ corrupts an honest party, it can attempt to carry out the aforementioned linking attack by computing serial numbers via the PRF. Since serial numbers already published in transactions were derived randomly, they would not match the output of the PRF, allowing $\mathcal{E}$ to distinguish the ideal world from the real world.

We address this issue as follows. First, we work in the secure-erasure model and ensure that honest parties delete (a) all records output from $\mathsf{Execute}$ (after sending their contents to the intended recipients), and (b) all records that have been consumed. Hence, at the time a party is corrupted, the state revealed to the adversary does not contain secrets of past records, so the adversary cannot derive those records' serial numbers. Next, we have to convincingly match the address public keys of unconsumed records with corresponding address secret keys. To do this, we modify $\mathsf{DPC.GenAddress}$ to use trapdoor commitments to construct address public keys. The trapdoor property then allows us to open public keys to the correct secret keys.

However, these measures by themselves are not enough. Consider the following scenario: the adversary corrupts an honest user and learns her secret key. For every transaction in the ledger, it computes the serial number nonces of the output records from the serial numbers of the input records. The adversary can then use these nonces along with the secret key to derive candidate serial numbers for the output records. If these candidate serial numbers appear on the ledger, then the adversary learns that the record has been consumed.

To prevent this, we randomize the serial number nonces of all records output by $\mathsf{Execute}$ by deriving them as $\rho_j := \mathsf{CRH}(j \| r_{\rho,j} \| \mathsf{sn}_1 \| \cdots \| \mathsf{sn}_m)$ for some randomness $r_{\rho,j}$ that is deleted after invoking $\mathsf{Execute}$. This randomization ensures that the serial number nonce of an output record cannot be derived deterministically from the (publicly visible) serial numbers of the input records.

The above measures, however, are still insufficient: the adversary still knows the secrets of records that a corrupted party sent to an honest party. After corrupting this honest party, the adversary can learn its address secret key and therefore derive the serial number of those records. To overcome this obstacle, one can replace the PRF with a *programmable PRF* [PS18], for which the owner of the secret key can "program" the PRF to output pre-determined values on specific inputs: for all polynomial-sized sets $S = \{(x_i, y_i)\}_i$, the owner of a PRF secret key sk can derive a second key sk_S such that $\mathsf{PRF}_{\mathsf{sk}_s}(x_i) = y_i$ for each $(x_i, y_i) \in S$, while $\mathsf{PRF}_{\mathsf{sk}_S}(x) = \mathsf{PRF}_{\mathsf{sk}}(x)$ for other inputs x. This fixes the foregoing issue because $\mathcal{S}$ can now give $\mathcal{E}$ a programmed PRF secret key for the set $S = \{(\rho_i, \mathsf{sn}_i)\}_i$, where ρ_i is the serial number nonce of the i-th record received from a corrupted party.

Chapter 5

Impact and adoption

The work in this dissertation has resulted in both academic and industrial impact. Below we first elaborate on the adoption of Marlin (Section 5.1) and Zexe (Section 5.2), and then, in Section 5.3, we describe `arkworks`, our open source Rust ecosystem for programming zkSNARKs.

5.1 Marlin

The AHP abstraction and concrete AHPs constructed in [CHMMVW20] have spurred a number of novel academic works on universal zkSNARKs:

- Fractal[COS20] adapts the holographic lincheck of Marlin to the IOP setting, and uses it to construct a transparent zkSNARK in the ROM.
- Claymore [SZ20] constructs new AHPs in the monomial basis.
- Bünz et al. [BCMS20; BCLMS21] show how to construct PCD from AHP-based SNARKs via the new paradigm of accumulation, and Darlin [HGD21] provides an end-to-end construction instantiating this paradigm with the Marlin AHP.
- Lunar [CFFQR20] constructs new AHPs that achieve better efficiency.
- Ràfols and Zapico [RZ21] and Zhang et al. [ZZWG21] define new generalizations of AHPs and Marlin's AHP-to-SNARK compiler, and provide constructions which, after compilation, achieve better proof size and prover time.
- Eclipse [ABCGOT21] shows how to add commit-and-prove capabilities to AHP-based SNARKs.

Marlin's efficiency and succinctness properties have also resulted in it being used for applications: it is used to construct PCD in [CCDW20], and auditable MPC in [KZGM21]. Finally, it has also seen deployment in industrial projects: Aleo [Aleo] is an implementation of Zexe [BCGMMW20] that uses Marlin to implement its birth and death predicates, while Horizen Network [Horizen] uses Marlin as part of its PCD implementation [HGD21].

5.2 Zexe

Zexe's construction of decentralized private computation (DPC) schemes is being deployed in a number of cryptocurrency projects [Aleo; Aztec; Mir], and some projects are considering using our construction of privacy-preserving DEXs as well [Anoma]. Subsequent academic works have developed new techniques to construct two-chains of curves, such as [EG20], which constructs a new "outer" curve for BLS12-377. These curves have been deployed in industrial projects [Gab+20].

5.3 arkworks

We have developed our initial implementation of Zexe into `arkworks` [con], a state-of-the-art open source Rust ecosystem for zkSNARK development. This ecosystem provides all the components required for zkSNARK programming, organized into generic, efficient, and easy-to-use modules, such as:

- Generic implementations of efficient algorithms for finite fields, elliptic curves, and pairings, as well as instantiations of widely-used curves.
- State-of-the-art zkSNARKs, including Marlin (see Chapter 2 and [CHMMVW20]), as well as those of Groth [Gro16], and Groth and Maller [GM17].
- Ergonomic interfaces for expressing a computation as arithmetic constraints (the "language" of zkSNARKs).
- Recursive composition of arbitrary SNARKs, including recursion from atomic and split accumulation schemes (that is, the constructions in Chapter 3 and [BGH19; BCMS20; BCLMS21]).
- Libraries for aggregating proofs and signatures [BMMTV21].

The modular design of `arkworks` means that improvements in one component (such as finite field arithmetic) are inherited "for free" by downstream components (such as zkSNARK implementations). We achieve this composability without sacrificing performance: our generic libraries are competitive with the best specialized libraries.

The `arkworks` ecosystem is open-source, and has a vibrant contributor community with over fifty unique contributors that have collectively contributed over 800 pull requests spanning $120,000$ lines of Rust code. Our community has also developed a tutorial to help beginners get started with programming zkSNARKs.[1]

As a result of our open source ethos and modular and efficient design, `arkworks` libraries have seen deployment in a number of state-of-the-art industry projects, such as:

- Mina [Mina] is a blockchain that uses proof-carrying data to provide a short verifiable digest of the entire chain's history. Their implementation of proof-carrying-data [Pickles20] builds on top of our finite field (`ark-ff`[2]) and elliptic curve (`ark-ec`[2]) libraries.
- Celo [Celo] is a blockchain that has developed a custom protocol [Gab+20] to aggregate BLS signature [BLS04]. The implementation of this protocol relies on our libraries for the Groth16 SNARK

[1] `https://github.com/arkworks-rs/r1cs-tutorial/`

[2] `https://github.com/arkworks-rs/algebra`

(`ark-groth16`[3]) and constraint gadgets (`ark-r1cs-std`[4] and `ark-crypto-primitives`[5]).
- Anoma Network [Anoma] uses `ark-ec` to implement threshold encryption.
- Manta Network [Manta] uses `ark-r1cs-std` and `ark-crypto-primitives` to implement private transactions and a private DEX.
- ZoKrates [ET18; ZoKrates] and Noir [Noir] are domain-specific languages designed for programming zkSNARKs. Their reference compilers include a backend that produces `arkworks` constraints by relying on `ark-r1cs-std` and `ark-crypto-primitives`.

A number of academic works have also used `arkworks` libraries to implement and evaluate their cryptographic protocols [BCGMMW20; CHMMVW20; CCDW20; BCLMS21; BMMTV21; ZX21; TFBT21; CXZ21; FQZDC21; GJMMST21; KMSV21; Ili+21].

[3] `https://github.com/arkworks-rs/groth16`

[4] `https://github.com/arkworks-rs/r1cs-std`

[5] `https://github.com/arkworks-rs/crypto-primitives`

Bibliography

[ABCGOT21] D. F. Aranha, E. M. Bennedsen, M. Campanelli, C. Ganesh, C. Orlandi, and A. Takahashi. "ECLIPSE: Enhanced Compiling method for Pedersen-committed zkSNARK Engines". Cryptology ePrint Archive, Report 2021/934. 2021.

[ABLSZ19] B. Abdolmaleki, K. Baghery, H. Lipmaa, J. Siim, and M. Zajac. "UC-Secure CRS Generation for SNARKs". In: *Proceedings of the 11th International Conference on Cryptology in Africa*. AFRICACRYPT '19. 2019, pp. 99–117.

[ADMM14a] M. Andrychowicz, S. Dziembowski, D. Malinowski, and L. Mazurek. "Fair Two-Party Computations via Bitcoin Deposits". In: *Proceedings of the BITCOIN workshop at the 18th International Conference on Financial Cryptography and Data Security*. FC '14. 2014, pp. 105–121.

[ADMM14b] M. Andrychowicz, S. Dziembowski, D. Malinowski, and L. Mazurek. "Secure Multiparty Computations on Bitcoin". In: *Proceedings of the 35th IEEE Symposium on Security and Privacy*. S&P '14. 2014, pp. 443–458.

[AFKMR12] D. F. Aranha, L. Fuentes-Castañeda, E. Knapp, A. Menezes, and F. Rodríguez-Henríquez. "Implementing Pairings at the 192-Bit Security Level". In: *Proceedings of the 5th International Conference on Pairing-Based Cryptography*. Pairing '12. 2012, pp. 177–195.

[AHIV17] S. Ames, C. Hazay, Y. Ishai, and M. Venkitasubramaniam. "Ligero: Lightweight Sublinear Arguments Without a Trusted Setup". In: *Proceedings of the 24th ACM Conference on Computer and Communications Security*. CCS '17. 2017, pp. 2087–2104.

[AirSwap] "AirSwap". `https://www.airswap.io/`. Accessed 2018-12-27.

[AKRSC13] E. Androulaki, G. Karame, M. Roeschlin, T. Scherer, and S. Capkun. "Evaluating User Privacy in Bitcoin". In: *Proceedings of the 17th International Conference on Financial Cryptography and Data Security*. FC '13. 2013, pp. 34–51.

[Aleo] "Aleo". `https://aleo.org`.

[ALMSS98] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. "Proof verification and the hardness of approximation problems". In: *Journal of the ACM* 45.3 (1998), pp. 501–555.

[Anoma] "Anoma Network". `https://anoma.network/`. 2020.

[ANWW13] J. Aumasson, S. Neves, Z. Wilcox-O'Hearn, and C. Winnerlein. "BLAKE2: Simpler, Smaller, Fast as MD5". In: *Proceedings of the 11th International Conference on Applied Cryptography and Network Security*. ACNS '13. 2013, pp. 119–135.

[AS98] S. Arora and S. Safra. "Probabilistic checking of proofs: a new characterization of NP". In: *Journal of the ACM* 45.1 (1998), pp. 70–122.

[Aztec] "Aztec Protocol". `https://aztec.network`.

[Bab85] L. Babai. "Trading group theory for randomness". In: *Proceedings of the 17th Annual ACM Symposium on Theory of Computing*. STOC '85. 1985, pp. 421–429.

[BAZB20] B. Bünz, S. Agrawal, M. Zamani, and D. Boneh. "Zether: Towards Privacy in a Smart Contract World". In: *Proceedings of the 20th International Conference on Financial Cryptography and Data Security*. FC '20. 2020, pp. 423–443.

[BB04] D. Boneh and X. Boyen. "Short Signatures Without Random Oracles". In: *Proceedings of the 23rd Annual International Conference on Theory and Application of Cryptographic Techniques*. EUROCRYPT '04. 2004, pp. 56–73.

[BBBPWM18] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell. "Bulletproofs: Short Proofs for Confidential Transactions and More". In: *Proceedings of the 39th IEEE Symposium on Security and Privacy*. S&P '18. 2018, pp. 315–334.

[BBDJLZ17] I. Bentov, L. Breidenbach, P. Daian, A. Juels, Y. Li, and X. Zhao. "The Cost of Decentralization in 0x and EtherDelta". `http://hackingdistributed.com/2017/08/13/cost-of-decent/`. Accessed 2019-01-03. 2017.

[BBHR19] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev. "Scalable Zero Knowledge with No Trusted Setup". In: *Proceedings of the 39th Annual International Cryptology Conference*. CRYPTO '19. 2019, pp. 733–764.

[BCC88] G. Brassard, D. Chaum, and C. Crépeau. "Minimum disclosure proofs of knowledge". In: *Journal of Computer and System Sciences* 37.2 (1988), pp. 156–189.

[BCCGP16] J. Bootle, A. Cerulli, P. Chaidos, J. Groth, and C. Petit. "Efficient Zero-Knowledge Arguments for Arithmetic Circuits in the Discrete Log Setting". In: *Proceedings of the 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques*. EUROCRYPT '16. 2016, pp. 327–357.

[BCCT13] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer. "Recursive Composition and Bootstrapping for SNARKs and Proof-Carrying Data". In: *Proceedings of the 45th ACM Symposium on the Theory of Computing*. STOC '13. 2013, pp. 111–120.

[BCGGRS19] E. Ben-Sasson, A. Chiesa, L. Goldberg, T. Gur, M. Riabzev, and N. Spooner. "Linear-Size Constant-Query IOPs for Delegating Computation". In: *Proceedings of the 17th Theory of Cryptography Conference*. TCC '19. 2019.

[BCGMMW20] S. Bowe, A. Chiesa, M. Green, I. Miers, P. Mishra, and H. Wu. "ZEXE: Enabling Decentralized Private Computation". In: *Proceedings of the 41st IEEE Symposium on Security and Privacy*. S&P '20. 2020, pp. 947–964.

[BCGRS17] E. Ben-Sasson, A. Chiesa, A. Gabizon, M. Riabzev, and N. Spooner. "Interactive Oracle Proofs with Constant Rate and Query Complexity". In: *Proceedings of the 44th International Colloquium on Automata, Languages and Programming*. ICALP '17. 2017, 40:1–40:15.

[BCGTV13] E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer, and M. Virza. "SNARKs for C: verifying program executions succinctly and in zero knowledge". In: *Proceedings of the 33rd Annual International Cryptology Conference*. CRYPTO '13. 2013, pp. 90–108.

[BCGTV15] E. Ben-Sasson, A. Chiesa, M. Green, E. Tromer, and M. Virza. "Secure Sampling of Public Parameters for Succinct Zero Knowledge Proofs". In: *Proceedings of the 36th IEEE Symposium on Security and Privacy*. S&P '15. 2015, pp. 287–304.

[BCGV16] E. Ben-Sasson, A. Chiesa, A. Gabizon, and M. Virza. "Quasilinear-Size Zero Knowledge from Linear-Algebraic PCPs". In: *Proceedings of the 13th Theory of Cryptography Conference*. TCC '16-A. 2016, pp. 33–64.

[BCIOP13] N. Bitansky, A. Chiesa, Y. Ishai, R. Ostrovsky, and O. Paneth. "Succinct Non-Interactive Arguments via Linear Interactive Proofs". In: *Proceedings of the 10th Theory of Cryptography Conference*. TCC '13. 2013, pp. 315–333.

[BCLMS21] B. Bünz, A. Chiesa, W. Lin, P. Mishra, and N. Spooner. "Proof-carrying Data without Succinct Arguments". In: *Proceedings of the 41st Annual International Cryptology Conference*. CRYPTO '21. 2021.

[BCMS20] B. Bünz, A. Chiesa, P. Mishra, and N. Spooner. "Proof-Carrying Data from Accumulation Schemes". In: *Proceedings of the 18th Theory of Cryptography Conference*. TCC '20. 2020.

[BCPR16] N. Bitansky, R. Canetti, O. Paneth, and A. Rosen. "On the Existence of Extractable One-Way Functions". In: *SIAM Journal on Computing* 45.5 (2016), pp. 1910–1952.

[BCRSVW19] E. Ben-Sasson, A. Chiesa, M. Riabzev, N. Spooner, M. Virza, and N. P. Ward. "Aurora: Transparent Succinct Arguments for R1CS". In: *Proceedings of the 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques*. EUROCRYPT '19. 2019, pp. 103–128.

[BCS16] E. Ben-Sasson, A. Chiesa, and N. Spooner. "Interactive Oracle Proofs". In: *Proceedings of the 14th Theory of Cryptography Conference*. TCC '16-B. 2016, pp. 31–60.

[BCTV14] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza. "Succinct Non-Interactive Zero Knowledge for a von Neumann Architecture". In: *Proceedings of the 23rd USENIX Security Symposium*. USENIX Security '14. 2014, pp. 781–796.

[BCTV17] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza. "Scalable Zero Knowledge Via Cycles of Elliptic Curves". In: *Algorithmica* 79.4 (2017), pp. 1102–1160.

[BDFG21] D. Boneh, J. Drake, B. Fisch, and A. Gabizon. "Halo Infinite: Recursive zk-SNARKs from any Additive Polynomial Commitment Scheme". In: *Proceedings of the 41st Annual International Cryptology Conference*. CRYPTO '21. 2021.

[BDJT17] L. Breidenbach, P. Daian, A. Juels, and F. Tramèr. "To Sink Frontrunners, Send in the Submarines". `http://hackingdistributed.com/2017/08/28/submarine-sends/`. Accessed 2019-01-03. 2017.

[Ben+14] E. Ben-Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza. "Zerocash: Decentralized Anonymous Payments from Bitcoin". In: *Proceedings of the 35th IEEE Symposium on Security and Privacy*. S&P '14. 2014, pp. 459–474.

[Ben+17] E. Ben-Sasson, I. Bentov, A. Chiesa, A. Gabizon, D. Genkin, M. Hamilis, E. Pergament, M. Riabzev, M. Silberstein, E. Tromer, and M. Virza. "Computational integrity with a public random string from quasi-linear PCPs". In: *Proceedings of the 36th Annual International Conference on Theory and Application of Cryptographic Techniques*. EUROCRYPT '17. 2017, pp. 551–579.

[BFLS91] L. Babai, L. Fortnow, L. A. Levin, and M. Szegedy. "Checking computations in polylogarithmic time". In: *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing*. STOC '91. 1991, pp. 21–32.

[BG04] D. R. L. Brown and R. P. Gallant. "The Static Diffie–Hellman Problem". Cryptology ePrint Archive, Report 2004/306. 2004.

[BG12] S. Bayer and J. Groth. "Efficient Zero-Knowledge Argument for Correctness of a Shuffle". In: *Proceedings of the 31st Annual International Conference on Theory and Applications of Cryptographic Techniques*. EUROCRYPT '12. 2012, pp. 263–280.

[BGG17] S. Bowe, A. Gabizon, and M. Green. "A multi-party protocol for constructing the public parameters of the Pinocchio zk-SNARK". Cryptology ePrint Archive, Report 2017/602. 2017.

[BGG18] S. Bowe, A. Gabizon, and M. D. Green. "A multi-party protocol for constructing the public parameters of the Pinocchio zk-SNARK". 2018.

[BGH19] S. Bowe, J. Grigg, and D. Hopwood. "Halo: Recursive Proof Composition without a Trusted Setup". Cryptology ePrint Archive, Report 2019/1021. 2019.

[BGM17] S. Bowe, A. Gabizon, and I. Miers. "Scalable Multi-party Computation for zk-SNARK Parameters in the Random Beacon Model". Cryptology ePrint Archive, Report 2017/1050. 2017.

[Binance] "Binance". `https://www.binance.com/`. Accessed 2019-01-03.

[BISW17] D. Boneh, Y. Ishai, A. Sahai, and D. J. Wu. "Lattice-Based SNARGs and Their Application to More Efficient Obfuscation". In: *Proceedings of the 36th Annual International Conference on Theory and Applications of Cryptographic Techniques*. EUROCRYPT '17. 2017, pp. 247–277.

[BISW18] D. Boneh, Y. Ishai, A. Sahai, and D. J. Wu. "Quasi-Optimal SNARGs via Linear Multi-Prover Interactive Proofs". In: *Proceedings of the 37th Annual International Conference on Theory and Application of Cryptographic Techniques*. EUROCRYPT '18. 2018, pp. 222–255.

[Bit15] Bitcoin. "Some miners generating invalid blocks". `https://bitcoin.org/en/alert/2015-07-04-spv-mining`. 2015.

[BKM17] I. Bentov, R. Kumaresan, and A. Miller. "Instantaneous Decentralized Poker". In: *Proceedings of the 23rd Annual International Conference on the Theory and Application of Cryptology and Information Security*. ASIACRYPT '17. 2017, pp. 410–440.

[BLS02] P. Barreto, B. Lynn, and M. Scott. "Constructing Elliptic Curves with Prescribed Embedding Degrees". In: *Proceedings of the 3rd International Conference on Security in Communication Networks*. SCN '02. 2002, pp. 257–267.

[BLS04] D. Boneh, B. Lynn, and H. Shacham. "Short Signatures from the Weil Pairing". In: *Journal of Cryptology* 17.4 (2004), pp. 297–319.

[BMMTV21] B. Bünz, M. Maller, P. Mishra, N. Tyagi, and P. Vesely. "Proofs for Inner Pairing Products and Applications". In: *Proceedings of the 27th International Conference on the Theory and Application of Cryptology and Information Security*. ASIACRYPT '21. 2021.

[BMRS20] J. Bonneau, I. Meckler, V. Rao, and E. Shapiro. "Coda: Decentralized Cryptocurrency at Scale". Cryptology ePrint Archive, Report 2020/352. 2020.

[BN06] M. Bellare and G. Neven. "Multi-signatures in the plain public-Key model and a general forking lemma". In: *Proceedings of the 13th ACM Conference on Computer and Communications Security*. CCS '06. 2006, pp. 390–399.

[Bow17a] S. Bowe. "Bellman". 2017. URL: `https://github.com/zkcrypto/bellman`.

[Bow17b] S. Bowe. "Pairing". 2017. URL: `https://github.com/zkcrypto/pairing`.

[BP15] E. Boyle and R. Pass. "Limits of Extractability Assumptions with Distributional Auxiliary Input". In: *Proceedings of the 21st International Conference on the Theory and Application of Cryptology and Information Security*. ASIACRYPT '15. 2015, pp. 236–261.

[BR93] M. Bellare and P. Rogaway. "Random Oracles Are Practical: A Paradigm for Designing Efficient Protocols". In: *Proceedings of the 1st ACM Conference on Computer and Communications Security*. CCS '93. 1993, pp. 62–73.

[BS08] E. Ben-Sasson and M. Sudan. "Short PCPs with Polylog Query Complexity". In: *SIAM Journal on Computing* 38.2 (2008), pp. 551–607.

[Can01] R. Canetti. "Universally Composable Security: A New Paradigm for Cryptographic Protocols". In: *Proceedings of the 42nd Annual Symposium on Foundations of Computer Science*. FOCS '01. 2001, pp. 136–145.

[CCDW20] W. Chen, A. Chiesa, E. Dauterman, and N. P. Ward. "Reducing Participation Costs via Incremental Verification for Ledger Systems". Cryptology ePrint Archive, Report 2020/1522. 2020.

[CCW19] A. Chiesa, L. Chua, and M. Weidner. "On Cycles of Pairing-Friendly Elliptic Curves". In: *SIAM Journal on Applied Algebra and Geometry* 3.2 (2019), pp. 175–192.

[CD98] R. Cramer and I. Damgård. "Zero-Knowledge Proofs for Finite Field Arithmetic; or: Can Zero-Knowledge be for Free?" In: *Proceedings of the 18th Annual International Cryptology Conference*. CRYPTO '98. 1998, pp. 424–441.

[Celo] "Celo Cryptocurrency". `https://celo.org/`. 2018.

[CFFQR20] M. Campanelli, A. Faonio, D. Fiore, A. Querol, and H. Rodríguez. "Lunar: a Toolbox for More Efficient Universal and Updatable zkSNARKs and Commit-and-Prove Extensions". Cryptology ePrint Archive, Report 2020/1069. 2020.

[CFQ19] M. Campanelli, D. Fiore, and A. Querol. "LegoSNARK: Modular Design and Composition of Succinct Zero-Knowledge Proofs". In: *Proceedings of the 26th ACM Conference on Computer and Communications Security*. CCS '19. 2019, pp. 2075–2092.

[CFS17] A. Chiesa, M. A. Forbes, and N. Spooner. "A Zero Knowledge Sumcheck and its Applications". Cryptology ePrint Archive, Report 2017/305. 2017.

[CGLMMM17] A. Chiesa, M. Green, J. Liu, P. Miao, I. Miers, and P. Mishra. "Decentralized Anonymous Micropayments". In: *Proceedings of the 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques*. EUROCRYPT '17. 2017, pp. 609–642.

[Cha14] Chainalysis. "Chainalysis Inc". `https://chainalysis.com/`. 2014.

[Che10] J. H. Cheon. "Discrete Logarithm Problems with Auxiliary Inputs". In: *Journal of Cryptology* 23.3 (2010), pp. 457–476.

[CHMMVW20] A. Chiesa, Y. Hu, M. Maller, P. Mishra, N. Vesely, and N. Ward. "Marlin: Preprocessing zkSNARKs with Universal and Updatable SRS". In: *Proceedings of the 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques*. EUROCRYPT '20. 2020, pp. 738–768.

[Cim18] C. Cimpanu. "Zaif cryptocurrency exchange loses $60 million in recent hack". `https://www.zdnet.com/article/zaif-cryptocurrency-exchange-loses-60-million-in-july-hack/`. Accessed 2018-12-27. 2018.

[CLN11] C. Costello, K. E. Lauter, and M. Naehrig. "Attractive Subfamilies of BLS Curves for Implementing High-Security Pairings". In: *Proceedings of the 12th International Conference on Cryptology in India*. INDOCRYPT '11. 2011, pp. 320–342.

[Coinbase] "Coinbase". `https://www.coinbase.com/`. Accessed 2019-01-03.

[con] arkworks contributors. "`arkworks` zkSNARK ecosystem". `https://arkworks.rs`.

[Cos12] C. Costello. "Particularly Friendly Members of Family Trees". Cryptology ePrint Archive, Report 2012/072. 2012.

[COS20] A. Chiesa, D. Ojha, and N. Spooner. "Fractal: Post-Quantum and Transparent Recursive Proofs from Holography". In: *Proceedings of the 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques*. EUROCRYPT '20. 2020, pp. 769–793.

[CRR11] R. Canetti, B. Riva, and G. N. Rothblum. "Practical delegation of computation using multiple servers". In: *Proceedings of the 19th ACM Conference on Computer and Communications Security*. CCS '11. 2011, pp. 445–454.

[CRR13] R. Canetti, B. Riva, and G. N. Rothblum. "Refereed delegation of computation". In: *Information and Computation* 226 (2013), pp. 16–36.

[CT10] A. Chiesa and E. Tromer. "Proof-Carrying Data and Hearsay Arguments from Signature Cards". In: *Proceedings of the 1st Symposium on Innovations in Computer Science*. ICS '10. 2010, pp. 310–331.

[CTV13] S. Chong, E. Tromer, and J. A. Vaughan. "Enforcing Language Semantics Using Proof-Carrying Data". Cryptology ePrint Archive, Report 2013/513. 2013.

[CTV15] A. Chiesa, E. Tromer, and M. Virza. "Cluster Computing in Zero Knowledge". In: *Proceedings of the 34th Annual International Conference on Theory and Application of Cryptographic Techniques*. EUROCRYPT '15. 2015, pp. 371–403.

[CXZ21] S. Chu, Y. Xia, and Z. Zhang. “Manta: a Plug and Play Private DeFi Stack”. Cryptology ePrint Archive, Report 2021/743. 2021.

[CZJKJS17] E. Cecchetti, F. Zhang, Y. Ji, A. E. Kosba, A. Juels, and E. Shi. “Solidus: Confidential Distributed Ledger Transactions via PVORM”. In: *Proceedings of the 24th ACM Conference on Computer and Communications Security*. CCS ’17. 2017, pp. 701–717.

[Dai+20] P. Daian, S. Goldfeder, T. Kell, Y. Li, X. Zhao, I. Bentov, L. Breidenbach, and A. Juels. “Flash Boys 2.0: Frontrunning, Transaction Reordering, and Consensus Instability in Decentralized Exchanges”. In: *Proceedings of the 41st IEEE Symposium on Security and Privacy*. S&P ’20. 2020, pp. 910–927.

[DDOPS01] A. De Santis, G. Di Crescenzo, R. Ostrovsky, G. Persiano, and A. Sahai. “Robust Non-interactive Zero Knowledge”. In: *Proceedings of the 21st Annual International Cryptology Conference*. CRYPTO ’01. 2001, pp. 566–598.

[De18] N. De. “Coincheck Confirms Crypto Hack Loss Larger than Mt. Gox”. `https://www.coindesk.com/coincheck-confirms-crypto-hack-loss-larger-than-mt-gox`. Accessed 2018-12-27. 2018.

[Den02] A. W. Dent. “Adapting the Weaknesses of the Random Oracle Model to the Generic Group Model”. In: *Proceedings of the 8th International Conference on the Theory and Application of Cryptology and Information Security*. ASIACRYPT ’02. 2002, pp. 100–109.

[DF91] Y. Desmedt and Y. Frankel. “Shared Generation of Authenticators and Signatures (Extended Abstract)”. In: *Proceedings of the 11th Annual International Cryptology Conference*. CRYPTO ’91. 1991, pp. 457–469.

[DFGK14] G. Danezis, C. Fournet, J. Groth, and M. Kohlweiss. “Square Span Programs with Applications to Succinct NIZK Arguments”. In: *Proceedings of the 20th International Conference on the Theory and Application of Cryptology and Information Security*. ASIACRYPT ’14. 2014, pp. 532–550.

[Dod07] Y. Dodis. “Lecture notes: Exposure-Resilient Cryptography”. `https://www.cs.nyu.edu/courses/spring07/G22.3033-013/`. 2007.

[DSCOZ15] T. T. A. Dinh, P. Saxena, E. Chang, B. C. Ooi, and C. Zhang. “M2R: Enabling Stronger Privacy in MapReduce Computation”. In: *Proceedings of the 24th USENIX Security Symposium*. USENIX Security ’15. 2015, pp. 447–462.

[EG20] Y. El Housni and A. Guillevic. “Optimized and Secure Pairing-Friendly Elliptic Curves Suitable for One Layer Proof Composition”. In: *Proceedings of the 19th International Conference on Cryptology and Network Security*. CANS ’20. 2020.

[Ell13] Elliptic. “Elliptic Enterprises Limited”. `https://www.elliptic.co/`. 2013.

[EMC19] S. Eskandari, S. Moosavi, and J. Clark. “SoK: Transparent Dishonesty: front-running attacks on Blockchain”. arXiv cs.CR/1902.05164. 2019.

[ET18] J. Eberhardt and S. Tai. “ZoKrates — Scalable Privacy-Preserving Off-Chain Computations”. In: iThings/GreenCom/CPSCom/SmartData. 2018, pp. 1084–1091.

[Eth16] Ethereum. "I thikn the attacker is this miner - today he made over $50k". https://www.reddit.com/r/ethereum/comments/55xh2w/i_thikn_the_attacker_is_this_miner_today_he_made/. 2016.

[Eth18] Etherscan. "The Ethereum Block Explorer". https://etherscan.io/tokens. 2018.

[Eth21] Ethereum. "Layer 2 Rollups". https://ethereum.org/en/developers/docs/scaling/layer-2-rollups/#zk-rollups. 2021.

[FK97] U. Feige and J. Kilian. "Making Games Short". In: *Proceedings of the 29th ACM Symposium on the Theory of Computing*. STOC '97. 1997, pp. 506–516.

[FKL18] G. Fuchsbauer, E. Kiltz, and J. Loss. "The Algebraic Group Model and its Applications". In: *Proceedings of the 38th Annual International Cryptology Conference*. CRYPTO '18. 2018, pp. 33–62.

[FKMSSS16] N. Fleischhacker, J. Krupp, G. Malavolta, J. Schneider, D. Schröder, and M. Simkin. "Efficient Unlinkable Sanitizable Signatures from Signatures with Re-randomizable Keys". In: *Proceedings of the 19th International Conference on Practice and Theory in Public-Key Cryptography*. PKC '16. 2016, pp. 301–330.

[FQZDC21] B. Feng, L. Qin, Z. Zhang, Y. Ding, and S. Chu. "ZEN: An Optimizing Compiler for Verifiable, Zero-Knowledge Neural Network Inferences". Cryptology ePrint Archive, Report 2021/087. 2021.

[FS86] A. Fiat and A. Shamir. "How to prove yourself: practical solutions to identification and signature problems". In: *Proceedings of the 6th Annual International Cryptology Conference*. CRYPTO '86. 1986, pp. 186–194.

[FST10] D. Freeman, M. Scott, and E. Teske. "A taxonomy of pairing-friendly elliptic curves". In: *Journal of Cryptology* 23.2 (2010), pp. 224–280.

[Gab+20] A. Gabizon, K. Gurkan, P. Jovanovic, G. Konstantopoulos, A. Oines, M. Olszewski, M. Straka, E. Tromer, and P. Vesely. "Plumo: Towards Scalable Interoperable Blockchains Using Ultra Light Validation Systems". https://celo.org/papers/plumo. 2020.

[Gab19] A. Gabizon. "Improved prover efficiency and SRS size in a Sonic-like system". Cryptology ePrint Archive, Report 2019/601. 2019.

[Gemini] "Gemini". https://gemini.com/dollar. Accessed 2019-01-24.

[GGPR13] R. Gennaro, C. Gentry, B. Parno, and M. Raykova. "Quadratic Span Programs and Succinct NIZKs without PCPs". In: *Proceedings of the 32nd Annual International Conference on Theory and Application of Cryptographic Techniques*. EUROCRYPT '13. 2013, pp. 626–645.

[GJMMST21] K. Gurkan, P. Jovanovic, M. Maller, S. Meiklejohn, G. Stern, and A. Tomescu. "Aggregatable Distributed Key Generation". Cryptology ePrint Archive, Report 2021/005. 2021.

[GKMMM18] J. Groth, M. Kohlweiss, M. Maller, S. Meiklejohn, and I. Miers. "Updatable and Universal Common Reference Strings with Applications to zk-SNARKs". In: *Proceedings of the 38th Annual International Cryptology Conference*. CRYPTO '18. 2018, pp. 698–728.

[GKR15] S. Goldwasser, Y. T. Kalai, and G. N. Rothblum. "Delegating Computation: Interactive Proofs for Muggles". In: *Journal of the ACM* 62.4 (2015), 27:1–27:64.

[GM17] J. Groth and M. Maller. "Snarky Signatures: Minimal Signatures of Knowledge from Simulation-Extractable SNARKs". In: *Proceedings of the 37th Conference on the Theory and Applications of Cryptographic Techniques*. CRYPTO '17. 2017, pp. 581–612.

[GMR89] S. Goldwasser, S. Micali, and C. Rackoff. "The knowledge complexity of interactive proof systems". In: *SIAM Journal on Computing* 18.1 (1989), pp. 186–208.

[GO94] O. Goldreich and Y. Oren. "Definitions and properties of zero-knowledge proof systems". In: *Journal of Cryptology* 7.1 (1994), pp. 1–32.

[Goo14] L. Goodman. "Tezos — a self-amending crypto-ledger". `https://tezos.com/static/white_paper-2dc8c02267a8fb86bd67a108199441bf.pdf`. 2014.

[Gro06] J. Groth. "Simulation-Sound NIZK Proofs for a Practical Language and Constant Size Group Signatures". In: *Proceedings of the 12th International Conference on the Theory and Application of Cryptology and Information Security*. ASIACRYPT '06. 2006, pp. 444–459.

[Gro10] J. Groth. "Short Pairing-Based Non-interactive Zero-Knowledge Arguments". In: *Proceedings of the 16th International Conference on the Theory and Application of Cryptology and Information Security*. ASIACRYPT '10. 2010, pp. 321–340.

[Gro16] J. Groth. "On the Size of Pairing-Based Non-interactive Arguments". In: *Proceedings of the 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques*. EUROCRYPT '16. 2016, pp. 305–326.

[GT21] A. Ghoshal and S. Tessaro. "Tight State-Restoration Soundness in the Algebraic Group Model". In: *Proceedings of the 41st Annual Conference on Advances in Cryptology*. Vol. 12827. CRYPTO '21. 2021, pp. 64–93.

[GWC19] A. Gabizon, Z. J. Williamson, and O. Ciobotaru. "PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge". Cryptology ePrint Archive, Report 2019/953. 2019.

[Halo20] S. Bowe, J. Grigg, and D. Hopwood. *Halo2*. 2020. URL: `https://github.com/zcash/halo2`.

[Har18] C. Harper. "In Search of Stability: An Overview of the Budding Stablecoin Ecosystem". `https://bitcoinmagazine.com/articles/search-stability-overview-budding-stablecoin-ecosystem/`. Accessed 2019-1-24. 2018.

[HBHW20] D. Hopwood, S. Bowe, T. Hornby, and N. Wilcox. "Zcash Protocol Specification". 2020. URL: `https://github.com/zcash/zips/blob/master/protocol/protocol.pdf`.

[HGD21] U. Haböck, A. Garoffolo, and D. Di Benedetto. "Darlin: A proof carrying data scheme based on Marlin". Cryptology ePrint Archive, Report 2021/930. 2021.

[Hop20] D. Hopwood. "The Pasta Curves for Halo 2 and Beyond". `http://electriccoin.co/blog/the-pasta-curves-for-halo-2-and-beyond/`. 2020.

[Horizen] "Horizen Protocol". https://www.horizen.io/.

[IKOS06] Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai. "Cryptography from Anonymity". In: *Proceedings of the 47th Annual Symposium on Foundations of Computer Science*. FOCS '06. 2006, pp. 239–248.

[Ili+21] M. Iliadi, F. Mergoupis-Anagnou, S. Heiberg, B. Schoenmakers, M. Kohlweiss, D. Friolo, I. Visconti, and P. Louridas. "Report on Tools for Privacy-Preserving Applications on Ledgers". https://media.voog.com/0000/0042/1115/files/D4.5%20-%20Report%20on%20Tools%20for%20Privacy-Preserving%20Applications%20on%20Ledgers.pdf. 2021.

[JSST16] S. Jain, P. Saxena, F. Stephan, and J. Teutsch. "How to verify computation with a rational network". arXiv cs.GT/1606.05917. 2016.

[Kal+20] H. Kalodner, M. Möser, K. Lee, S. Goldfeder, M. Plattner, A. Chator, and A. Narayanan. "BlockSci: Design and applications of a blockchain analysis platform". In: *Proceedings of the 29th USENIX Security Symposium*. USENIX Security '20. 2020, pp. 2721–2738.

[KB16] R. Kumaresan and I. Bentov. "Amortizing Secure Computation with Penalties". In: *Proceedings of the 23rd ACM Conference on Computer and Communications Security*. CCS '16. 2016, pp. 418–429.

[KB20] A. Kattis and J. Bonneau. "Proof of Necessary Work: Succinct State Verification with Fairness Guarantees". Cryptology ePrint Archive, Report 2020/190. 2020.

[KGCWF18] H. A. Kalodner, S. Goldfeder, X. Chen, S. M. Weinberg, and E. W. Felten. "Arbitrum: Scalable, private smart contracts". In: *Proceedings of the 27th USENIX Security Symposium*. USENIX Security '18. 2018, pp. 1353–1370.

[KGM19] G. Kaptchuk, M. Green, and I. Miers. "Giving State to the Stateless: Augmenting Trustworthy Computation with Ledgers". In: *Proceedings of the 26th Annual Network and Distributed System Security Symposium*. NDSS '19. 2019.

[Kil92] J. Kilian. "A note on efficient zero-knowledge proofs and arguments". In: *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*. STOC '92. 1992, pp. 723–732.

[KKM07] S. Kozaki, T. Kutsuma, and K. Matsuo. "Remarks on Cheon's Algorithms for Pairing-Related Problems". In: *Proceedings of the 1st International Conference on Pairing-Based Cryptography*. Pairing '07. 2007, pp. 302–316.

[KMB15] R. Kumaresan, T. Moran, and I. Bentov. "How to Use Bitcoin to Play Decentralized Poker". In: *Proceedings of the 22nd ACM Conference on Computer and Communications Security*. CCS '15. 2015, pp. 195–206.

[KMSV21] M. Kohlweiss, M. Maller, J. Siim, and M. Volkhov. "Snarky Ceremonies". Cryptology ePrint Archive, Report 2021/219. 2021.

[KMSWP16] A. E. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou. "Hawk: The Blockchain Model of Cryptography and Privacy-Preserving Smart Contracts". In: *Proceedings of the 37th IEEE Symposium on Security and Privacy*. S&P '16. 2016, pp. 839–858.

[KZG10] A. Kate, G. M. Zaverucha, and I. Goldberg. "Constant-Size Commitments to Polynomials and Their Applications". In: *Proceedings of the 16th International Conference on the Theory and Application of Cryptology and Information Security*. ASIACRYPT '10. 2010, pp. 177–194.

[KZGM21] S. Kanjalkar, Y. Zhang, S. Gandlur, and A. Miller. "Publicly Auditable MPC-as-a-Service with succinct verification and universal setup". In: *Proceedings of the 5th IEEE Workshop on Security and Privacy on the Blockchain*. S&B '21. 2021.

[Lin03] Y. Lindell. "Parallel Coin-Tossing and Constant-Round Secure Two-Party Computation". In: *Journal of Cryptology* 16.3 (2003), pp. 143–184.

[Lip12] H. Lipmaa. "Progression-Free Sets and Sublinear Pairing-Based Non-Interactive Zero-Knowledge Arguments". In: *Proceedings of the 9th Theory of Cryptography Conference on Theory of Cryptography*. TCC '12. 2012, pp. 169–189.

[LTKS15] L. Luu, J. Teutsch, R. Kulkarni, and P. Saxena. "Demystifying Incentives in the Consensus Computer". In: *Proceedings of the 22nd ACM Conference on Computer and Communications Security*. CCS '15. 2015, pp. 706–719.

[Manta] "Manta Network". `https://manta.network`.

[Mau05] U. M. Maurer. "Abstract Models of Computation in Cryptography". In: *Proceedings of the 10th IMA International Conference on Cryptography and Coding*. IMA '05. 2005, pp. 1–12.

[MBKM19] M. Maller, S. Bowe, M. Kohlweiss, and S. Meiklejohn. "Sonic: Zero-Knowledge SNARKs from Linear-Size Universal and Updateable Structured Reference Strings". In: *Proceedings of the 26th ACM Conference on Computer and Communications Security*. CCS '19. 2019.

[McK+13] F. McKeen, I. Alexandrovich, A. Berenzon, C. V. Rozas, H. Shafi, V. Shanbhogue, and U. R. Savagaonkar. "Innovative instructions and software model for isolated execution". In: *Proceedings of the Second Workshop on Hardware and Architectural Support for Security and Privacy*. HASP '13. 2013, p. 10.

[Mei+13] S. Meiklejohn, M. Pomarole, G. Jordan, K. Levchenko, D. McCoy, G. M. Voelker, and S. Savage. "A Fistful of Bitcoins: characterizing payments among men with no names". In: *Proceedings of the 2013 Internet Measurement Conference*. IMC '13. 2013, pp. 127–140.

[Mer87] R. C. Merkle. "A Digital Signature Based on a Conventional Encryption Function". In: *Proceedings of the 7th Conference on the Theory and Applications of Cryptographic Techniques*. CRYPTO '87. 1987, pp. 369–378.

[MGGR13] I. Miers, C. Garman, M. Green, and A. D. Rubin. "Zerocoin: Anonymous Distributed E-Cash from Bitcoin". In: *Proceedings of the 34th IEEE Symposium on Security and Privacy*. S&P '13. 2013, pp. 397–411.

[Mic00] S. Micali. "Computationally Sound Proofs". In: *SIAM Journal on Computing* 30.4 (2000), pp. 1253–1298.

[Mina] O. Labs. "Mina Cryptocurrency". `https://minaprotocol.com/`. 2017.

[Mir] "Mir Protocol". https://mirprotocol.org.

[MRK03] S. Micali, M. O. Rabin, and J. Kilian. "Zero-Knowledge Sets". In: *Proceedings of the 44th Annual Symposium on Foundations of Computer Science*. FOCS '03. 2003, pp. 80–91.

[MS18] I. Meckler and E. Shapiro. "Coda: Decentralized cryptocurrency at scale". https://cdn.codaprotocol.com/v2/static/coda-whitepaper-05-10-2018-0.pdf. 2018.

[Nak09] S. Nakamoto. "Bitcoin: a peer-to-peer electronic cash system". 2009. URL: http://www.bitcoin.org/bitcoin.pdf.

[NKDM03] A. Nicolosi, M. Krohn, Y. Dodis, and D. Mazières. "Proactive Two-Party Signatures for User Authentication". In: *Proceedings of the Network and Distributed System Security Symposium*. NDSS '03. 2003.

[Noir] "The Noir Programming Language". https://github.com/noir-lang/noir.

[NT16] A. Naveh and E. Tromer. "PhotoProof: Cryptographic Image Authentication for Any Set of Permissible Transformations". In: *Proceedings of the 37th IEEE Symposium on Security and Privacy*. S&P '16. 2016, pp. 255–271.

[NVV18] N. Narula, W. Vasquez, and M. Virza. "zkLedger: Privacy-Preserving Auditing for Distributed Ledgers". In: *Proceedings of the 15th USENIX Symposium on Networked Systems Design and Implementation*. NSDI '18. 2018, pp. 65–80.

[PA14] N. Popper and R. Abrams. "Apparent Theft at Mt. Gox Shakes Bitcoin World". https://www.nytimes.com/2014/02/25/business/apparent-theft-at-mt-gox-shakes-bitcoin-world.html. Accessed 2018-12-27. 2014.

[Paradex] "Paradex". https://paradex.io/. Accessed 2018-12-27.

[Pas03] R. Pass. "On Deniability in the Common Reference String and Random Oracle Model". In: *Proceedings of the 23rd Annual International Cryptology Conference*. CRYPTO '03. 2003, pp. 316–337.

[Paxos] "Paxos Standard". https://paxos.com/standard. Accessed 2019-01-24.

[PB17] J. Poon and V. Buterin. "Plasma: Scalable Autonomous Smart Contracts". https://plasma.io/. 2017.

[Pickles20] O(1) Labs. *Pickles*. URL: https://github.com/o1-labs/marlin.

[Pro18] T. B. Project. "An Overview of Decentralized Trading of Digital Assets". https://collaborate.thebkp.com/project/TL/document/9/version/10/. Accessed 2018-12-27. 2018.

[PS00] D. Pointcheval and J. Stern. "Security Arguments for Digital Signatures and Blind Signatures". In: *Journal of Cryptology* 13.3 (2000), pp. 361–396.

[PS18] C. Peikert and S. Shiehian. "Privately Constraining and Programming PRFs, the LWE Way". In: *Proceedings of the 21st International Conference on Practice and Theory in Public-Key Cryptography*. PKC '18. 2018, pp. 675–701.

[PST13] C. Papamanthou, E. Shi, and R. Tamassia. “Signatures of Correct Computation”. In: *Proceedings of the 10th Theory of Cryptography Conference*. TCC ’13. 2013, pp. 222–242.

[PV05] P. Paillier and D. Vergnaud. “Discrete-Log-Based Signatures May Not Be Equivalent to Discrete Log”. In: *Proceedings of the 11th International Conference on the Theory and Application of Cryptology and Information Security*. ASIACRYPT ’05. 2005, pp. 1–20.

[Radar] “Radar Relay”. `https://radarrelay.com/`. Accessed 2018-12-27.

[Rei16] C. Reiwießner. “From Smart Contracts to Courts with not so Smart Judges”. `https://blog.ethereum.org/2016/02/17/smart-contracts-courts-not-smart-judges/`. 2016.

[RGJKM17] A. Rai Choudhuri, M. Green, A. Jain, G. Kaptchuk, and I. Miers. “Fairness in an Unfair World: Fair Multiparty Computation from Public Bulletin Boards”. In: *Proceedings of the 24th ACM Conference on Computer and Communications Security*. CCS ’17. 2017, pp. 719–728.

[RH11] F. Reid and M. Harrigan. “An Analysis of Anonymity in the Bitcoin System”. In: *Proceedings of the 3rd IEEE International Conference on Privacy, Security, Risk and Trust (PASSAT), and the 3rd IEEE International Conference on Social Computing (SocialCom)*. SocialCom/PASSAT ’11. 2011, pp. 1318–1326.

[RRR16] O. Reingold, R. Rothblum, and G. Rothblum. “Constant-Round Interactive Proofs for Delegating Computation”. In: *Proceedings of the 48th ACM Symposium on the Theory of Computing*. STOC ’16. 2016, pp. 49–62.

[RS13] D. Ron and A. Shamir. “Quantitative Analysis of the Full Bitcoin Transaction Graph”. In: *Proceedings of the 17th International Conference on Financial Cryptography and Data Security*. FC ’13. 2013, pp. 6–24.

[RZ21] C. Ràfols and A. Zapico. “An Algebraic Framework for Universal and Updatable SNARKs”. In: *Proceedings of the 41st Annual International Cryptology Conference*. CRYPTO ’21. 2021.

[Sah99] A. Sahai. “Non-Malleable Non-Interactive Zero Knowledge and Adaptive Chosen-Ciphertext Security”. In: *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*. FOCS ’99. 1999, pp. 543–553.

[Sch+15] F. Schuster, M. Costa, C. Fournet, C. Gkantsidis, M. Peinado, G. Mainar-Ruiz, and M. Russinovich. “VC3: Trustworthy Data Analytics in the Cloud Using SGX”. In: *Proceedings of the 36th IEEE Symposium on Security and Privacy*. S&P ’15. 2015, pp. 38–54.

[Sch91] C. Schnorr. “Efficient Signature Generation by Smart Cards”. In: *Journal of Cryptology* 4.3 (1991), pp. 161–174.

[Set20] S. Setty. “Spartan: Efficient and General-Purpose zkSNARKs Without Trusted Setup”. In: *Proceedings of the 40th Annual International Cryptology Conference*. CRYPTO ’20. 2020, pp. 704–737.

[Sho97] V. Shoup. "Lower bounds for discrete logarithms and related problems". In: *Proceedings of the 16th International Conference on the Theory and Application of Cryptographic Techniques*. EUROCRYPT '97. 1997, pp. 256–266.

[SJ99] C. Schnorr and M. Jakobsson. "Security Of Discrete Log Cryptosystems in the Random Oracle and the Generic Model". In: *The Mathematics of Public-Key Cryptography*. MPKC '99. 1999.

[SMZ14] M. Spagnuolo, F. Maggi, and S. Zanero. "BitIodine: Extracting Intelligence from the Bitcoin Network". In: *Proceedings of the 18th International Conference on Financial Cryptography and Data Security*. FC '14. 2014, pp. 457–468.

[SS01] D. R. S. Stinson and R. Strobl. "Provably Secure Distributed Schnorr Signatures and a (t, n) Threshold Scheme for Implicit Certificates". In: *Proceedings of the 5th Australasian Conference on Information Security and Privacy*. ACISP '01. 2001, pp. 417–434.

[SZ20] A. Szepieniec and Y. Zhang. "Polynomial IOPs for Linear Algebra Relations". Cryptology ePrint Archive, Report 2020/1022. 2020.

[TFBT21] N. Tyagi, B. Fisch, J. Bonneau, and S. Tessaro. "Client-Auditable Verifiable Registries". Cryptology ePrint Archive, Report 2021/627. 2021.

[TR17] J. Teutsch and C. Reiwießner. "A scalable verification solution for blockchains". `https://people.cs.uchicago.edu/~teutsch/papers/truebit.pdf`. 2017.

[Val08] P. Valiant. "Incrementally Verifiable Computation or Proofs of Knowledge Imply Time/Space Efficiency". In: *Proceedings of the 5th Theory of Cryptography Conference*. TCC '08. 2008, pp. 1–18.

[Van+19] J. Van Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx. "Breaking Virtual Memory Protection and the SGX Ecosystem with Foreshadow". In: *IEEE Micro* 39.3 (2019), pp. 66–74.

[VB15] F. Vogelsteller and V. Buterin. "ERC-20 Token Standard". `https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md`. 2015.

[Whi18] B. Whitehat. "Roll up: Scale Ethereum with SNARKs". `https://github.com/barryWhiteHat/roll_up`. 2018.

[Woo16] G. Wood. "Polkadot: Vision for a Heterogeneous Multi-Chain Framework". `https://polkadot.network/PolkaDotPaper.pdf`. 2016.

[Woo17] G. Wood. "Ethereum: A Secure Decentralised Generalised Transaction Ledger". `http://yellowpaper.io`. 2017.

[WSRBW15] R. Wahby, S. Setty, Z. Ren, A. Blumberg, and M. Walfish. "Efficient RAM and control flow in verifiable outsourced computation". In: *Proceedings of the 22nd Annual Network and Distributed System Security Symposium*. NDSS '15. 2015.

[WTSTW18] R. S. Wahby, I. Tzialla, A. Shelat, J. Thaler, and M. Walfish. "Doubly-Efficient zkSNARKs Without Trusted Setup". In: *Proceedings of the 39th IEEE Symposium on Security and Privacy*. S&P '18. 2018, pp. 926–943.

[XZZPS19] T. Xie, J. Zhang, Y. Zhang, C. Papamanthou, and D. Song. "Libra: Succinct Zero-Knowledge Proofs with Optimal Prover Computation". In: *Proceedings of the 39th Annual International Cryptology Conference*. CRYPTO '19. 2019, pp. 733–764.

[Yak18] A. Yakovenko. "Solana: A new architecture for a high performance blockchain". `https://solana.com/solana-whitepaper.pdf`. 2018.

[Zcash] "ZCash". https://z.cash/. 2015.

[ZcashCmny] "ZCash Parameter Generation". `https://z.cash/technology/paramgen.html`. Accessed: 2017-09-28. 2016.

[ZcashMPC] "The Zcash Ceremony". `https://z.cash/blog/the-design-of-the-ceremony.html`. 2016.

[ZDBPGS17] W. Zheng, A. Dave, J. G. Beekman, R. A. Popa, J. E. Gonzalez, and I. Stoica. "Opaque: An Oblivious and Encrypted Distributed Analytics Platform". In: *Proceedings of the 14th USENIX Symposium on Networked Systems Design and Implementation*. NSDI '17. 2017, pp. 283–298.

[ZGKPP17a] Y. Zhang, D. Genkin, J. Katz, D. Papadopoulos, and C. Papamanthou. "A Zero-Knowledge Version of vSQL". Cryptology ePrint Archive, Report 2017/1146. 2017.

[ZGKPP17b] Y. Zhang, D. Genkin, J. Katz, D. Papadopoulos, and C. Papamanthou. "vSQL: Verifying Arbitrary SQL Queries over Dynamic Outsourced Databases". In: *Proceedings of the 38th IEEE Symposium on Security and Privacy*. S&P '17. 2017, pp. 863–880.

[ZGKPP18] Y. Zhang, D. Genkin, J. Katz, D. Papadopoulos, and C. Papamanthou. "vRAM: Faster Verifiable RAM with Program-Independent Preprocessing". In: *Proceedings of the 39th IEEE Symposium on Security and Privacy*. S&P '18. 2018, pp. 908–925.

[Zha+20] F. Zhang, W. He, R. Cheng, J. Kos, N. Hynes, N. M. Johnson, A. Juels, A. Miller, and D. Song. "The Ekiden Platform for Confidentiality-Preserving, Trustworthy, and Performant Smart Contracts". In: *IEEE Secur. Priv.* 18.3 (2020), pp. 17–27.

[Zha18] W. Zhao. "Bithumb $31 Million Crypto Exchange Hack: What We Know (And Don't)". `https://www.coindesk.com/bithumb-exchanges-31-million-hack-know-dont-know`. Accessed 2018-12-27. 2018.

[ZoKrates] "ZoKrates". `https://github.com/Zokrates/ZoKrates`.

[ZX21] W. Zhang and Y. Xia. "Hydra: Succinct Fully Pipelineable Interactive Arguments of Knowledge". Cryptology ePrint Archive, Report 2021/641. 2021.

[ZZWG21] Y. Zhang, R. Zhang, G. Wang, and D. Gu. "VCProof: Constructing Shorter and Faster-to-Verify zkSNARKs with Vector Oracles". Cryptology ePrint Archive, Report 2021/710. 2021.