Authenticated Outsourced Incrementally Verifiable MPC

Anders Dalskov[1], Anders Kallesøe[2], and Hamidreza Khoshakhlagh [⋆2]

[1] Partisia
anderspkd@fastmail.com
[2] Aarhus University
{ak, hamidreza }@cs.au.dk

Abstract. Many privacy-preserving applications such as iterative machine learning training require executing ongoing computations over a secret state for long periods of time. While MPC enables secure computation, long-lived secret state is vulnerable to adaptive corruptions, requiring periodic share refreshes and committee rotation. In many settings, it is desirable for external parties to be able to verify that these state updates and rotations are performed correctly, but existing approaches for such public verifiability incur high overhead as they require MPC nodes to generate expensive proofs over secret-shared data.
We propose *Authenticated Outsourced Incrementally Verifiable MPC* (IV-MPC), a framework for efficient public verification of long-running MPC computation. Our approach decouples private proof computation (run by MPC nodes) from public tasks within the proof generation that can be delegated to a single external prover $\mathtt{P}^*$. To realize this architecture, we introduce three main components: (1) a Private-opening Polynomial Commitment Scheme (PoPCS) that lets an incoming committee prove they received the correct state shares without revealing the underlying data; (2) an efficient distributed proof system based on the sumcheck protocol and compressed Σ-protocols to verify state updates; and (3) a stateless handover technique that anchors committee transfers to a fixed state commitment, preventing proof sizes from growing over time. Combined with an incrementally verifiable computation (IVC) proof run by the external prover $\mathtt{P}^*$, IV-MPC enables external auditors to verify correctness of MPC computation at every round with verification time asymptotically independent of the total number of epochs.

1 Introduction

Multiparty Computation (MPC) allows a group of parties to run a computation on distributed inputs, without revealing anything except the output. MPC has evolved from a purely theoretical endeavor [Yao86, GMW87, BGW88, CCD88] to something usable in practice. An MPC protocol typically works in phases: Initially, the parties will *secret-share* their respective inputs towards the other

⋆ At the time of this work, the author was also affiliated with Partisia.

protocol participants. Next, an interactive protocol is run on these secret-shares in order to compute the desired function. Finally, the secret-shares representing the function's output is reconstructed so that everybody may learn the output. The MPC threat model assumes an adversary that corrupts at most some $t < n$ of the n parties. Different choices of t (as a fraction of n) leads to different protocols, with different trade-offs. The distributed nature of MPC makes it an ideal technology when two or more parties wish to collaborate, but where data sharing is either prohibitive or not allowed. However, this distributed nature may itself be a limiting factor. Indeed, the main performance bottleneck of MPC is the interaction between parties, and although this interaction can be made very small (see e.g., [ESW24]), it cannot be eliminated. This makes MPC difficult to use *in practice* because its users—the parties—have strict requirements in terms of performance and connectivity. And the more parties that wish to participate, the more difficult an MPC is to pull off in practice.

Outsourced MPC. A natural way to overcome some of these issues is to employ a model similar to traditional cloud computing: Instead of running a protocol between all the input providing parties, delegate execution of the protocol to a much smaller (but hopefully more robust) set of dedicated "computing parties". It is also possible to consider a mixture of outsourced, and non-outsourced MPC protocols. I.e., where some of the computing parties also act as inputting parties. This might be necessary either for efficiency (for example, turning a two-party protocol into a three-party protocol by adding an additional non-colluding helper), or for privacy (the participating inputting party wishes to retain some influence in the protocol). This type of MPC has been utilized extensively in practice, e.g., for secure auctions [BCD+09], Iris matching [BKSW24], Cancer research [BCB+24] and social studies [BKK+16]. The outsourced MPC model presents both challenges and opportunities, which we attempt to tackle in this work.

1.1 Opportunity: reactive computations

The outsourced MPC model makes it natural to consider a setting where computations are *reactive*. That is, computations where inputs can be provided after an output has been performed. There are many types of computations which either require or benefit from being reactive. This is particularly true for KPIs, measurements, statistics, and machine learning systems. E.g., we can imagine a system that regularly updates a machine learning model based on new training data. If this machine learning model at the same time needs to be available for inference, then secure protocol updating the model needs to be reactive.

To formalize things a bit, we will consider a setup where computations take place in discrete "epochs". For each epoch i, we associate a pair of functions (F_i, G_i) where F_i is a "state" update function, and G_i is an output function. As input for the epoch, the multiparty computation receives the secret-shared state $[\mathsf{st}_{i-1}]$ from the previous epoch, as well as m secret-shared fresh inputs

$[x_1], \ldots, [x_m]$ received from clients.[3] Next, two things happen. First the state is updated as by running an MPC protocol $[\mathsf{st}_i] \leftarrow \Pi_{\mathsf{MPC}}(F_i, [\mathsf{st}_{i-1}], [x_1], \ldots, [x_m])$. Then the output is computed as $[y_i] \leftarrow \Pi_{\mathsf{MPC}}(G_i, [\mathsf{st}_i])$. The secret-shared output $[y_i]$ can then be revealed, and the updated state $[\mathsf{st}_i]$ can be input to the next epoch. This process is shown in Figure 1.

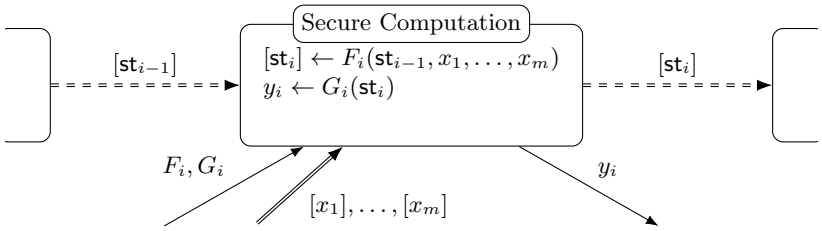

Fig. 1: Reactive computation. Values on double-drawn lines are delivered privately, whereas single-drawn lines are delivered through some public (but authenticated) channel. The dashed lines illustrate that the secure computation keeps its track of its state between epochs.

1.2 Challenge: long running computations

As there is no upper limit on the amount of epochs a reactive computation might run in, we may envision computations that simply run forever. This, however, creates a clash with the security definition. It might be reasonable to assume that at most 2 out of 5 nodes get compromised during a computation which takes one day. However, the same might not hold if the computation takes several weeks, months or even years. Although privacy for the compromised epoch cannot be guaranteed if the security assumption breaks[4], the same is not true for integrity.

An MPC protocol becomes *publicly verifiable* if the result y_i can be verified by an external party. The soundness of the verification must be independent of the trust assumption of the MPC protocol. One way of achieving publicly verifiable MPC, is to require the MPC to output a Zero-Knowledge proof π_i which proves correctness of y_i. That is, π_i is a proof that $y_i = G_i(F_i(\mathsf{st}_{i-1}, x_1, \ldots, x_m))$, that the inputs $x_1, \ldots, x_m$ to F_i correspond to the inputs delivered securely by the clients, and that st_{i-1} is the state from the previous epoch. However, note that

[3] The bracket notation $[v]$ used here indicates that the value v is split into some n shares, such that at least $t+1$ are needed in order to recover v.

[4] If the adversary exceeds the privacy bound in a given epoch, it only learns a snapshot of the state. Since the next committee is elected independently, a corrupted committee at epoch e does not hold the secret shares for epoch $e+1$. Thus, the adversary cannot observe the new client inputs x_{e+1} or compute the subsequent state $s_{e+1} = f(s_e, x_{e+1})$. Techniques such as proactive secret-sharing [HJKY95] can further reduce the risk of long-term compromise.

this statement quickly gets out of hand. Indeed, the actual statement that we would need to prove is closer to

1. $y_i = G(\mathsf{st}_i)$ for some st_i.
2. $\mathsf{st}_i = F(\mathsf{st}_{i-1}, x_1, \ldots, x_m)$ where x_1 to x_m were input in this epoch.
3. $\mathsf{st}_{i-1} = F(\mathsf{st}_{i-2}, x'_1, \ldots, x'_{m'})$ where x'_1 to $x'_{m'}$ were input in the previous epoch.
4. $\mathsf{st}_{i-2} =$ ... and so on.

Although the first step is easy to deal with, proving correctness of the auxiliary state input means that we need to track information about *all* previous states. This is not tenable for reactive computations which run forever. This verification bottleneck is further exacerbated by the need for proactive security. As mentioned above, preventing an adversary from gradually compromising the network over long periods requires periodically handing over the secret state to a freshly chosen committee of nodes. Consequently, a public verifier must not only check the evolution of the state across all past epochs, but also track the cryptographic chain of custody to ensure each committee legally received the valid state from the previous one.

This work. The focus of this work is to describe a method for making reactive MPC publicly verifiable. That is, is a method for constructing a *succinct* proof π_i that proves correctness of the output of the i'th epoch, as computed by some MPC protocol. As mentioned above, the fact that the computation continuously receives new inputs, provides outputs, and requires periodic committee handovers complicates things substantially.

To resolve these challenges, we introduce *Authenticated Outsourced Incrementally Verifiable MPC (IV-MPC)* . Our architecture resolves the tension between public verifiability and the prohibitive costs of verifying the correctness of a long MPC execution. Our core contributions are threefold:

Handling committee handovers via PoPCS. To maintain proactive security, the secret-shared state must be periodically transferred between committees. We introduce a novel primitive called *private opening PCS* (PoPCS) to make this process publicly verifiable. PoPCS allows an incoming committee to prove that they received a valid resharing of the secret state held by the previous committee. This ensures any external party can verify the correctness of the handover protocol without learning the underlying state.

Verifying MPC via distributed CSP and sumcheck. We make the sequential execution of the MPC protocol publicly verifiable by developing a novel distributed proof system for state transitions based on techniques from *Compressed Σ-Protocols* (CSP) and *Sumcheck*. This allows the current committee to (collaboratively) prove step-by-step evolution of the shared state.

Decoupled architecture: IV-MPC + stateless handovers. Naively verifying a continuous chain of state updates and committee handovers would require verification time linear in the number of epochs. To achieve succinctness, we

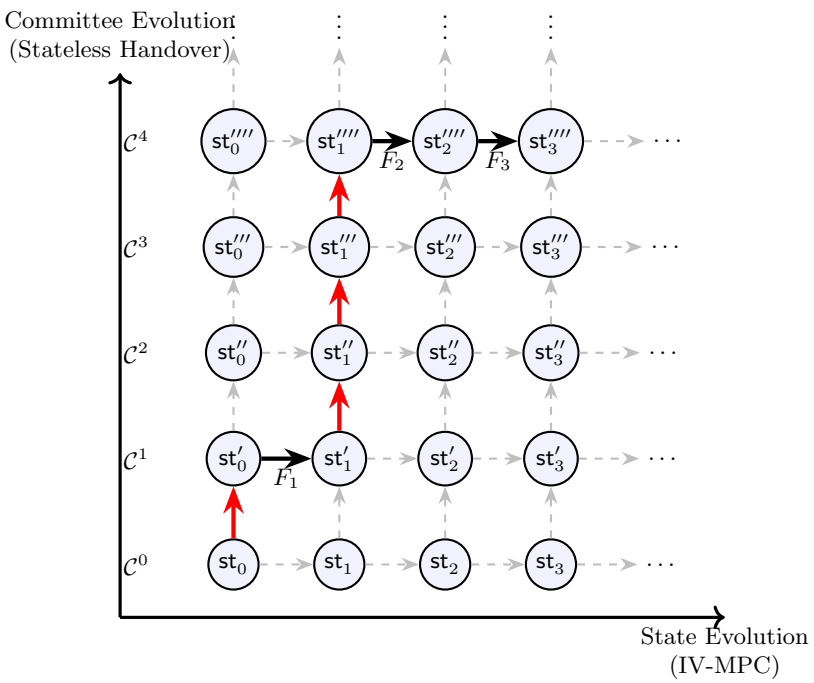

Fig. 2: The horizontal axis represents the step-by-step MPC execution updating the secret state (tracked via IV-MPC proofs). The vertical axis represents the proactive committee handovers maintaining custody of the state (tracked via stateless handover proofs).

decouple the two axes of verification (illustrated in fig. 2). For the horizontal axis, we employ techniques from incrementally verifiable computation (IVC) to recursively accumulate the proofs of correctness for the MPC executions, yielding Incrementally Verifiable MPC (IV-MPC). For the vertical axis, we observe that because resharing a state does not change its value, we can anchor the vertical handover proofs to the canonical state commitment produced by the most recent MPC update. This makes handovers *stateless*: verifying the latest handover suffices to guarantee correctness without needing to verify the entire chain of handovers. Thus, we use recursion (IVC) only for horizontal state updates, while our stateless handover technique keeps vertical proof generation and verification independent of the number of epochs.

1.3 Our construction

The problem we face is to compute $\pi_i = (\pi_i^G, \pi_i^F)$ such that π_i^G is a proof that the output was produced correctly, given, say, a commitment of the state, and where π_i^F is a proof that the state was computed correctly. The former is relatively straightforward to handle and is not the focus of our work. The main technical challenge is computing π_i^F over an unbounded number of epochs. Suppose for now that $F_i = F_j = F$; i.e., that all state updates are done in the same manner. Then we can view π_i^F as a proof that $\mathsf{st}_i = F^{(i)}(\mathsf{st}_0, w_i)$ for a known initial state st_0 and an auxiliary witness w_i consisting of a list of clients inputs $w_i = (x_{i,1}, \ldots, x_{i,m})$. Thus, we may utilize *incrementally verifiable computation* (IVC) [Val08] to compute π_i^F.

There are still several challenges, however.

Input binding. IVC allows us to construct a π_i^F which shows that $\mathsf{st}_i = F^{(i)}(\mathsf{st}_0)$, and where each application of F also receives an additional witness w_i. This witness should capture the inputs received for the i'th round of computation, but how? The inputs must be "fixed" in a manner which cannot be changed by the MPC nodes—Indeed, we ultimately want a proof which shows correctness of the computation (and thus the state) *even* if all the MPC nodes misbehave. If all the MPC nodes misbehave, nothing is stopping them from choosing new inputs for a previous state in order to obtain a different output for the current epoch.[5] Suppose there is fixed a "trusted signer" which is tasked with producing a signature for each input under the public key pk_{auth}. Then the witness w_i can be defined to consist of "*input* $x_{i,k}$ *and signature* $\sigma_{i,k}$ *such that* $\mathsf{verf}(\mathsf{pk}_{auth}, x_{i,k}||i||k, \sigma_{i,k}) = 1$" for each $k \in [m]$. Realizing this authority can be done using blind signatures, and an assumption that it acts semi-honestly and independent of the MPC adversary.

State update function. Standard IVC requires that each update be performed in the same manner. Above this is expressed by requiring that $F_i = F_j$ for all i, j. There are two ways to overcome this: First, we may use non-uniform IVC scheme such that, each step, we may select F_i from some set of pre-defined functions. A second approach—especially relevant if F_i is not known ahead of time—is to utilize universal circuits. In this case $F_i = F_j = UC$, with $|UC| = O(n \log n)$ and n the number of gates. This will allow our construction to work for any function of $O(n)$ gates. The actual circuit would then be expressed as an additional input, which would be handled as explained above.

MPC friendliness. Finally, we must recall that the point of our endeavor is to get something which works in MPC. Thus, a significant challenge is to come up with a proof system that is MPC friendly, and which can be used in an IVC scheme. To address this challenge we make use of compressed Σ-protocols [AC20].

[5] Note that this issue does not show up for the current epoch, as here we can assume that the verifier knows e.g., commitments of the current inputs. We cannot assume that the verifier knows the commitments for inputs made in *all* epochs, as that would clearly make the proof non-succinct.

1.4 Related work

Publicly verifiable MPC. Typically, MPC protocols ensure internal correctness among participants, but external auditability requires correctness to be publicly verifiable (in this work, we use the terms *publicly verifiable* and *publicly auditable* interchangeably). First introduced by Baum et al. [BDO14], public verifiability allows external observers to validate the correctness of an MPC execution using only public data. Subsequent works have optimized this property under various execution paradigms [SV15, BOSS20, YLRD26]. Our design shares this goal but targets long-running, ongoing state computations. We focus on achieving public verifiability with minimal overhead on the MPC committee by delegating all non-secret computations to an external helper prover $\mathtt{P}^*$.

Collaborative SNARKs and collaborative IVC. Collaborative zk-SNARKs [OB22] enable a set of mutually distrustful provers holding secret shares of a witness to jointly generate a succinct proof. Follow-up works have heavily optimized this paradigm; for instance, zkSaaS [GGJ+23] observed that certain heavy operations (such as multi-scalar multiplication) in popular SNARKs exhibit structures that are highly MPC-friendly, and using packed secret sharing [FY92] can yield sublinear complexity per prover. More recently, Liu et al. [LZW+25] proposed a fully scalable collaborative SNARK framework where all provers achieve running times sublinear to the circuit size by relying on a proof system based on sumcheck protocol. Recently, Tshuva et al. [TGK+26] extended this paradigm to sequential computation by formalizing *Collaborative IVC*, where a committee of provers interactively executes a distributed IVC proof generation protocol. While our work shares a similar objective, our approach differs significantly in design: our committee only executes an efficient distributed sumcheck and compressed Σ-protocols to produce a public uncompressed proof transcript, leaving the heavy recursive computation entirely to a single untrusted helper $\mathtt{P}^*$.

2 Preliminaries

2.1 Network assumptions and adversarial model

We adopt a similar system model and communication assumptions as in [MZW+19, BGG+20], with adjustments tailored to our setting. We assume a public bulletin board (e.g., a blockchain) that serves three essential roles: a synchronization mechanism, an authenticated broadcast channel, and a lightweight public key infrastructure (PKI). For synchrony, we assume all participants track a global clock (e.g., the current block height). Communication proceeds in discrete rounds: any party can broadcast a message in round i, and it is guaranteed to be visible to all parties by round $i+\delta$, for a known bound δ. The authenticated broadcast channel guarantees message integrity, sender authenticity, and global consistency: a message posted in round i is verifiably from its sender and observed by all parties in the same round. This channel is fully public, exposing all messages and senders to every participant, including adversaries. It also serves as a PKI, with parties

broadcasting their public key to maintain a globally consistent, authenticated key registry.

The adversarial model we consider is a *mobile adversary* capable of observing the broadcast channel and dynamically corrupting parties over time. We define two adversarial bounds for privacy and integrity:

- *Privacy (honest majority).* We assume the underlying MPC protocol operates in the honest-majority setting. Specifically, the adversary is constrained by a corruption budget of $t < n/2$ during any single epoch. The bound is required by our compiler to reconstruct the degree-$2t$ polynomials generated during the sumcheck phase in our protocol (see section 5.2).
- *Integrity (full corruption).* For integrity which is the focus of our work, we assume an adversary capable of corrupting the entire MPC committee ($t = n$). Even under full compromise, we must ensure that the adversary cannot forge a valid proof for an incorrect state update.

2.2 Secret sharing, verifiable extensions, and DPVSS

A (t, n)-secret sharing scheme distributes a secret s into n shares $s_1, \ldots, s_n$ such that any $t + 1$ or more shares can reconstruct s, while any set of t or fewer shares reveals no information about the secret[Sha79]. This mechanism typically involves three roles: a *dealer*, who distributes the secret; *shareholders*, who store shares; and a *combiner*, who reconstructs the secret when needed.

To address active adversaries who corrupt shares during reconstruction, *Verifiable Secret Sharing (VSS)* augments secret sharing with commitments, allowing participants to verify share consistency and detect invalid shares [Fel87, Ped92, Sch99, DXT$^+$23]. Throughout this work, when we refer to verifiability, we mean public verifiability—that is, the correctness of a sharing can be verified by any third party, not just the shareholders.

Proactive Secret Sharing (PSS), introduced in [HJKY95], protects secrets in long-lived systems against mobile adversaries capable of eventually compromising more than t parties. This is achieved by periodically refreshing shares without changing the underlying secret. As long as an adversary never acquires more than t shares between successive renewal, PSS ensures the secrets remain secure against continuous player compromise. *Proactive Verifiable Secret Sharing* (PVSS) combines proactive share renewal with verifiability. A natural extension, *Dynamic Proactive Verifiable Secret Sharing (DPVSS)* [BELO15, MZW$^+$19, GKM$^+$22], allows the committee of shareholders to change over time while preserving the confidentiality and integrity of the secret. The central challenge in DPVSS is securely executing the resharing protocol in the presence of up to $2t$ malicious parties—t in the outgoing committee and t in the incoming one. Below we define DPVSS using fixed parameters, though in practice n and t may vary between committees over time. This is for notational simplicity and ease of exposition, and generalizing the construction to variable (t, n) is straightforward.

Definition 1. *A* dynamic proactive (t, n)-secret sharing scheme *provides the following procedures:*

- ***Setup.*** *Provides the public parameters required for the scheme.*
- ***Sharing.*** *Distributes a secret s among an initial holding committee of size n.*
- ***Handover.*** *An n-party protocol that securely reshares existing shares to a new holding committee.*
- ***Reconstruction.*** *Recovers the secret s from $t + 1$ or more shares of the current committee.*
- ***Verification.*** *A public procedure that allows anyone to check a sharing or handover was done correctly.*

In addition to completeness, a DPVSS protocol must satisfy the following properties for any PPT adversary $\mathcal{A}$: (i) Secrecy: if $\mathcal{A}$ corrupts no more than t nodes during **Sharing** and **Reconstruction**, and no more than $2t$ nodes in two successive committees during **Handover**, then $\mathcal{A}$ learns no information about the shared secret. (ii) Integrity: the shared secret remains unchanged and can be reconstructed from any set of $t + 1$ honest shares, regardless of the number of nodes corrupted by $\mathcal{A}$.

2.3 Polynomial commitments

Informally, a polynomial commitment scheme (PCS) enables one to compute a short commitment value C to a polynomial f over a finite field $\mathbb{F}$, so that they can later compute an opening proof that certifies the polynomial committed to by C evaluates to $v \in \mathbb{F}$ at some point $z \in \mathbb{F}$. It consists of PPT algorithms $\mathsf{PCS} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open}, \mathsf{VerifyOpen})$:

- Setup Generates commitment keys.
- Commit Produces a commitment C to a polynomial f.
- Open Generates an evaluation proof op for $y = f(z)$.
- $\mathsf{VerifyOpen}$ Checks op against the commitment C and evaluation point z.

We assume the PCS satisfies standard *correctness* and *binding* properties. The binding property guarantees that once a commitment is made, an adversary cannot produce valid openings for two different values at a given point. See appendix A.1 for a formal definition of these properties.

2.4 Non-interactive arguments of knowledge (NARK)

A (zero-knowledge) non-interactive argument of knowledge (zk-NARK) is a cryptographic protocol that allows a prover to convince a verifier that it knows a satisfying witness to a specific statement without revealing the witness. Our zk-NARK is defined by algorithms $(\mathcal{G}, \mathcal{I}, \mathcal{P}, \mathcal{V})$:

- $\mathcal{G}$ Generates global public parameters pp and a simulation trapdoor td.
- $\mathcal{I}$ Takes a specific circuit and produces proving and verifying keys pk and vk, respectively for the prover and verifier.

– $\mathcal{P}$ Uses pk, a statement $\mathbb{x}$, and a witness $\mathbb{w}$ to produce a proof π.
– $\mathcal{V}$ Uses vk to check π against the statement $\mathbb{x}$, outputting accept or reject.

Informally, a NARK must satisfy two properties: *completeness* which ensures that an honest prover with a valid witness can always convince an honest verifier, and *knowledge soundness* which guarantees that any prover who produces a valid proof must actually "know" the witness. Optionally, a NARK may satisfy zero-knowledge (zk-NARK) ensuring the proof reveals nothing about the witness (beyond the fact that $\mathcal{P}$ knows a satisfying witness). We provide formal definitions of these properties in appendix A.2.

2.5 Accumulation schemes and IVC

We define accumulation schemes and incrementally verifiable computation (IVC) informally here, deferring the full formal definitions to appendix A.3 and appendix A.4. An accumulation scheme for a predicate Φ (e.g., a SNARK verifier) allows us to aggregate multiple "predicate instances" into one. Instead of verifying every proof instance individually, we maintain a running accumulator. The scheme consists of algorithms $(\mathrm{G}, \mathrm{I}, \mathrm{P}, \mathrm{V}, \mathrm{D})$:

– The Setup and Indexing algorithms (G, I) for a specific predicate Φ.
– Prover P that takes a current accumulator acc and a new predicate input q, producing an updated accumulator acc^*.
– Verifier V that checks the transition from acc to acc^* was done correctly.
– Decider D which is the final and more expensive check that validates the entire chain of accumulation.

Soundness property in accumulation schemes ensures that if D accepts the final accumulator and V accepts the latest update, then both the new input and the previous accumulator were valid (with overwhelming probability).

IVC [Val08] enables the verification of long running, iterative computations $z_i = F^i(z_0)$. Rather than re-verifying the entire computation path, a prover can take a proof π_{i-1} of the previous step and "increment" it to produce a new proof π_i that includes the most recent step. An IVC scheme consists of algorithms $(\mathbb{G}, \mathbb{I}, \mathbb{P}, \mathbb{V})$:

– The Setup and Indexing algorithms $(\mathbb{G}, \mathbb{I})$ for a specific function F.
– The Prover $\mathbb{P}$ who takes (z_{i-1}, π_{i-1}) and a new witness w_{i-1} to compute z_i and the updated proof π_i.
– The Verifier $\mathbb{V}$ who checks π_i to confirm that z_i is the result of i valid applications of F.

We require IVC to be *succinct* (proof size is independent of the number of steps i) and satisfy *knowledge soundness*, ensuring that a valid proof implies the prover actually knows the sequence of intermediate witnesses.

3 Private opening PCS

Recall that our architecture splits verification into two axes: state evolution and secure handovers. Let C_i denote the canonical commitment to st_i, computed in the i-th round of MPC. To enable succinct verification of handovers for st_i, we prove (recursively) that every new committee received valid shares of st_i corresponding to C_i. We do so by introducing a PCS variant with *private openings* (PoPCS). Unlike a standard PCS, where an opening reveals the evaluation $y = f(z)$ of the committed polynomial f, a PoPCS keeps y hidden. Instead, the prover outputs a ciphertext ct, encrypted under a receiver's public key pk, alongside a proof showing that ct correctly encrypts the evaluation of the committed polynomial at (public) z. This forms the core of our VSS protocol, enabling incoming committees to prove the integrity of the handover process while preserving the state privacy bound.

In the following, we formalize the syntax and security definitions for this new PoPCS primitive.

Definition 2 (Private opening polynomial commitment schemes). *Let* $\mathsf{PKE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *be a public-key encryption scheme with message space* $\mathbb{F}$*. A private opening polynomial commitment scheme w.r.t.* PKE *is a tuple of PPT algorithms* $\mathsf{PoPCS} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open}, \mathsf{VerifyOpen})$ *defined as follows:*

- $\mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda, 1^d)$*: The setup algorithm takes as input a security parameter* λ *and an upper bound* d*, and returns a commitment key* ck*.*
- $C \leftarrow \mathsf{Commit}(\mathsf{ck}, f; \omega)$*: The polynomial commitment algorithm takes as input a commitment key* ck *and a polynomial* $f \in \mathbb{F}[X]$*, and returns commitment* C*. The randomness* ω *is used for hiding commitments.*
- $(\mathsf{op}, \mathsf{ct}) \leftarrow \mathsf{Open}(\mathsf{ck}, f, z, \mathsf{pk}; \omega)$*: The opening algorithm takes a commitment key* ck*, polynomial* $f \in \mathbb{F}[X]$*, evaluation point* $z \in \mathbb{F}$*, a public key* pk*, and randomness* ω*, and returns an opening proof* op *and a ciphertext* ct*.*
- $b \leftarrow \mathsf{VerifyOpen}(\mathsf{ck}, C, \mathsf{op}, z, \mathsf{ct})$*: The verification algorithm takes as input a commitment key* ck*, a commitment* C*, an opening proof* op*, an evaluation point* z*, and a ciphertext* ct*, and returns a bit* b*.*

A private opening polynomial commitment scheme should satisfy the following properties.

Definition 3 (Completeness). *For any* $\lambda, d \in \mathbb{N}$*, and any adversary* $\mathcal{A}$*, the following probability is 1.*

$$\Pr\left[\begin{array}{c}\deg(f) < d \\ \wedge \mathsf{VerifyOpen}(\mathsf{ck}, C, \mathsf{op}, z, \mathsf{ct})\end{array} \;\middle|\; \begin{array}{r}\mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda, 1^d); (f, z, \mathsf{pk}, \omega) \leftarrow \mathcal{A}(\mathsf{ck}) \\ C \leftarrow \mathsf{Commit}(\mathsf{ck}, f; \omega) \\ (\mathsf{op}, \mathsf{ct}) \leftarrow \mathsf{Open}(\mathsf{ck}, f, z, \mathsf{pk}; \omega)\end{array}\right]$$

Definition 4 (Extractability). *For any $\lambda, d \in \mathbb{N}$ and any PPT adversary $\mathcal{A}$, there exists an efficient extractor* Ext *such that the following holds.*

$$\Pr\left[\begin{array}{c|r} \mathsf{VerifyOpen}(\mathsf{ck}, C, \mathsf{op}, z, \mathsf{ct}) = 1 & \mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda, 1^d) \\ \Downarrow & (\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\lambda) \\ \mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) = f(z) & (C, z, \mathsf{ct}, \mathsf{op}) \leftarrow \mathcal{A}(\mathsf{ck}, \mathsf{sk}) \\ \wedge \deg(f) < d & f \leftarrow \mathsf{Ext}_{\mathcal{A}}(\mathsf{ck}, \mathsf{sk}) \end{array}\right] \geq 1 - \mathsf{negl}(\lambda).$$

Definition 5 (Zero-knowledge). *For any $\lambda, d \in \mathbb{N}$, and any PPT adversary $\mathcal{A}$, there exists an efficient simulator $\mathcal{S}$, such that for any polynomial f of degree at most d,*

$$|\Pr\left[\boldsymbol{Real}_{\mathcal{A},f}(1^\lambda) = 1\right] - \Pr\left[\boldsymbol{Ideal}_{\mathcal{A},\sim}(1^\lambda) = 1\right]| \leq \mathsf{negl}(\lambda)$$

where $\boldsymbol{Real}_{\mathcal{A},f}(1^\lambda)$ and $\boldsymbol{Ideal}_{\mathcal{A},\sim}(1^\lambda)$ are defined as follows:

$\boldsymbol{Real}_{\mathcal{A},f}(1^\lambda)$	$\boldsymbol{Ideal}_{\mathcal{A},\mathcal{S}}(1^\lambda)$
$\mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda, 1^d)$	
$C \leftarrow \mathsf{Commit}(\mathsf{ck}, f; \omega)$	$(\mathsf{ck}, \mathsf{td}, C) \leftarrow \mathcal{S}(1^\lambda, 1^d)$
$(z, \mathsf{pk}) \leftarrow \mathcal{A}(\mathsf{ck}, C)$	$(z, \mathsf{pk}) \leftarrow \mathcal{A}(\mathsf{ck}, C)$
$(\mathsf{op}, \mathsf{ct}) \leftarrow \mathsf{Open}(\mathsf{ck}, f, z, \mathsf{pk}; \omega)$	$(\mathsf{op}, \mathsf{ct}) \leftarrow \mathcal{S}(\mathsf{td}, C, z, f(z), \mathsf{pk})$
$b \leftarrow \mathcal{A}(\mathsf{ck}, C, z, \mathsf{op}, \mathsf{ct})$	$b \leftarrow \mathcal{A}(\mathsf{ck}, C, z, \mathsf{op}, \mathsf{ct})$
return b	**return** b

3.1 Construction based on Bulletproofs

We present a simple construction based on Bulletproofs. For the PKE, we consider an ElGamal-style scheme over a group $\mathbb{G}$ of order q. Key generation samples $\mathsf{sk} \leftarrow\!\!\$\, \mathbb{F}_q$ and sets $\mathsf{pk} = [\mathsf{sk}]\mathsf{G}$ for some fixed generator $\mathsf{G} \in \mathbb{G}$. To encrypt a group encoding of a message $m \in \mathbb{F}_q$ under base $\mathsf{U} \in \mathbb{G}$ with randomness $\hat{\omega} \in \mathbb{F}_q$, the encryption algorithm outputs

$$\mathsf{Enc}_{\mathsf{pk}}([m]\mathsf{U}; \hat{\omega}) = \big([\hat{\omega}]\mathsf{G},\ [m]\mathsf{U} + [\hat{\omega}]\mathsf{pk}\big).$$

For a ciphertext (c_1, c_2), decryption recovers the group encoding of the message as $[m]\mathsf{U} = c_2 - [\mathsf{sk}]c_1$.

Our private-opening PCS $\mathsf{PoPCS} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open}, \mathsf{VerifyOpen})$ is defined as follows.

- $\mathsf{Setup}(1^\lambda, d)$: given a security parameter λ and $d = 2^k$, runs $(\mathbb{G}, q, \mathsf{G}) \leftarrow \mathsf{GroupGen}(1^\lambda)$ and $(\mathbf{G} = (G_0, \ldots, G_{d-1}), H) \leftarrow \mathsf{GenSamp}(\mathbb{G}, d+1)$, and returns $\mathsf{ck} = (\mathbb{G}, q, \mathsf{G}, \mathbf{G}, H)$.

– $\mathsf{Commit}(\mathsf{ck}, f; \omega)$: the commitment to a polynomial $f(X) \in \mathbb{F}_q^{<d}[X]$ is defined as $C = \langle \mathbf{f}, \mathbf{G} \rangle + [\omega]H$, where $f_i \in \mathbb{F}_q$ is the coefficient of $f(X)$ for the i-th degree term.
– $(\mathsf{Open}, \mathsf{VerifyOpen})$ is an argument of knowledge for the NP relation

$$\{((C, z, \mathsf{U}, \mathsf{pk}, \mathsf{ct}); (\mathbf{f}, \omega, \hat{\omega})) : C = \langle \mathbf{f}, \mathbf{G} \rangle + [\omega]H \wedge \mathsf{ct} = \mathsf{Enc}_{\mathsf{pk}}([\langle \mathbf{f}, \mathbf{z} \rangle]\mathsf{U}; \hat{\omega})\}$$

where $\mathbf{z} = (1, z, \ldots, z^{d-1})$ is the vector of powers of z. The protocol is similar to the inner product argument (IPA) of Bulletproofs with minor modifications: unlike the original protocol, we verify the proof w.r.t. an encryption of the evaluation, namely ct which is part of the statement. In particular, we incorporate ct into the commitment using a challenge value e, and use the Bulletproof IPA to show that

$$D = \langle \mathbf{f}, \mathbf{G} \rangle + [\omega]H + e([\hat{\omega}]\mathsf{G} + [\hat{\omega}]\mathsf{pk} + [\langle \mathbf{f}, \mathbf{z} \rangle]\mathsf{U}).$$

The IPA proceeds in $k = \log_2 d$ iterations. The argument begins at round k with the vectors $\mathbf{f}^k := \mathbf{f}$, $\mathbf{G}^k := \mathbf{G}$, and $\mathbf{z}^k := \mathbf{z}$. During each round $j = k, k-1, \ldots, 1$, the prover calculates two values L_j and R_j by computing specific inner products of $\mathbf{f}$ with $\mathbf{G}$ and $\mathbf{z}$, wherein the lower half of $\mathbf{f}$ is paired with the upper half of $\mathbf{G}$ and $\mathbf{z}$, and vice versa:

$$L_j = \langle \mathbf{f}_{\mathrm{lo}}^{(j)}, \mathbf{G}_{\mathrm{hi}}^{(j)} \rangle + [l_j]H + e[\langle \mathbf{f}_{\mathrm{lo}}^{(j)}, \mathbf{z}_{\mathrm{hi}}^{(j)} \rangle]\mathsf{U}$$
$$R_j = \langle \mathbf{f}_{\mathrm{hi}}^{(j)}, \mathbf{G}_{\mathrm{lo}}^{(j)} \rangle + [r_j]H + e[\langle \mathbf{f}_{\mathrm{hi}}^{(j)}, \mathbf{z}_{\mathrm{lo}}^{(j)} \rangle]\mathsf{U}$$

In each iteration, the prover uses random elements $l_j, r_j \in \mathbb{F}_q$ to make the commitments hiding. Subsequently, the prover uses the challenge u_j received from the verifier to compress both the lower and upper halves of the input vectors (i.e., $\mathbf{f}, \mathbf{G}$, and $\mathbf{z}$), resulting in new vectors half the length of their original.

$$\mathbf{f}^{(j-1)} = \mathbf{f}_{\mathrm{hi}}^{(j)} + \mathbf{f}_{\mathrm{lo}}^{(j)} \cdot u_j^{-1}$$
$$\mathbf{z}^{(j-1)} = \mathbf{z}_{\mathrm{hi}}^{(j)} + \mathbf{z}_{\mathrm{lo}}^{(j)} \cdot u_j$$
$$\mathbf{G}^{(j-1)} = \mathbf{G}_{\mathrm{hi}}^{(j)} + \mathbf{G}_{\mathrm{lo}}^{(j)} \cdot u_j$$

These vectors will then be provided as input to the next iteration $j-1$. Upon concluding the final iteration $\mathrm{j} = 1$, each of the vectors $\mathbf{f}^{(0)}, \mathbf{G}^{(0)}$, and $\mathbf{z}^{(0)}$ will be of length 1, using which along with the challenges $\{u_j\}$ and cross-terms $\{L_j, R_j\}$ one can verify that ct is an encryption of the evaluation of the polynomial contained "inside" the commitment C at z. In more detail, the verifier computes the commitment $W = D + \sum_{j=1}^{k} [u_j^{-1}]L_j + \sum_{j=1}^{k} [u_j]R_j$ and checks if the prover has demonstrated knowledge of field elements $(\mathbf{f}^{(0)}, \hat{\omega}, \xi, m)$, where $m = \langle \mathbf{f}, \mathbf{z} \rangle$, and $\xi = \omega + \sum_{j=1}^{k} l_j u_j^{-1} + \sum_{j=1}^{k} r_j u_j$, such that

$$W = [\mathbf{f}^{(0)}]\mathbf{G}^{(0)} + [\xi]H + e[\hat{\omega}]\mathsf{G} + e[\mathbf{f}^{(0)}\mathbf{z}^{(0)}]\mathsf{U} + e[\hat{\omega}]\mathsf{pk},$$
$$\wedge \quad \mathsf{ct} = ([\hat{\omega}]\mathsf{G}, [\hat{\omega}]\mathsf{pk} + [m]\mathsf{U}).$$

In appendix B, we show how to instantiate this scheme using CL encryption [CL15], which avoids discrete logarithm computation during decryption.

4 The vertical axis: stateless handover

In this section, we define the vertical axis of our design. Following the discussion in section 1, state handovers must be tracked independently of state evolution. Over the lifetime of the network, the active committee may change multiple times between computation steps. To ensure that verification cost remains independent of the number of handovers, we rely on the observation that all handovers can be tied to the original state commitment (which we refer to as the *canonical state commitment*) output by the most recent state computation. Building on this idea, we first present a simple dynamic proactive VSS scheme built on top of our PoPCS primitive to handle a single handover (section 4.1). We then show how anchoring these proofs to the state's canonical commitment makes the handovers stateless (section 4.2).

4.1 Dynamic proactive VSS from PoPCS

At the start of each epoch in which only a handover (i.e., without MPC execution) occurs, a handover protocol transfers the secret shares from the current committee to the next. Committees are denoted by $\mathcal{C}^i$ for $i = 0, 1, \ldots$, with their corresponding public keys $\mathsf{pk}_1^i, \ldots, \mathsf{pk}_n^i$. We assume that before $\mathcal{C}^i$ begins its operation, the public keys of the incoming committee $\mathcal{C}^{i+1}$ are already published. During the handover, the old committee $\mathcal{C}^i$ reshares the secret to $\mathcal{C}^{i+1}$ and securely erases its local state. $\mathcal{C}^{i+1}$ then assumes responsibility for the state.

A standard framework for constructing VSS protocols uses polynomial commitments. A dealer selects a random degree-t polynomial f whose constant term encodes the secret s, broadcasts a commitment C to f, and privately sends the evaluation $y_j = f(x_j)$ to participant j. The dealer also provides a proof to each participant verifying that their received share is consistent with C. We build our DPVSS by upgrading this standard framework with the PoPCS introduced in section 3, so that opening proofs are publicly verifiable by anyone not only by the recipient. Across successive handovers, naively including all prior proofs would cause both the overall proof size and the verification to grow linearly in the number of handovers. To avoid this, we instead verify handover proofs against a fixed canonical commitment C produced in the most recent MPC execution. This commitment is already a core component of the distributed ZK proof for verifiable MPC that runs alongside the handover, making its inclusion here natural (as we elaborate later). This approach decouples handover cost from the number of epochs.

4.1.1 Construction. Our construction uses a PoPCS $\mathsf{PoPCS} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open}, \mathsf{VerifyOpen})$ defined w.r.t. a PKE scheme $\mathsf{PKE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$. For ease of presentation, we assume that all epoch committees are of fixed size n, and the protocol tolerates up to t corrupt members per committee. This allows us to associate a fixed evaluation point α_j with each party $j \in [n]$. The construction naturally extends to a setting where each epoch i has parameters (n_i, t_i).

Setup. For a degree bound t, the setup algorithm runs $\mathsf{ck} \leftarrow \mathsf{PoPCS.Setup}(1^\lambda, 1^t)$. We assume all parties are given access to ck, an authenticated broadcast channel and the PKE parameters.

Initial sharing. On secret s, the dealer chooses randomness r, and two random degree-t polynomials f_0 and g_0, where $f_0(0) = s$ and $g_0(0) = r$. For a given initial committee $\mathcal{C}^0 = \{\mathsf{pk}_1^0, \ldots, \mathsf{pk}_n^0\}$, the dealer then sends to party j the shares $s_j = f_0(\alpha_j)$ and $r_j = g_0(\alpha_j)$. Finally, the dealer computes commitments to f_0 and g_0 as $C_{f_0} \leftarrow \mathsf{PoPCS.Commit}(\mathsf{ck}, f_0; \omega_s)$ and $C_{g_0} \leftarrow \mathsf{PoPCS.Commit}(\mathsf{ck}, g_0; \omega_r)$, and generates ciphertext-proof pairs

$$(\mathsf{op}_{1,j}, \mathsf{ct}_{1,j}) \leftarrow \mathsf{PoPCS.Open}(\mathsf{ck}, f_0, \alpha_j, \mathsf{pk}_j; \omega_s)$$
$$(\mathsf{op}_{2,j}, \mathsf{ct}_{2,j}) \leftarrow \mathsf{PoPCS.Open}(\mathsf{ck}, g_0, \alpha_j, \mathsf{pk}_j; \omega_r)$$

and broadcasts $\{\mathsf{op}_{i,j}, \mathsf{ct}_{i,j}\}_{i\in[2],j\in[n]}$ and C_{f_0}.

From here on, we assume that at the end of each epoch i, the committee $\mathcal{C}^i$ holds shares of (s, r), which must be passed to the next committee $\mathcal{C}^{i+1}$. We also assume the existence of the canonical state commitment $C = [s]\mathsf{U}_1 + [r]\mathsf{U}_2$. Although treated here abstractly as an output of the initial sharing phase, in our full protocol, C is the commitment produced by the most recent MPC execution step. All subsequent vertical handovers transferring shares of this same state in C are therefore verified against C.

The handover and reconstruction. Inductively, the parties in $\mathcal{C}^i$ hold evaluations of degree-t polynomials f_i and g_i with $f_i(0) = s$ and $g_i(0) = r$, where party j holds the shares $s_j = f_i(\alpha_j)$ and $r_j = g_i(\alpha_j)$.

(1) Outgoing share transfer. A party j with public key pk_j and holding (s_j, r_j) proceeds as follows:

1. Sample random polynomials $\bar{f}_j, \bar{g}_j$ of degree t such that $\bar{f}_j(0) = s_j$ and $\bar{g}_j(0) = r_j$.
2. Compute polynomial commitments to $\bar{f}_j$ and $\bar{g}_j$ as
$$C_{\bar{f}_j} \leftarrow \mathsf{PoPCS.Commit}(\mathsf{ck}, \bar{f}_j; \omega_{sj})$$
$$C_{\bar{g}_j} \leftarrow \mathsf{PoPCS.Commit}(\mathsf{ck}, \bar{g}_j; \omega_{rj}).$$
3. For each $\alpha \in I$, compute the shares $s_{j,\alpha} = \bar{f}_j(\alpha)$ and $r_{j,\alpha} = \bar{g}_j(\alpha)$, and run
$$(\bar{\mathsf{op}}_{1,\alpha}, \bar{\mathsf{ct}}_{1,\alpha}) \leftarrow \mathsf{PoPCS.Open}(\mathsf{ck}, \bar{f}_j, \alpha, \mathsf{pk}_\alpha; \omega_{sj})$$
$$(\bar{\mathsf{op}}_{2,\alpha}, \bar{\mathsf{ct}}_{2,\alpha}) \leftarrow \mathsf{PoPCS.Open}(\mathsf{ck}, \bar{g}_j, \alpha, \mathsf{pk}_\alpha; \omega_{rj})$$
4. Perform an additional polynomial opening at 0:
$$(\bar{\mathsf{op}}_{1,\star}, \bar{\mathsf{ct}}_{1,\star}) \leftarrow \mathsf{PoPCS.Open}(\mathsf{ck}, \bar{f}_j, 0, \mathsf{pk}_j; \omega_{sj})$$
$$(\bar{\mathsf{op}}_{2,\star}, \bar{\mathsf{ct}}_{2,\star}) \leftarrow \mathsf{PoPCS.Open}(\mathsf{ck}, \bar{g}_j, 0, \mathsf{pk}_j; \omega_{rj})$$
where $\bar{\mathsf{ct}}_{1,\star}$ encrypts $[\bar{f}_j(0)]\mathsf{U}_1$ and $\bar{\mathsf{ct}}_{2,\star}$ encrypts $[\bar{g}_j(0)]\mathsf{U}_2$. Generate a Σ-protocol proof $\pi_{j,\star}$ showing that $\bar{\mathsf{ct}}_{1,\star} \cdot \bar{\mathsf{ct}}_{2,\star}$ encrypts the same message as the commitment $C_j = [\bar{f}_j(0)]\mathsf{U}_1 + [\bar{g}_j(0)]\mathsf{U}_2$.

5. Broadcast commitments $(C_{\bar{f}_j}, C_{\bar{g}_j}, C_j)$, openings $\{\bar{\mathsf{op}}_{i,\alpha}, \bar{\mathsf{ct}}_{i,\alpha}\}_{i\in[2],\alpha\in I\cup\{\star\}}$, and the Σ-protocol proof $\pi_{j,\star}$.

(2) Incoming share reception. Upon receiving the broadcast messages from the outgoing share transfer, every party k in the new committee $\mathcal{C}^{i+1}$ processes the incoming data as follows:

1. From the received messages, select $t+1$ opening vectors, say those sent by parties $\ell_1, \ldots, \ell_{t+1}$, and extract the components addressed to party k. This yields all pairs $(\bar{\mathsf{op}}_{i,k}, \bar{\mathsf{ct}}_{i,k})_{i=1,2}$ sent by parties $\ell_1, \ldots, \ell_{t+1}$.
2. Using secret key sk_k, decrypt each ciphertext and recover the underlying values $s_{\ell_1,k}, \ldots, s_{\ell_{t+1},k}, r_{\ell_1,k}, \ldots, r_{\ell_{t+1},k}$.
3. Reconstruct the share of the global secrets by performing the Lagrange interpolation

$$s_k = \sum_{\ell\in\{\ell_1,\ldots,\ell_{t+1}\}} \lambda_\ell \cdot s_{\ell,k} \qquad ; \qquad r_k = \sum_{\ell\in\{\ell_1,\ldots,\ell_{t+1}\}} \lambda_\ell \cdot r_{\ell,k},$$

where $\lambda_{\ell_1}, \ldots, \lambda_{\ell_{t+1}}$ are the Lagrange coefficients associated with indices $\ell_1, \ldots, \ell_{t+1}$.

Verification. To (publicly) verify a handover, an external verifier checks the opening and Σ-protocol proofs from the different nodes. From there, a Lagrange consistency check ensures the resulting commitments $\{C_j\}_j$ properly reconstruct the original commitment C.

Theorem 1. *Let* $\mathsf{PoPCS} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open}, \mathsf{VerifyOpen})$ *be a PoPCS as per definition 2, and* $\mathsf{PKE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *a PKE scheme associated with it. Then the construction in section 4.1.1 is a DPVSS scheme satisfying secrecy and integrity as defined in definition 1, in the network model defined in section 2.1 against polynomial-time mobile adversaries.*

Proof (Sketch). We prove secrecy and integrity under a polynomial-time mobile adversary $\mathcal{A}$ defined in section 2.1, tolerating up to $t < n/2$ corruptions within any single epoch committee.

Secrecy. We show that $\mathcal{A}$ corrupting at most t nodes in committee $\mathcal{C}^i$ and at most t nodes in the next committee $\mathcal{C}^{i+1}$ during a handover learns no information about the underlying secrets (s, r). By the threshold properties of the secret sharing scheme, the secret polynomials f_i and g_i used to track the state across epochs are of degree t. Consequently, any subset of t or fewer evaluations (s_j, r_j) contains no information about the constant terms $s = f_i(0)$ and $r = g_i(0)$. For the remaining $n - t$ honest nodes, their reshares are sent over a public channel using the $\mathsf{PoPCS.Open}$ algorithm. The adversary's view of these messages consists of: (1) polynomial commitments $C_{\bar{f}_j}, C_{\bar{g}_j}$, (2) Σ-protocol proofs $\pi_{j,\star}$, and (3) ciphertexts $\bar{\mathsf{ct}}_{1,\alpha}, \bar{\mathsf{ct}}_{2,\alpha}$. By using a hybrid reduction, we can show that this view leaks no information. Specifically, by the hiding property of PoPCS, $C_{\bar{f}_j}$ and $C_{\bar{g}_j}$ are computationally indistinguishable from commitments to random polynomials. By the perfect zero-knowledge property of the Σ-protocol, the proofs $\pi_{j,\star}$ can be perfectly simulated. Finally, by the semantic security of PKE, the

ciphertexts for honest nodes in $\mathcal{C}^{i+1}$ are computationally indistinguishable from encryptions of random elements. As a result, the joint view of any two successive epochs can be perfectly simulated, implying secrecy according to definition 1.

Integrity. We show that $\mathcal{A}$ cannot distribute invalid shares to the incoming committee without detection, ensuring that the reconstructed state always matches the canonical state commitment C. First, a malicious node $j \in \mathcal{C}^i$ attempting to distribute bad reshares to the next committee will be detected by any external verifier running $\mathsf{VerifyOpen}$. By the evaluation binding of PoPCS, node j cannot open a commitment to two different values except with negligible probability. Therefore, the values decrypted by the incoming nodes are guaranteed to be the valid polynomial evaluations $s_{j,k} = \bar{f}_j(\alpha_k)$ and $r_{j,k} = \bar{g}_j(\alpha_k)$ matching $C_{\bar{f}_j}$ and $C_{\bar{g}_j}$. Also, by the knowledge soundness of the Σ-protocol proof $\pi_{j,\star}$, a valid transcript implies that C_j indeed commits to the constant terms $\bar{f}_j(0)$ and $\bar{g}_j(0)$ encrypted in the ciphertexts. Finally, the public verification phase executes a Lagrange consistency check. Because the underlying commitment is homomorphic and binding, this check passes if and only if the constant terms recombine to the exact values in the canonical commitment $C = [s]\mathsf{U}_1 + [r]\mathsf{U}_2$. Any bad share breaks this invariant and fails verification, ensuring that integrity holds under full corruption.

4.2 Stateless handovers

Verifying k consecutive state handovers naively requires $\mathcal{O}(k \cdot n)$ work which is linear in k. We bypass this by anchoring our DPVSS to the horizontal computation axis, rendering handover verification *stateless*. Because a handover reshapes secret shares without changing the underlying state, the canonical state commitment C (output by the most recent MPC execution) can serve as a global invariant. During the transition from committee $\mathcal{C}^i$ to $\mathcal{C}^{i+1}$, the outgoing nodes use PoPCS and Lagrange consistency to prove their ciphertexts correctly encrypt shares of the state inside C. Crucially, by the knowledge soundness and binding of the PoPCS, $\mathcal{C}^i$ cannot generate a valid proof without having valid shares of C. The proof for epoch i is therefore self-certifying, guaranteeing the integrity of the state held by $\mathcal{C}^{i+1}$ independent of the prior handovers. This decoupling achieves succinctness as an external verifier only needs to check the latest transition, which takes $\mathcal{O}(n)$ time.

5 The horizontal axis: IV-MPC

Recall that the IV-MPC proofs track the step-by-step evolution of the secret state in the MPC protocol. This requires the active MPC committee to collaboratively generate a proof for the state transition function F, which must then be efficiently accumulated. To achieve this, we take the framework of Compressed Σ-Protocols (CSP) [AC20] as our starting point. CSP provides a framework for efficiently proving knowledge of a vector $\mathbf{x} \in \mathbb{F}_q^n$ committed via a Pedersen vector

commitment $C = \langle \mathbf{x}, \mathbf{G} \rangle + [\gamma]H$, while allowing the verifier to check linear relations $\mathbb{L}(\mathbf{x}) = y$ for any linear form $\mathbb{L} : \mathbb{F}_q^n \to \mathbb{F}_q$. While CSP achieves logarithmic proof sizes, it presents two major bottlenecks in a recursive setting. First, verifying a CSP proof requires linear-time computation in the size of the committed vector. In a recursive setting, this verification must be implemented inside the recursion circuit, causing the (linear-time) verifier to translate into a large and growing circuit. Second, the prover must perform superlinear $\mathcal{O}(n \log n)$ work which is prohibitive in a distributed MPC setting.

Challenges in making CSP recursion-friendly. A naive implementation of recursive proofs requires the recursion circuit to fully verify the proof from previous step. If the underlying proof system has a linear-time verifier (as is the case with CSP), the circuit size multiplies at each layer causing an exponential blow-up. To circumvent this, one can take the more recent approach of constructing IVC based on *NARKs with accumulation schemes* [BGH19, BCMS20, BCL$^+$21]. Instead of performing the full linear time verification inside the recursion circuit, an accumulation scheme allows the prover to efficiently *accumulate* the expensive verification checks of multiple proofs into a single running instance. Consequently, the recursive circuit only needs to perform sublinear checks. The heavy (i.e., linear-time) verification is deferred and evaluated only once at the end, thus making verification cost independent of number of epochs.

While this solves the recursive bottleneck, applying it to CSP introduces a challenge: folded commitment generators can be expressed and accumulated as an inner product relation, but the folded linear forms rely on unstructured maps that do not seem to have an accumulation scheme. Our key observation is that these linear maps are defined entirely by the wiring of the underlying circuit. Therefore, they can be preprocessed once: the prover commits to the corresponding linear forms and later prove their correct openings. Building on this observation, we first recall the CSP approach for proving circuit satisfiability, and then explain how we modify it to address both the lack of accumulation friendliness and the non-linear prover issue.

CSP for circuit satisfiability. Let F be an arithmetic circuit over $\mathbb{F}_q$ with input wires $\mathbf{x} \in \mathbb{F}_q^{\ell_{\mathsf{in}}}$, output wires $\mathbf{y} \in \mathbb{F}_q^{\ell_{\mathsf{out}}}$, and ℓ_{mul} multiplication gates with input-output wires $\{(l_i, r_i, o_i)\}_{i=1}^{\ell_{\mathsf{mul}}}$. The circuit satisfiability relation is

$$\mathcal{R}_{cs} := \{(F; \mathbf{x}, \mathbf{y}) : F(\mathbf{x}) = \mathbf{y}\}.$$

CSP reduces $\mathcal{R}_{cs}$ to linear-form openings using a *linearization technique* as follows:

1. For each multiplication gate, define affine forms u_i, v_i over inputs and prior outputs, such that $l_i = u_i(\mathbf{x}, o_1, \ldots, o_{i-1})$ and $r_i = v_i(\mathbf{x}, o_1, \ldots, o_{i-1})$.
2. Define the output affine form w such that $\mathbf{y} = w(\mathbf{x}, o_1, \ldots, o_{\ell_{\mathsf{mul}}})$.
3. Encode left and right gate vectors $(l_1, \ldots, l_{\ell_{\mathsf{mul}}})$, $(r_1, \ldots, r_{\ell_{\mathsf{mul}}})$ into low-degree polynomials $L(X), R(X) \in \mathbb{F}_q[X]_{\leq \ell_{\mathsf{mul}}}$, and set $O(X) := L(X)R(X)$.

4. Commit to the vector

$$\boldsymbol{\mathcal{X}} := (\mathbf{x}, L(0), R(0), O(0), \ldots, O(2\ell_{\mathsf{mul}}), \mathbf{y}), \tag{1}$$

   which implicitly defines all gate values.
5. The verifier selects a random challenge $c \leftarrow\!\!\$\ \mathbb{F}_q$ and the prover returns $l = L(c)$, $r = R(c)$, and $o = O(c)$, together with a CSP proof which shows that $\mathbb{L}_i(\boldsymbol{\mathcal{X}}) = 0$ for $i = 1, \ldots, 4$, where

$$\mathbb{L}_1 := F(\mathbf{x}) - \mathbf{y}, \quad \mathbb{L}_2 := L(c) - l, \quad \mathbb{L}_3 := R(c) - r, \quad \mathbb{L}_4 := O(c) - o$$

5.1 Our linear-time CSP+Sumcheck protocol

Our goal is to replace heavy parts of CSP with sumcheck and further make the proof verification accumulation-friendly. Towards this goal, we begin with switching from univariate to multilinear setting. This allows us to define our polynomials over a boolean hypercube, enabling the use of sumcheck protocol. Instead of the witness vector in eq. (1), the prover commits to

$$\boldsymbol{\mathcal{X}} = (\mathbf{x}, o_1, \ldots, o_{\ell_{\mathsf{mul}}}, \mathbf{y})^6,$$

using a perfectly hiding commitment. Let $\mu = \log \ell_{\mathsf{mul}}$[7]. The prover proves that

- $l_i \cdot r_i = o_i$ for all $i \in [\ell_{\mathsf{mul}}]$,
- $M_F \boldsymbol{\mathcal{X}} = \mathbf{0}$, where $M_F \in \mathbb{F}_q^{\ell_{\mathsf{out}} \times |\boldsymbol{\mathcal{X}}|}$ is the matrix defining the wiring of the circuit output gates w.r.t. $\boldsymbol{\mathcal{X}}$.

We now show how to reduce the first type of constraints to the second type, and then explain how to handle the latter which ultimately amount to linear openings over χ. Define three functions $L, R, O : \{0,1\}^\mu \to \mathbb{F}_q$ mapping gate indices to wire values. Let $\tilde{L}, \tilde{R}, \tilde{O}$ denote their multilinear extensions, respectively. The first constraint holds if and only if for all $y \in \{0,1\}^\mu$,

$$\tilde{L}(y)\tilde{R}(y) - \tilde{O}(y) = 0.$$

To encode the multiplication constraints as a sumcheck instance, we define the low-degree polynomial $Q(\cdot)$ over the boolean hypercube:

$$Q(X) = \sum_{y \in \{0,1\}^\mu} \big(\tilde{L}(y)\tilde{R}(y) - \tilde{O}(y)\big) \cdot \tilde{eq}(X, y),$$

where $\tilde{eq}(X, y) = \prod_{k=1}^{\mu} \big(X_k \cdot y_k + (1 - X_k) \cdot (1 - y_k)\big)$. Observe that $Q(\cdot)$ is a zero polynomial (i.e., it vanishes at all points) if and only if $\tilde{L}(\cdot)\tilde{R}(\cdot) - \tilde{O}(\cdot)$ vanishes

[6] Note that this is half the size of the original when measuring multiplication gates. We omit the details of the commitment generation here, deferring a full treatment to the next section where they are integrated into the MPC protocol.

[7] We assume both ℓ_{mul} and $|\boldsymbol{\mathcal{X}}|$ are powers of two; otherwise, we pad the vectors with dummy gates or zero value.

on the μ-dimension Boolean hypercube. Therefore, it suffices to check whether $Q(t) = 0$ for a uniformly random $t \leftarrow\!\$\ \mathbb{F}_q^{\mu}$[8]. Using the sumcheck protocol, the verifier reduces this to checking:

$$e_y \stackrel{?}{=} \big(\tilde{L}(t_y) \cdot \tilde{R}(t_y) - \tilde{O}(t_y)\big) \cdot \tilde{eq}(t, t_y),$$

for a randomly sampled $t_y \leftarrow\!\$\ \mathbb{F}_q^{\mu}$. To compute this, the verifier needs the evaluations $\tilde{L}(t_y)$, $\tilde{R}(t_y)$, $\tilde{O}(t_y)$, and $\tilde{eq}(t, t_y)$. While $\tilde{eq}(t, t_y)$ can be computed by the verifier in $\mathcal{O}(\mu)$ time, evaluating the wire polynomials is expensive and therefore requires running an additional protocol. This means the prover will send three claims to the verifier asserting that $l_y = \tilde{L}(t_y)$, $r_y = \tilde{R}(t_y)$, and $o_y = \tilde{O}(t_y)$. The verifier then checks whether $e_y \stackrel{?}{=} \big(l_y \cdot r_y - o_y\big) \cdot \tilde{eq}(t, t_y)$. We now describe the protocol for verifying these claims. We focus on the claim $l_y = \tilde{L}(t_y)$; the arguments for $\tilde{R}$ and $\tilde{O}$ are analogous. Considering the evaluation of $\tilde{L}(\cdot)$ in the multilinear Lagrange basis, we have $\tilde{L}(t_y) = \sum_{j \in \{0,1\}^{\mu}} \tilde{eq}(j, t_y) \cdot l_j$. Equivalently,

$$\tilde{L}(t_y) = \langle \mathbf{E}(t_y), \mathbf{l} \rangle = \mathbf{E}(t_y)^{\top} \mathbf{l},$$

where $\mathbf{E}(t_y) = (\tilde{eq}(j, t_y))_{j \in \{0,1\}^{\mu}}$, and $\mathbf{l} = (l_1, \ldots, l_{\ell_{\mathsf{mul}}})$. On the other hand, by the definition of $\boldsymbol{\mathcal{X}}$, there exists a matrix $M_l \in \mathbb{F}_q^{\ell_{\mathsf{mul}} \times |\boldsymbol{\mathcal{X}}|}$ such that $\mathbf{l} = M_l \boldsymbol{\mathcal{X}}$. It follows that the evaluation of $\tilde{L}$ at t_y can be written as

$$\tilde{L}(t_y) = \mathbf{E}(t_y)^{\top} M_l \boldsymbol{\mathcal{X}}.$$

In the language of CSP theory, this corresponds to opening $\boldsymbol{\mathcal{X}}$ with respect to the linear form $\mathbb{L}_l := \mathbf{E}(t_y)^{\top} M_l \in \mathbb{F}^{|\boldsymbol{\mathcal{X}}|}$. Since computing $\mathbb{L}_l$ directly is expensive for the verifier, the prover instead commits to $\mathbb{L}_l$, and the verifier checks its correctness probabilistically. In fact, the verifier needs the folded form of $\mathbb{L}_l$. Thus, the prover commits to the folded form and proves that the folding is performed correct. Concretely, for $\mu_x := \log |\boldsymbol{\mathcal{X}}|$ and the sequence of CSP challenges $(e_1, \ldots, e_{\mu_x})$, the folded linear form $\mathbb{L}_l^f$ is computed as

$$\mathbb{L}_l^f = \langle \mathbb{L}_l, \mathbf{e} \rangle = \langle \mathbf{E}(t_y)^{\top} M_l, \mathbf{e} \rangle,$$

where

$$\mathbf{e} := (e_1 e_2 \cdots e_{\mu_x},\ e_1 \cdots e_{\mu_x - 1},\ \ldots,\ 1) \in \mathbb{F}^{|\boldsymbol{\mathcal{X}}|}.$$

Let $\tilde{M}_l$, and $\tilde{\mathbf{e}}$ denote the multilinear extensions of M_l and $\mathbf{e}$, respectively. We then have

$$\mathbb{L}_l^f = \sum_{\bar{x} \in \{0,1\}^{\mu}} \sum_{\bar{y} \in \{0,1\}^{\mu_x}} \tilde{eq}(\bar{x}, t_y) \tilde{M}_l(\bar{x}, \bar{y})\, \tilde{\mathbf{e}}(\bar{y}).$$

This sum can be delegated to the prover via another sumcheck protocol. At the end of the sumcheck, the verifier evaluates the claimed polynomial at

[8] By the Schwartz-Zippel lemma, this incurs a soundness error of at most $3\mu/q$.

a random point (s_x, s_y). The values $\tilde{eq}(s_x, t_y)$ and $\tilde{\mathbf{e}}(s_y)$ can be computed by the verifier in logarithmic time, but evaluating $\tilde{M}_l(s_x, s_y)$ is still costly for the verifier. To address this, we observe that while M_l is dimensionally large, it is highly sparse[9]. We exploit this sparsity using the SPARK compiler [Set20] (see appendix D for details), which allows us to commit to M_l with costs that scale proportionally to its sparsity. In appendix D.1, we show how to instantiate the SPARK compiler via an accumulation-friendly multilinear PCS based on CSP theory. With this instantiation, the overall design yields a linear-time prover and a verifier amenable to accumulation, thus enabling efficient recursion for the entire protocol.

5.2 Generalization to the distributed setting

The protocol described above assumes a single prover. In our setting, the prover is distributed across multiple nodes jointly running the proof generation while preserving state privacy against up to $t < n/2$ corruptions. We next describe how to adapt the protocol to this distributed setting.

5.2.1 Distributed sumcheck. Consider a set of computing parties $\mathtt{P}_1, \ldots, \mathtt{P}_n$, and a set of clients $\mathtt{C}_1, \ldots, \mathtt{C}_m$ providing inputs. Given a secret-shared input vector $\mathbf{x} \in \mathbb{F}_q^{\ell_{\mathsf{in}}}$ provided by the clients, the goal is for the computing parties to collaboratively generate a CSP proof (as described in section 5.1) demonstrating that $F(\mathbf{x}) = \mathbf{y} \in \mathbb{F}_q^{\ell_{\mathsf{out}}}$. To formalize the statement, we fix the following group generators:

- $\mathbf{G}^{\mathsf{x}} := (G_1^{\mathsf{x}}, \ldots, G_{\ell_{\mathsf{in}}}^{\mathsf{x}})$, used to commit to (shares of) input wires $\mathbf{x} \in \mathbb{F}_q^{\ell_{\mathsf{in}}}$.
- $\mathbf{G}^{\mathsf{o}} := (G_1^{\mathsf{o}}, \ldots, G_{\ell_{\mathsf{mul}}}^{\mathsf{o}})$, used to commit to (shares of) values corresponding to multiplication-gate outputs.
- $\mathbf{G}^{\mathsf{y}} := (G_1^{\mathsf{y}}, \ldots, G_{\ell_{\mathsf{out}}}^{\mathsf{y}})$, used to commit to (shares of) output wires $\mathbf{y} \in \mathbb{F}_q^{\ell_{\mathsf{out}}}$.
- H, an additional generator used to hide the commitments.

Using this notation, the NP relation for which the CSP proofs are generated is defined as

$$\mathcal{R} = \left\{ (F, C_{\mathsf{x}}, C_{\mathsf{y}}; \mathbf{x}, \mathbf{y}, \nu_x, \nu_y) \;:\; \begin{array}{r} F(\mathbf{x}) = \mathbf{y} \in \mathbb{F}_q^{\ell_{\mathsf{out}}} \\ \wedge\, C_{\mathsf{x}} = \langle \mathbf{G}^{\mathsf{x}}, \mathbf{x} \rangle + [\nu_x] H \\ \wedge\, C_{\mathsf{y}} = \langle \mathbf{G}^{\mathsf{y}}, \mathbf{y} \rangle + [\nu_y] H \end{array} \right\}.$$

The protocol proceeds as follows. First, the clients commit to their inputs using a perfectly hiding vector commitment:

$$C_{\mathsf{x}} = \langle \mathbf{G}^{\mathsf{x}}, \mathbf{x} \rangle + [\nu_x] H.$$

[9] In practice, we assume these matrices have a sparsity of $\mathcal{O}(|X|)$. This is because real-world circuits have strong locality, i.e., each multiplication gate typically combines only a small constant number of variables.

This is achieved by having each client $\mathtt{C}_i$ provide a commitment to its input x_i, together with a proof of knowledge, using the assigned generator G_i^{x} and a secret-shared blinding factor. These individual commitments are then combined into a single commitment to the input vector $\mathbf{x}$. Next, after executing the MPC protocol, the MPC nodes—holding all internal wire values of the circuit—compute two vector commitments C_{o} and C_{y}, respectively to the multiplication-gate outputs $\mathbf{o} = (o_1, \ldots, o_{\ell_{\mathsf{mul}}})$ and the circuit output wires $\mathbf{y}$:

$$C_{\mathsf{o}} = \langle \mathbf{G}^{\mathsf{o}}, \mathbf{o} \rangle + [\nu_o]H \qquad\qquad C_{\mathsf{y}} = \langle \mathbf{G}^{\mathsf{y}}, \mathbf{y} \rangle + [\nu_y]H$$

Let $\mathbf{G} := (\mathbf{G}^{\mathsf{x}}, \mathbf{G}^{\mathsf{o}}, \mathbf{G}^{\mathsf{y}})$. Combining the individual commitments yields a compact commitment C to the complete witness vector $\boldsymbol{\mathcal{X}} := (\mathbf{x}, \mathbf{o}, \mathbf{y}) \in \mathbb{F}_q^{\ell_{\mathsf{in}}+\ell_{\mathsf{mul}}+\ell_{\mathsf{out}}}$ with aggregated randomness $\nu = \nu_x + \nu_o + \nu_y$:

$$C = C_{\mathsf{x}} + C_{\mathsf{o}} + C_{\mathsf{y}} = \langle \mathbf{G}, \boldsymbol{\mathcal{X}} \rangle + [\nu]H.$$

To ensure soundness, the MPC nodes must also prove that the committed sub-vectors $\mathbf{o}$ and $\mathbf{y}$ live on disjoint sets of generators. This can be achieved via nullity checks[10]. The MPC nodes now run a distributed version of the sumcheck protocol over

$$Q(t) = \sum_{y \in \{0,1\}^{\mu}} \big(\tilde{L}(y)\tilde{R}(y) - \tilde{O}(y)\big) \cdot \tilde{eq}(t, y),$$

which ultimately reduces to opening a collection of linear forms over the witness vector $\boldsymbol{\mathcal{X}}$ via distributed CSP (see section 5.2.2). Specifically, letting

$$f(y) = \big(\tilde{L}(y)\tilde{R}(y) - \tilde{O}(y)\big) \cdot \tilde{eq}(t, y),$$

they run a distributed sumcheck on the instance $Q(t) = \sum_{y \in \{0,1\}^{\mu}} f(y)$, where the MPC nodes hold secret shares of the Lagrange coefficients for $\tilde{L}, \tilde{R}$, and $\tilde{O}$. In standard MPC, multiplying secret-shared values requires an interactive degree-reduction to prevent degree blowup. However in our setting, because the intermediate polynomials generated during each round of sumcheck are intended to be revealed to the verifier, the nodes can perform the required quadratic combinations between the coefficients of $\tilde{L}$ and $\tilde{R}$ using only local computation and then, relying on honest-majority assumption, reconstruct them at the end.

At the end of the sumcheck, the MPC nodes jointly open the evaluations of $\tilde{L}(\cdot)$, $\tilde{R}(\cdot)$, and $\tilde{O}(\cdot)$ at the verifier's challenge point t_y using the protocol presented in section 5.2.2. Crucially, each of these evaluations corresponds to a linear form over $\boldsymbol{\mathcal{X}}$ that is defined by the public circuit topology and the public challenge point. As discussed in the single-prover setting, the verifier ultimately needs the folded version of these linear forms. Because this folding depends entirely on public information, we can exploit a natural optimization: once the

[10] Taking C_{y} as an example, the MPC nodes show that there is a window of zeros in C_{y} with respect to the $\mathbf{G}^{\mathsf{x}}$ and $\mathbf{G}^{\mathsf{o}}$ generators. By amortizing the nullity checks as described in appendix C.3, this can be enforced with logarithmic communication overhead.

MPC nodes open the evaluations at t_y, their role in proof generation is complete. The remaining steps can be delegated to a single untrusted *helper prover* $\mathtt{P}^*$ operating in the clear. $\mathtt{P}^*$ receives the partially constructed proof artifacts from the MPC committee, independently executes the SPARK compiler to prove the correctness of the folded linear forms, and finally combines all proof shares into a single state-transition CSP proof as detailed in section 5.2.2. Note that because $\mathtt{P}^*$ already operates outside the MPC, it also serves as the natural candidate to execute the IV-MPC recursion circuit described in the next section.

ZK sumcheck. By itself, the standard sumcheck protocol is not zero-knowledge; the intermediate univariate polynomials and final evaluations leak information about the secret shared witness. We achieve ZK using a distributed variant of the zk-sumcheck protocol in Libra [XZZ+19] which follows the Σ-protocol paradigm: the MPC nodes collaboratively sample a random masking polynomial g of the same degree as f, send a commitment of g with the claimed sum $\sum g(y)$, and execute the sumcheck over the randomized linear combination $e \cdot f(y) + g(y)$ for a verifier challenge e. As shown in Libra, g does not need to be a fully dense μ-variate polynomial. It suffices to sample g as a sum of independent univariate polynomials: $g(y_1, \ldots, y_\mu) = g_0 + g_1(y_1) + \ldots + g_\mu(y_\mu)$, where each g_i has degree d matching the maximum partial degree of f. This optimization makes the zk-sumcheck viable within our distributed architecture. Rather than sampling $\mathcal{O}(\ell_{\mathsf{mul}})$ random elements, the MPC committee only needs to sample $\mathcal{O}(d \cdot \mu)$ random field elements to define the coefficients of g. Furthermore, $e \cdot f + g$ is combined linearly and thus the optimizations for evaluating f in MPC can be maintained. The MPC nodes simply compute their shares of f, multiply by the public scalar e, and add their share of g, preserving the non-interactive nature of the distributed prover rounds.

Remark 1 (Linear-time prover). Our distributed prover preserves the $\mathcal{O}(\ell_{\mathsf{mul}})$ computational complexity of the standard sumcheck protocol. The linear-time prover in standard sumcheck relies on dynamic programming, maintaining and reusing tables of sub-computations across protocol rounds. As shown in [LXYY25], secret sharings of these tables can also be efficiently constructed in MPC using local computation at each node.

5.2.2 Distributed linear form opening via CSP. At the conclusion of the distributed sumcheck, the MPC nodes holding secret shares of $\boldsymbol{\mathcal{X}}$ must prove two types of constraints with respect to C. First, they must prove that $\tilde{L}, \tilde{R}$, and $\tilde{O}$ evaluate correctly at the verifier's challenge point, and that the circuit wiring is satisfied. This corresponds to proving $\mathbb{L}_i(\boldsymbol{\mathcal{X}}) = y_i$ for the evaluation linear maps $\mathbb{L}_l, \mathbb{L}_r, \mathbb{L}_o$, and $\mathbb{L}_F$. Second, they must prove the generator disjointness constraints: C_{o} and C_{new} must be constructed using strictly separate generators to preserve binding. To enforce this, we define the linear maps $\mathbb{L}_O$ and $\mathbb{L}_Y$, which compute random linear combinations over the indices of $\boldsymbol{\mathcal{X}}$ corresponding to the remaining generators. Proving nullity checks $\mathbb{L}_O(\boldsymbol{\mathcal{X}}) = 0$ and $\mathbb{L}_Y(\boldsymbol{\mathcal{X}}) = 0$ guarantees that the committed sub-vectors contain strictly zeros in those positions.

Altogether, this yields a set of linear forms $\mathbb{L}_1, \ldots, \mathbb{L}_6$ with corresponding target values $y_1, \ldots, y_6$, where $y_i = 0$ for the nullity checks. To minimize communication overhead, the verifier samples a random challenge $\xi \leftarrow\!\!\$\ \mathbb{F}_q$. This allows the distributed prover to batch the multiple linear forms into a single linear form $\mathbb{L} := \sum_{i=1}^{6} \xi^{i-1}\mathbb{L}_i$, with a corresponding target value $y := \sum_{i=1}^{6} \xi^{i-1} y_i$. By the Schwartz-Zippel lemma, successfully proving $\mathbb{L}(\boldsymbol{\mathcal{X}}) = y$ guarantees the correctness of all individual linear forms with overwhelming probability.

Opening $\mathbb{L}$ using Σ-protocols results in an $\mathcal{O}(\ell_{\mathsf{mul}})$-sized proof, violating succinctness. On the other hand, executing the full CSP compression mechanism inside MPC would incur significant overhead. To balance these trade-offs, we split the protocol into two phases: the MPC nodes first execute an uncompressed Σ-protocol, and the resulting public transcript is then compressed outside MPC by the helper prover $\mathtt{P}^*$. This ensures proof succinctness without adding MPC overhead.

Phase 1: distributed uncompressed opening. MPC nodes execute a Σ-protocol over their secret shared witness $[\boldsymbol{\mathcal{X}}]$ and randomness $[\nu]$. Because the commitment and $\mathbb{L}$ are both linearly homomorphic, the nodes can generate the protocol messages using purely local computation:

1. *Commit.* The nodes generate uniformly random masking shares $[\mathbf{r}] \in \mathbb{F}_q^{|\boldsymbol{\mathcal{X}}|}$ and $[\rho] \in \mathbb{F}_q$. Then, they locally compute and broadcast partial evaluations for Σ-protocol first message, which are used to reconstruct $t = \mathbb{L}(\mathbf{r})$ and commitment $A = \langle \mathbf{G}, \mathbf{r} \rangle + [\rho]H$.
2. *Challenge.* The verifier sends a random challenge $e \leftarrow\!\!\$\ \mathbb{F}_q$[11].
3. *Response.* The nodes locally compute the masked responses over their shares:

$$[\mathbf{z}] = e[\boldsymbol{\mathcal{X}}] + [\mathbf{r}] \qquad \text{and} \qquad [\phi] = e[\nu] + [\rho]$$

   The nodes broadcast their shares to publicly reconstruct the masked witness $\mathbf{z}$ and masked randomness ϕ.

If the MPC nodes were honest, the public transcript successfully satisfies $A + eC = \langle \mathbf{G}, \mathbf{z} \rangle + [\phi]H$ and $\mathbb{L}(\mathbf{z}) = ey + t$.

Phase 2: compression by the helper prover $\mathtt{P}^*$. The above protocol is zero-knowledge with a linear proof size. Since the proof transcript $(A, t, e, \mathbf{z}, \phi)$ is public, the MPC committee's task is complete. The remaining task is to compress this proof into a logarithmic-size proof. The MPC nodes pass the transcript to $\mathtt{P}^*$ who can operate in the clear. $\mathtt{P}^*$ proceeds as follows: first, it applies the compression mechanism in CSP [AC20] to the statement $(C' := A + eC, \mathbb{L}, y' := t; \mathbf{z}, \phi)$ as detailed in appendix C.2. Next, it proves correctness of the openings of the public matrices $\tilde{M}_l, \tilde{M}_r, \tilde{M}_o$ at the challenge point (s_x, s_y) using the SPARK compiler

[11] In practice, this is derived by the MPC nodes using the Fiat-Shamir heuristic via a public hash function, e.g., $e = \mathcal{H}(C, \mathbb{L}, \xi, y, A, t)$.

in appendix D. By instantiating the SPARK compiler with the accumulation-friendly PCS from appendix D.1, the resulting proof achieves linear-time verification that supports accumulation, as required for our IVC setting.

The distributed proof generation executed by the MPC nodes is illustrated in fig. 3. The single-prover protocol run by $\mathtt{P}^*$ is illustrated in fig. 4.

5.3 Incrementally Verifiable MPC

For a given epoch transitioning from a previous state $\mathsf{st}_{\mathsf{old}}$ to a new state $\mathsf{st}_{\mathsf{new}}$, $\mathtt{P}^*$ receives the uncompressed transcript and applies the CSP compression mechanism. Next, it runs the SPARK compiler—instantiated with a CSP-based PCS—to prove the correctness of the public matrix openings. The result is a logarithmic-size step proof denoted π_{step}.

To prove the entire chain of execution, $\mathtt{P}^*$ builds an IVC proof using the framework of NARKs with accumulation schemes [BGH19, BCMS20]. For concreteness, let $\mathsf{ARG} = (\mathcal{G}, \mathcal{I}, \mathcal{P}, \mathcal{V})$ be a NARK compiled with the CSP-based PCS from appendix D.1, and let $\mathsf{AC} = (\mathrm{G}, \mathrm{I}, \mathrm{P}, \mathrm{V}, \mathrm{D})$ be an accumulation scheme for the verifier of ARG. To update the running IV-MPC proof, $\mathtt{P}^*$ calls $\mathcal{P}$ to generate a proof for the recursion circuit. This circuit takes the old and new state commitments C_{old} and C_{new}, the old and new accumulators $\mathsf{acc}_{\mathsf{old}}$ and $\mathsf{acc}_{\mathsf{new}}$, the input commitment C_{x}, the signature σ, the step proof π_{step}, and the previous IV-MPC proof Π_{old}. It then enforces the following constraints:

1. *Sublinear verification.* It verifies the sublinear checks for both the current step proof π_{step} and the previous IV-MPC proof Π_{old}.
2. *Input authentication.* It verifies the authority's signature σ on the input commitment C_{x} against pk_{auth}.
3. *Accumulator update.* It verifies that $\mathsf{acc}_{\mathsf{new}}$ is the correct result of folding $\mathsf{acc}_{\mathsf{old}}$ with the expensive, linear-time PCS opening claims extracted from π_{step} and Π_{old}.

Upon successful execution of $\mathcal{P}$, $\mathtt{P}^*$ obtains a new IV-MPC proof Π_{new} attesting to the correct evaluation of the recursive circuit. The final output for the epoch is the updated state commitment C_{new}, the new accumulator $\mathsf{acc}_{\mathsf{new}}$, and the proof Π_{new}. See fig. 5 for an illustration.

5.4 Security analysis

Since our design decouples the distributed computation from the single-prover recursive proof generation, we adopt a modular approach for the security analysis. We focus our formal treatment on the novel distributed proof generation phase. This is because $\mathtt{P}^*$ operates exclusively on public data, so the IVC layer requires no zero-knowledge properties. Its knowledge soundness also follows straightforwardly from the knowledge soundness of the underlying NARK and its accumulation scheme as shown in [BCMS20].

We define security properties for the distributed phase with respect to a distributed argument system $\Pi = (\mathcal{G}, \mathcal{I}, \overrightarrow{\mathcal{P}}, \mathcal{V})$ where $\overrightarrow{\mathcal{P}} = (\mathtt{P}_1, \ldots, \mathtt{P}_n)$ denotes

the interactive protocol executed by the n-party MPC committee. We adapt the definitions from [OB22] to formalize t-zero-knowledge and knowledge soundness for Π in the random oracle model.

Definition 6 (t-zero-knowledge). *Let $\mathcal{U}(\lambda)$ be the uniform distribution over oracle functions $H : \{0,1\}^* \to \{0,1\}^\lambda$. A distributed argument system $\Pi = (\mathcal{G}, \mathcal{I}, \overrightarrow{\mathcal{P}}, \mathcal{V})$ satisfies t-zero-knowledge if for all expected polynomial-time adversaries $\mathcal{A}$ controlling a subset of $k \leq t$ MPC nodes $\{\mathtt{P}_{i_1}, \ldots, \mathtt{P}_{i_k}\}$, there exists an expected polynomial-time simulator $\mathcal{S}$ such that for any statement $\mathbb{x}$ and witnesses $\overrightarrow{\mathbb{w}}$ satisfying $\mathcal{R}(\mathbb{x}, \overrightarrow{\mathbb{w}}) = 1$, the following advantage is negligible in λ for every efficient distinguisher D:*

$$\left| \Pr\left[D^{H[\mu]}(\mathsf{tr}) = 1 : \begin{array}{c} H \leftarrow\!\$\, \mathcal{U}(\lambda) \\ \mathsf{pp} \leftarrow \mathcal{G}^H(1^\lambda) \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{I}^H(\mathsf{pp}, \mathcal{R}) \\ (\mathsf{tr}, \mu) \leftarrow \mathcal{S}^H(\mathsf{pp}, \mathbb{x}, \\ w_{i_1}, \ldots, \mathbb{w}_{i_k}) \end{array} \right] - \Pr\left[D^H(\mathsf{tr}) = 1 : \begin{array}{c} H \leftarrow\!\$\, \mathcal{U}(\lambda) \\ \mathsf{pp} \leftarrow \mathcal{G}^H(1^\lambda) \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{I}^H(\mathsf{pp}, \mathcal{R}) \\ \mathsf{tr} \leftarrow \mathsf{View}_{\mathcal{A}}^H[\mathbb{x}, \overrightarrow{\mathbb{w}}] \end{array} \right] \right|$$

where tr is the proof transcript, $\mathsf{View}_{\mathcal{A}}^H[\mathbb{x}, \overrightarrow{\mathbb{w}}]$ denotes the view of $\mathcal{A}$ during the execution of $\overrightarrow{\mathcal{P}}$, $\mathbb{w}_{i_j}$ represents the witness shares held by corrupted nodes, and $H[\mu]$ is the random oracle H reprogrammed by $\mathcal{S}$ at the points specified by μ.

Definition 7 (Knowledge soundness). *Let $\mathcal{U}(\lambda)$ be the uniform distribution over oracle functions $H : \{0,1\}^* \to \{0,1\}^\lambda$. A distributed argument system $\Pi = (\mathcal{G}, \mathcal{I}, \overrightarrow{\mathcal{P}}, \mathcal{V})$ satisfies knowledge soundness if for any statement $\mathbb{x}$ and any collection of PPT algorithms $\overrightarrow{\mathcal{P}}^* = (\mathtt{P}_1^*, \ldots, \mathtt{P}_n^*)$, there exists an expected polynomial-time extractor $\mathcal{E}_{mpc}$ such that:*

$$\Pr\left[\mathcal{R}(\mathbb{x}, \overrightarrow{\mathbb{w}}) = 1 : \begin{array}{c} H \leftarrow\!\$\, \mathcal{U}(\lambda) \\ \mathsf{pp} \leftarrow \mathcal{G}^H(1^\lambda) \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{I}^H(\mathsf{pp}, \mathcal{R}) \\ \overrightarrow{\mathbb{w}} \leftarrow \mathcal{E}_{mpc}^{H, \overrightarrow{\mathcal{P}}^{*H}}(\mathsf{pp}, \mathbb{x}) \end{array} \right] \geq \Pr\left[\mathcal{V}^H(\mathsf{vk}, \mathbb{x}, \mathsf{tr}) = 1 : \begin{array}{c} H \leftarrow\!\$\, \mathcal{U}(\lambda) \\ \mathsf{pp} \leftarrow \mathcal{G}^H(1^\lambda) \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{I}^H(\mathsf{pp}, \mathcal{R}) \\ \mathsf{tr} \leftarrow \overrightarrow{\mathcal{P}}^{*H}(\mathsf{pp}, \mathbb{x}) \end{array} \right] - \mathsf{negl}(\lambda)$$

where $\mathcal{E}_{mpc}^{H, \overrightarrow{\mathcal{P}}^{*H}}$ denotes that the extractor has oracle access to H, and can rewind $\overrightarrow{\mathcal{P}}^*$ and reprogramming H each time.

Theorem 2. *Assuming $t < n/2$, the distributed argument system presented in fig. 3 satisfies t-zero-knowledge and knowledge soundness in the random oracle model.*

Proof (Sketch). In order to prove t-zero-knowledge, we construct an efficient simulator $\mathcal{S}$ as follows. During the ZK-sumcheck and distributed Σ-protocol rounds, honest nodes mask their execution states using uniformly random polynomials and blinding scalars (ν_x, ν_o, ν_y) inside their Pedersen commitments. Because the adversary $\mathcal{A}$ controls $k \leq t < n/2$ nodes, the honest majority guarantees that the joint masking vectors are uniformly random and independent from $\mathcal{A}$'s perspective. The simulator $\mathcal{S}$ proceeds as follows:

1. It runs the setup algorithm to obtain pp, pk, and vk.
2. For the Σ-protocol part of the transcript, $\mathcal{S}$ runs the Σ-protocol simulator: it chooses a random challenge $e \leftarrow\!\!\$\ \{0,1\}^\lambda$ and a random response vector $(\mathbf{z}, \phi)$, computes the commitment $A := \langle \mathbf{G}, \mathbf{z} \rangle + \phi H - eC$, and programs the random oracle by setting $\mu(\mathsf{pp}, \mathbb{x}, A) = e$.
3. For the sumcheck protocol part of the transcript, the simulator simply runs the standard simulator of the underlying ZK-sumcheck protocol to generate an indistinguishable polynomial evaluation transcript.

Because all elements in tr are distributed identically to a real execution, tr is perfectly indistinguishable from a real view $\mathsf{View}^H_{\mathcal{A}}[\mathbb{x}, \overrightarrow{\mathbb{w}}]$, and thus t-ZK holds.

Knowledge soundness. We show how to construct an efficient extractor $\mathcal{E}_{mpc}$ that extracts valid witnesses $\overrightarrow{\mathbb{w}}$ from a set $\overrightarrow{\mathcal{P}}^*$ outputting a valid proof transcript tr with non-negligible probability ϵ. This transcript contains the commitment A and the random oracle response $e = H(\mathsf{pp}, \mathbb{x}, A)$. The extractor $\mathcal{E}_{mpc}$ is based on the following standard rewinding argument:

1. It runs $\overrightarrow{\mathcal{P}}^*$ up to the point where the commitment A is generated and the query to H is made. It records the state.
2. It samples a new random challenge $e' \neq e$ and reprograms the random oracle such that $H[\mu'](\mathsf{pp}, \mathbb{x}, A) = e'$.
3. It rewinds every prover in $\overrightarrow{\mathcal{P}}^*$ to the recorded state and resumes the execution with e'. By standard rewinding analysis in the ROM, $\overrightarrow{\mathcal{P}}^*$ will output a second valid transcript tr' containing a new response vector $\mathbf{z}'$ in expected polynomial time $\mathcal{O}(1/\epsilon)$.

The response vectors $\mathbf{z}$ and $\mathbf{z}'$ consist of the individual shares $(\mathbf{z}_i, \mathbf{z}'_i)$ computed by each node P^*_i during the Σ-protocol phase. For each node, $\mathcal{E}_{mpc}$ uses the linear relation

$$\mathbf{z}_i - \mathbf{z}'_i = (e - e')\mathbb{w}_i \implies \mathbb{w}_i = (\mathbf{z}_i - \mathbf{z}'_i)(e - e')^{-1}$$

to extract that witness share $\mathbb{w}_i$. By extracting enough valid witness shares, $\mathcal{E}_{mpc}$ can reconstruct the full witness $\overrightarrow{\mathbb{w}}$.

The extraction above addresses only the rewinding-based extraction from the final distributed Σ-protocol. In order to show that the extracted $\overrightarrow{\mathbb{w}}$ satisfies the underlying circuit relation, the full argument additionally requires the soundness of the sumcheck and the consistency of the folded linear-form checks. Specifically, the witness shares extracted via special soundness correspond to a witness that is consistent with the aggregated linear-form evaluation produced at the end of the sumcheck. The statistical soundness of the sumcheck then guarantees that this witness satisfies the underlying circuit relation except with negligible probability, with the error bound following from the Schwartz-Zippel lemma. □

Acknowledgment. Anders Kallesøe and Hamidreza Khoshakhlagh were supported by the European Research Council (ERC) under the EU's Horizon 2020 research and innovation programme (Grant Agreement No. 101124977, DECRYPSYS). Hamidreza Khoshakhlagh was additionally supported by the Danish Innovation Fund (Grant No. 4298-00014A).

MPC node $\mathtt{P}_i$ (holding shares $[\cdot]_i$)		**Verifier**
	Setup	
Public • $\mathbf{G}^{\mathsf{old}}, \mathbf{G}^{\mathsf{new}}$: generators for prev/new state • $\mathbf{G}^{\mathsf{x}}, \mathbf{G}^{\mathsf{o}}$: generators for inputs/gates • H: generator for hiding commitments • $C_{\mathsf{old}}, C_{\mathsf{x}}$: commitments to prev state / inputs		
Input • $[\mathbf{x}]_i, [\mathsf{st}_{\mathsf{old}}]_i$: shares of inputs / prev state • $[\mathsf{st}_{\mathsf{new}}]_i = F([\mathsf{st}_{\mathsf{old}}]_i, [\mathbf{x}]_i)$ • $[\mathbf{o}]_i$: shares of mult. gate output wires • $[\nu_x]_i, [\nu_o]_i, [\nu_y]_i$: shares of blindings		
Computation • $[C_{\mathsf{o}}]_i = \langle \mathbf{G}^{\mathsf{o}}, [\mathbf{o}]_i \rangle + [\nu_o]_i H$ • $[C_{\mathsf{new}}]_i = \langle \mathbf{G}^{\mathsf{new}}, [\mathsf{st}_{\mathsf{new}}]_i \rangle + [\nu_y]_i H$		
Reconstruct $C_{\mathsf{o}}, C_{\mathsf{new}}$	$\xrightarrow{C_{\mathsf{o}}, C_{\mathsf{new}}}$	
Define $[\boldsymbol{\mathcal{X}}]_i = ([\mathsf{st}_{\mathsf{old}}]_i, [\mathbf{x}]_i, [\mathbf{o}]_i, [\mathsf{st}_{\mathsf{new}}]_i)$		Public $C = C_{\mathsf{old}} + C_{\mathsf{x}} + C_{\mathsf{o}} + C_{\mathsf{new}}$
	Sumcheck protocol	
$[f(\mathbf{X})]_i = \big([\tilde{L}(\mathbf{X})]_i [\tilde{R}(\mathbf{X})]_i - [\tilde{O}(\mathbf{X})]_i\big) \cdot \tilde{eq}(t, \mathbf{X})$		
. . . . .		
	$\xleftarrow{\text{Challenge } c_i}$	
$[f_i(X)]_i = \sum_{y_{i+2}, \ldots, y_\mu} [f(c_1, \ldots, c_i, X, \ldots)]_i$		
Reconstruct $f_i(X)$	$\xrightarrow{f_i(X)}$	Check $f_i(0) + f_i(1) = f_{i-1}(c_i)$
. . . . .		
End of sumcheck (evaluation at t_y) $[l_y]_i = [\tilde{L}(t_y)]_i$, $[r_y]_i = [\tilde{R}(t_y)]_i$, $[o_y]_i = [\tilde{O}(t_y)]_i$		
Reconstruct l_y, r_y, o_y	$\xrightarrow{l_y, r_y, o_y}$	Check $l_y \cdot r_y = o_y$
$\rightarrow$ *Delegate proof of correctness for folded linear forms to* $\mathtt{P}^*$		
	Distributed Σ-protocol	
Define linear maps $\mathbb{L}_l, \mathbb{L}_r, \mathbb{L}_o, \mathbb{L}_F, \mathbb{L}_O, \mathbb{L}_Y$	$\xleftarrow{\text{Challenge } \xi}$	
Define $\mathbb{L} := \mathbb{L}_l + \xi \mathbb{L}_r + \xi^2 \mathbb{L}_o + \xi^3 \mathbb{L}_F + \xi^4 \mathbb{L}_O + \xi^5 \mathbb{L}_Y$		
Let $\mathbb{L}(\boldsymbol{\mathcal{X}}) = y$		
Generate joint randomness $[\mathbf{r}]_i \in \mathbb{F}_q^{\lvert\boldsymbol{\mathcal{X}}\rvert}$, $[\rho]_i \in \mathbb{F}_q$		
$[t_l]_i = \mathbb{L}([\mathbf{r}]_i)$		
$[A]_i = \langle \mathbf{G}, [\mathbf{r}]_i \rangle + [\rho]_i H$		
Reconstruct t_l, A	$\xrightarrow{t_l,\ A}$	
	$\xleftarrow{\text{Challenge } e}$	
$[\mathbf{z}]_i = e[\boldsymbol{\mathcal{X}}]_i + [\mathbf{r}]_i$		
$[\phi]_i = e[\nu]_i + [\rho]_i$		
Reconstruct $\mathbf{z}, \phi$	$\xrightarrow{\mathbf{z},\ \phi}$	
$\rightarrow$ *Delegate proof compression to* $\mathtt{P}^*$		Check (compressed version of) $A + eC = \langle \mathbf{G}, \mathbf{z} \rangle + [\phi]H$ $\wedge \quad \mathbb{L}(\mathbf{z}) = ey + t$

Fig. 3: Distributed proof generation before compression (run by MPC nodes).

Helper prover $\mathtt{P}^*$	**Verifier**

Public proof compression & SPARK compilation

Input: Uncompressed proof transcript

- $C, \mathbb{L}, y$
- $A, t, e, \mathbf{z}, \phi$

Verify correctness of MPC output

- $A + eC \stackrel{?}{=} \langle \mathbf{G}, \mathbf{z} \rangle + [\phi]H$
- $\mathbb{L}(\mathbf{z}) \stackrel{?}{=} ey + t$

Apply CSP compression to $\mathbf{z}$

$\pi_{csp} \leftarrow \text{CSP.Prove}\big(C' = A + eC, \mathbb{L}, y' = ey + t; \mathbf{z}, \phi\big)$

Prove folded public matrix claims via SPARK

$\pi_{spark} \leftarrow \text{SPARK.Prove}(\tilde{M}_l, \tilde{M}_r, \tilde{M}_o, \tilde{M}_F, (s_x, s_y))$

Set $\pi_{step} = (\pi_{csp}, \pi_{spark})$ $\xrightarrow{\pi_{step}}$

Verify π_{csp}

$\wedge$

Verify π_{spark}

Fig. 4: CSP compression and SPARK compilation (run by $\mathtt{P}^*$).

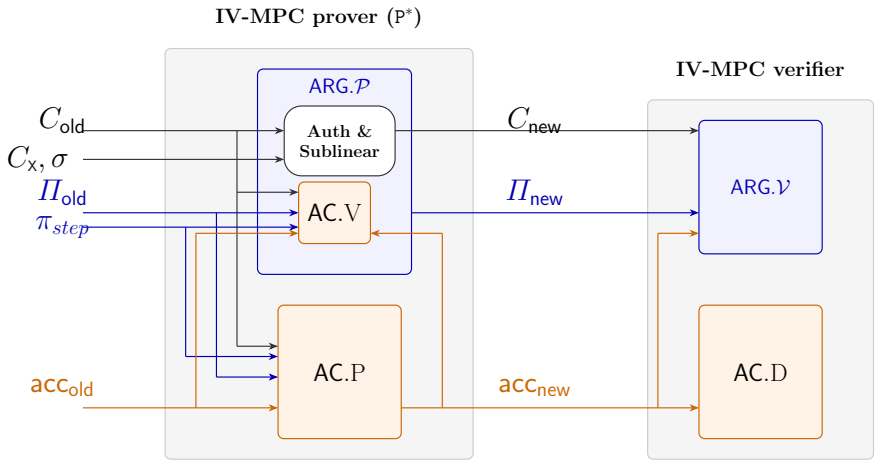

Fig. 5: Our IV-MPC from NARK and accumulation schemes. The recursion circuit verifies the input authentication, sublinear proof checks and accumulation verifier. The expensive linear-time claims are folded into the running accumulator (AC.P).

References

AC20. Thomas Attema and Ronald Cramer. Compressed Σ-protocol theory and practical application to plug & play secure algorithmics. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020, Part III*, volume 12172 of *Lecture Notes in Computer Science*, pages 513–543, Santa Barbara, CA, USA, August 17–21, 2020. Springer, Cham, Switzerland.

BBB+18. Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy*, pages 315–334, San Francisco, CA, USA, May 21–23, 2018. IEEE Computer Society Press.

BCB+24. Hendrik Ballhausen, Stefanie Corradini, Claus Belka, Dan Bogdanov, Luca Boldrini, Francesco Bono, Christian Goelz, Guillaume Landry, Giulia Panza, Katia Parodi, Riivo Talviste, Huong Elena Tran, Maria Antonietta Gambacorta, and Sebastian Marschner. Privacy-friendly evaluation of patient data with secure multiparty computation in a European pilot study. *npj Digital Medicine*, 7(1):280, October 2024.

BCD+09. Peter Bogetoft, Dan Lund Christensen, Ivan Damgård, Martin Geisler, Thomas Jakobsen, Mikkel Krøigaard, Janus Dam Nielsen, Jesper Buus Nielsen, Kurt Nielsen, Jakob Pagter, Michael I. Schwartzbach, and Tomas Toft. Secure multiparty computation goes live. In Roger Dingledine and Philippe Golle, editors, *FC 2009: 13th International Conference on Financial Cryptography and Data Security*, volume 5628 of *Lecture Notes in Computer Science*, pages 325–343, Accra Beach, Barbados, February 23–26, 2009. Springer Berlin Heidelberg, Germany.

BCL+21. Benedikt Bünz, Alessandro Chiesa, William Lin, Pratyush Mishra, and Nicholas Spooner. Proof-carrying data without succinct arguments. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021, Part I*, volume 12825 of *Lecture Notes in Computer Science*, pages 681–710, Virtual Event, August 16–20, 2021. Springer, Cham, Switzerland.

BCMS20. Benedikt Bünz, Alessandro Chiesa, Pratyush Mishra, and Nicholas Spooner. Recursive proof composition from accumulation schemes. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020: 18th Theory of Cryptography Conference, Part II*, volume 12551 of *Lecture Notes in Computer Science*, pages 1–18, Durham, NC, USA, November 16–19, 2020. Springer, Cham, Switzerland.

BDO14. Carsten Baum, Ivan Damgård, and Claudio Orlandi. Publicly auditable secure multi-party computation. In Michel Abdalla and Roberto De Prisco, editors, *SCN 14: 9th International Conference on Security in Communication Networks*, volume 8642 of *Lecture Notes in Computer Science*, pages 175–196, Amalfi, Italy, September 3–5, 2014. Springer, Cham, Switzerland.

BELO15. Joshua Baron, Karim El Defrawy, Joshua Lampkins, and Rafail Ostrovsky. Communication-optimal proactive secret sharing for dynamic groups. In Tal Malkin, Vladimir Kolesnikov, Allison Bishop Lewko, and Michalis Polychronakis, editors, *ACNS 2015: 13th International Conference on Applied Cryptography and Network Security*, volume 9092 of *Lecture Notes in Computer Science*, pages 23–41, New York, NY, USA, June 2–5, 2015. Springer, Cham, Switzerland.

BGG+20. Fabrice Benhamouda, Craig Gentry, Sergey Gorbunov, Shai Halevi, Hugo Krawczyk, Chengyu Lin, Tal Rabin, and Leonid Reyzin. Can a public blockchain keep a secret? In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020: 18th Theory of Cryptography Conference, Part I*, volume 12550 of *Lecture Notes in Computer Science*, pages 260–290, Durham, NC, USA, November 16–19, 2020. Springer, Cham, Switzerland.

BGH19. Sean Bowe, Jack Grigg, and Daira Hopwood. Halo: Recursive proof composition without a trusted setup. Cryptology ePrint Archive, Report 2019/1021, 2019.

BGW88. Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *20th Annual ACM Symposium on Theory of Computing*, pages 1–10, Chicago, IL, USA, May 2–4, 1988. ACM Press.

BKK+16. Dan Bogdanov, Liina Kamm, Baldur Kubo, Reimo Rebane, Ville Sokk, and Riivo Talviste. Students and taxes: a privacy-preserving study using secure computation. *Proceedings on Privacy Enhancing Technologies*, 2016(3):117–135, July 2016.

BKSW24. Remco Bloemen, Daniel Kales, Philipp Sippl, and Roman Walch. Large-scale MPC: Scaling private iris code uniqueness checks to millions of users. Cryptology ePrint Archive, Report 2024/705, 2024.

BOSS20. Carsten Baum, Emmanuela Orsini, Peter Scholl, and Eduardo Soria-Vazquez. Efficient constant-round MPC with identifiable abort and public verifiability. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020, Part II*, volume 12171 of *Lecture Notes in Computer Science*, pages 562–592, Santa Barbara, CA, USA, August 17–21, 2020. Springer, Cham, Switzerland.

CCD88. David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In *20th Annual ACM Symposium on Theory of Computing*, pages 11–19, Chicago, IL, USA, May 2–4, 1988. ACM Press.

CL15. Guilhem Castagnos and Fabien Laguillaumie. Linearly homomorphic encryption from DDH. In Kaisa Nyberg, editor, *Topics in Cryptology – CT-RSA 2015*, volume 9048 of *Lecture Notes in Computer Science*, pages 487–505, San Francisco, CA, USA, April 20–24, 2015. Springer, Cham, Switzerland.

CLT18. Guilhem Castagnos, Fabien Laguillaumie, and Ida Tucker. Practical fully secure unrestricted inner product functional encryption modulo p. In Thomas Peyrin and Steven Galbraith, editors, *Advances in Cryptology – ASIACRYPT 2018, Part II*, volume 11273 of *Lecture Notes in Computer Science*, pages 733–764, Brisbane, Queensland, Australia, December 2–6, 2018. Springer, Cham, Switzerland.

DGK+21. Ivan Damgård, Chaya Ganesh, Hamidreza Khoshakhlagh, Claudio Orlandi, and Luisa Siniscalchi. Balancing privacy and accountability in blockchain identity management. In Kenneth G. Paterson, editor, *Topics in Cryptology – CT-RSA 2021*, volume 12704 of *Lecture Notes in Computer Science*, pages 552–576, Virtual Event, May 17–20, 2021. Springer, Cham, Switzerland.

DXT+23. Sourav Das, Zhuolun Xiang, Alin Tomescu, Alexander Spiegelman, Benny Pinkas, and Ling Ren. Verifiable secret sharing simplified. Cryptology ePrint Archive, Report 2023/1196, 2023.

ESW24. Daniel Escudero, Yifan Song, and Wenhao Wang. Perfectly-secure multiparty computation with linear communication complexity over any modulus. In Kai-Min Chung and Yu Sasaki, editors, *Advances in Cryptology – ASIACRYPT 2024, Part VI*, volume 15489 of *Lecture Notes in Computer Science*, pages 102–132, Kolkata, India, December 9–13, 2024. Springer, Singapore, Singapore.

Fel87. Paul Feldman. A practical scheme for non-interactive verifiable secret sharing. In *28th Annual Symposium on Foundations of Computer Science*, pages 427–437, Los Angeles, CA, USA, October 12–14, 1987. IEEE Computer Society Press.

FY92. Matthew K. Franklin and Moti Yung. Communication complexity of secure computation (extended abstract). In *24th Annual ACM Symposium on Theory of Computing*, pages 699–710, Victoria, BC, Canada, May 4–6, 1992. ACM Press.

GGJ+23. Sanjam Garg, Aarushi Goel, Abhishek Jain, Guru-Vamsi Policharla, and Sruthi Sekar. zkSaaS: Zero-knowledge SNARKs as a service. In Joseph A. Calandrino and Carmela Troncoso, editors, *USENIX Security 2023: 32nd USENIX Security Symposium*, pages 4427–4444, Anaheim, CA, USA, August 9–11, 2023. USENIX Association.

GKM+22. Vipul Goyal, Abhiram Kothapalli, Elisaweta Masserova, Bryan Parno, and Yifan Song. Storing and retrieving secrets on a blockchain. In Goichiro Hanaoka, Junji Shikata, and Yohei Watanabe, editors, *PKC 2022: 25th International Conference on Theory and Practice of Public Key Cryptography, Part I*, volume 13177 of *Lecture Notes in Computer Science*, pages 252–282, Virtual Event, March 8–11, 2022. Springer, Cham, Switzerland.

GMW87. Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th Annual ACM Symposium on Theory of Computing*, pages 218–229, New York City, NY, USA, May 25–27, 1987. ACM Press.

HJKY95. Amir Herzberg, Stanislaw Jarecki, Hugo Krawczyk, and Moti Yung. Proactive secret sharing or: How to cope with perpetual leakage. In Don Coppersmith, editor, *Advances in Cryptology – CRYPTO'95*, volume 963 of *Lecture Notes in Computer Science*, pages 339–352, Santa Barbara, CA, USA, August 27–31, 1995. Springer Berlin Heidelberg, Germany.

KST22. Abhiram Kothapalli, Srinath Setty, and Ioanna Tzialla. Nova: Recursive zero-knowledge arguments from folding schemes. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology – CRYPTO 2022, Part IV*, volume 13510 of *Lecture Notes in Computer Science*, pages 359–388, Santa Barbara, CA, USA, August 15–18, 2022. Springer, Cham, Switzerland.

LXYY25. Zhe Li, Chaoping Xing, Yizhou Yao, and Chen Yuan. On the power of sumcheck in secure multiparty computation. Cryptology ePrint Archive, Report 2025/177, 2025.

LZW+25. Xuanming Liu, Zhelei Zhou, Yinghao Wang, Yanxin Pang, Jinye He, Bingsheng Zhang, Xiaohu Yang, and Jiaheng Zhang. Scalable collaborative zk-snark and its application to fully distributed proof delegation. In Lujo Bauer and Giancarlo Pellegrino, editors, *34th USENIX Security Symposium, USENIX Security 2025, Seattle, WA, USA, August 13-15, 2025*, pages 2025–2044. USENIX Association, 2025.

MZW+19. Sai Krishna Deepak Maram, Fan Zhang, Lun Wang, Andrew Low, Yupeng Zhang, Ari Juels, and Dawn Song. CHURP: Dynamic-committee proactive secret sharing. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019: 26th Conference on Computer and Communications Security*, pages 2369–2386, London, UK, November 11–15, 2019. ACM Press.

OB22. Alex Ozdemir and Dan Boneh. Experimenting with collaborative zk-SNARKs: Zero-knowledge proofs for distributed secrets. In Kevin R. B. Butler and Kurt Thomas, editors, *USENIX Security 2022: 31st USENIX Security Symposium*, pages 4291–4308, Boston, MA, USA, August 10–12, 2022. USENIX Association.

Ped92. Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *Advances in Cryptology – CRYPTO'91*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140, Santa Barbara, CA, USA, August 11–15, 1992. Springer Berlin Heidelberg, Germany.

Sch99. Berry Schoenmakers. A simple publicly verifiable secret sharing scheme and its application to electronic. In Michael J. Wiener, editor, *Advances in Cryptology – CRYPTO'99*, volume 1666 of *Lecture Notes in Computer Science*, pages 148–164, Santa Barbara, CA, USA, August 15–19, 1999. Springer Berlin Heidelberg, Germany.

Set20. Srinath Setty. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020, Part III*, volume 12172 of *Lecture Notes in Computer Science*, pages 704–737, Santa Barbara, CA, USA, August 17–21, 2020. Springer, Cham, Switzerland.

Sha79. Adi Shamir. How to share a secret. *Communications of the Association for Computing Machinery*, 22(11):612–613, November 1979.

SV15. Berry Schoenmakers and Meilof Veeningen. Universally verifiable multiparty computation from threshold homomorphic cryptosystems. In Tal Malkin, Vladimir Kolesnikov, Allison Bishop Lewko, and Michalis Polychronakis, editors, *ACNS 2015: 13th International Conference on Applied Cryptography and Network Security*, volume 9092 of *Lecture Notes in Computer Science*, pages 3–22, New York, NY, USA, June 2–5, 2015. Springer, Cham, Switzerland.

TGK+26. Eden Aldema Tshuva, Sanjam Garg, Abhiram Kothapalli, Rotem Oshman, Omkant Pandey, and Bhaskar Roberts. Collaborative incrementally verifiable computation. *IACR Cryptol. ePrint Arch.*, 2026:410, 2026.

Val08. Paul Valiant. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In Ran Canetti, editor, *TCC 2008: 5th Theory of Cryptography Conference*, volume 4948 of *Lecture Notes in Computer Science*, pages 1–18, San Francisco, CA, USA, March 19–21, 2008. Springer Berlin Heidelberg, Germany.

XZZ+19. Tiancheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song. Libra: Succinct zero-knowledge proofs with optimal prover computation. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part III*, volume 11694 of *Lecture Notes in Computer Science*, pages 733–764, Santa Barbara, CA, USA, August 18–22, 2019. Springer, Cham, Switzerland.

Yao86. Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th Annual Symposium on Foundations of Computer Science*, pages 162–167, Toronto, Ontario, Canada, October 27–29, 1986. IEEE Computer Society Press.

YLRD26. Avishay Yanai, Meital Levy, Mike Rosulek, and Hila Dahari-Garbian. gcvm: Publicly auditable MPC via garbled circuits with applications to private evm-compatible computation. *IACR Cryptol. ePrint Arch.*, 2026:170, 2026.

Supplementary Material

A Additional preliminaries

A.1 Polynomial commitments

Definition 8 (Polynomial commitments). *A polynomial commitment scheme over $\mathbb{F}$ is a tuple of PPT algorithms* $\mathsf{PCS} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open}, \mathsf{VerifyOpen})$ *defined as follows:*

- $\mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda, 1^d)$*: The setup algorithm takes as input a security parameter λ and an upper bound d, and returns a commitment key* ck*.*
- $C \leftarrow \mathsf{Commit}(\mathsf{ck}, f(X); r)$*: The polynomial commitment algorithm takes as input a commitment key* ck*, a polynomial $f \in \mathbb{F}[X]$, and randomness r, and returns a commitment C to the polynomial f.*
- $\mathsf{op} \leftarrow \mathsf{Open}(\mathsf{ck}, f, z; r)$*: The opening algorithm takes a commitment key* ck*, polynomial $f \in \mathbb{F}[X]$, evaluation point $z \in \mathbb{F}$, randomness r, and returns an opening proof* op *for $y = f(z)$.*
- $b \leftarrow \mathsf{VerifyOpen}(\mathsf{ck}, C, \mathsf{op}, z, y)$*: The verification algorithm takes as input a commitment key* ck*, a commitment C, an opening proof* op *and points $z, y \in \mathbb{F}$, and returns a bit b.*

A polynomial commitment scheme should satisfy the following properties.

Correctness. We say that $\mathsf{PCS} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open}, \mathsf{VerifyOpen})$ over $\mathbb{F}$ is correct, if for all $\lambda, d \in \mathbb{N}$, all $\mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda, 1^d)$, all polynomials $f \in \mathbb{F}[X]$ of degree $< d$, all $C \leftarrow \mathsf{Commit}(\mathsf{ck}, f)$, all $z \in \mathbb{F}$, and all $\mathsf{op} \leftarrow \mathsf{Open}(\mathsf{ck}, f, z)$, it holds that $\mathsf{VerifyOpen}(\mathsf{ck}, C, \mathsf{op}, z, f(z)) = 1$.

Binding. We say that $\mathsf{PCS} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open}, \mathsf{VerifyOpen})$ over $\mathbb{F}$ is (computationally) binding, if for all $\lambda, d \in \mathbb{N}$ and all PPT adversaries $\mathcal{A}$, it holds that

$$\Pr\left[\begin{array}{c} y_0 \neq y_1 \\ \wedge \mathsf{VerifyOpen}(\mathsf{ck}, C, \mathsf{op}_0, z, y_0) \\ \wedge \mathsf{VerifyOpen}(\mathsf{ck}, C, \mathsf{op}_1, z, y_1) \end{array} \middle| \begin{array}{r} \mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda, 1^d) \\ (C, z, y_0, y_1, \mathsf{op}_0, \mathsf{op}_1) \leftarrow \mathcal{A}(\mathsf{ck}) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

A.2 Non-Interactive Arguments of Knowledge (NARK)

Definition 9 (Non-interactive arguments of knowledge). *A (preprocessing) non-interactive argument of knowledge (NARK) is a tuple of PPT algorithms* $\mathsf{ARG} = (\mathcal{G}, \mathcal{I}, \mathcal{P}, \mathcal{V})$ *with the following syntax:*

- $\mathcal{G}(1^\lambda) \to (\mathsf{pp}, \mathsf{td})$*. On input the security parameter 1^λ, outputs public parameters* pp *and a trapdoor* td *(used by the zero-knowledge simulator; see below).*
- $\mathcal{I}(\mathsf{pp}, \mathtt{i}) \to (\mathsf{pk}, \mathsf{vk})$*. On input the public parameters* pp *and index* $\mathtt{i}$*, deterministically outputs a proving key* pk *and a verifying key* vk*.*

Game $\mathsf{ARG\text{-}KS}^{\mathcal{A},\mathcal{E}_\mathcal{A}}_{\mathsf{ARG},\mathcal{D}}(1^\lambda)$
$(\mathsf{pp},\mathsf{td}) \leftarrow \mathcal{G}(1^\lambda);\ \mathsf{ai} \leftarrow \mathcal{D}(\mathsf{pp})$
$(\mathbb{i},\mathbb{x},\pi) \leftarrow \mathcal{A}(\mathsf{pp},\mathsf{ai})$
$(\mathsf{pk},\mathsf{vk}) \leftarrow \mathcal{I}(\mathsf{pp},\mathbb{i})$
$\mathbb{w} \leftarrow \mathcal{E}_\mathcal{A}(\mathcal{A})$
return $\big(\mathcal{V}(\mathsf{vk},\mathbb{x},\pi) = 1 \wedge (\mathbb{i},\mathbb{x},\mathbb{w}) \notin \mathcal{R}_{\mathsf{pp}}\big)$

Fig. 6: Knowledge soundness for non-interactive arguments.

- $\mathcal{P}(\mathsf{pk},\mathbb{x},\mathbb{w}) \to \pi$. *On input the proving key* pk*, instance* $\mathbb{x}$*, and witness* $\mathbb{w}$*, outputs a proof* π*.*
- $\mathcal{V}(\mathsf{vk},\mathbb{x},\pi) \to b \in \{0,1\}$. *On input the verifying key* vk*, instance* $\mathbb{x}$*, and proof* π*, outputs a bit* b *indicating accept/reject.*

We require ARG *to satisfy* completeness *and* knowledge soundness*, and define an optional notion of* zero knowledge*.*

- ***Completeness.*** *For every unbounded adversary* $\mathcal{A}$*,*

$$\Pr\left[\begin{array}{c|c} (\mathbb{i},\mathbb{x},\mathbb{w}) \in \mathcal{R}_{\mathsf{pp}} & (\mathsf{pp},\mathsf{td}) \leftarrow \mathcal{G}(1^\lambda) \\ \Downarrow & (\mathbb{i},\mathbb{x},\mathbb{w}) \leftarrow \mathcal{A}(\mathsf{pp}) \\ \mathcal{V}(\mathsf{vk},\mathbb{x},\pi) = 1 & (\mathsf{pk},\mathsf{vk}) \leftarrow \mathcal{I}(\mathsf{pp},\mathbb{i}) \\ & \pi \leftarrow \mathcal{P}(\mathsf{pk},\mathbb{x},\mathbb{w}) \end{array}\right] = 1.$$

- ***Knowledge soundness.*** ARG *satisfies knowledge soundness with respect to an auxiliary input distribution* $\mathcal{D}$ *if for every expected polynomial-time adversary* $\mathcal{A}$*, there exists an expected polynomial-time extractor* $\mathcal{E}_\mathcal{A}$ *such that for every set* Z*,*

$$\mathbf{Adv}^{\mathsf{ARG\text{-}KS}}_{\mathcal{A},\mathcal{E}_\mathcal{A},\mathcal{D}}(\lambda) := \Pr\left[\mathsf{ARG\text{-}KS}^{\mathcal{A},\mathcal{E}_\mathcal{A}}_{\mathsf{ARG},\mathcal{D}}(1^\lambda) = 1\right] \leq \mathsf{negl}(\lambda)$$

  The knowledge soundness security game is defined in Figure 6.
- ***Zero-knowledge.*** ARG *satisfies* statistical *zero knowledge if there exists a PPT stateful simulator* $\mathcal{S}$ *with the following syntax:*
  - $\mathcal{S}(\mathsf{td},(\mathsf{pp},\mathbb{i},\mathbb{x})) \to \pi$ *returns a simulated proof* π *(using the trapdoor* td*) upon input* pp*, an index* $\mathbb{i}$*, and an instance* $\mathbb{x}$*.*

  such that for all unbounded distinguishers $\mathcal{A}$*, the following is negligible in* λ*:*

$$\mathbf{Adv}^{\mathsf{ARG\text{-}ZK}}_{\mathcal{A},\mathcal{S}}(\lambda) := \left|\Pr\left[\mathsf{ARG\text{-}ZK}^{\mathcal{A},\mathcal{P}}_{0,\mathsf{ARG},\mathcal{R}}(1^\lambda)\right] - \Pr\left[\mathsf{ARG\text{-}ZK}^{\mathcal{A},\mathcal{S}}_{1,\mathsf{ARG},\mathcal{R}}(1^\lambda)\right]\right|.$$

  The zero-knowledge security games are defined in Figure 7.

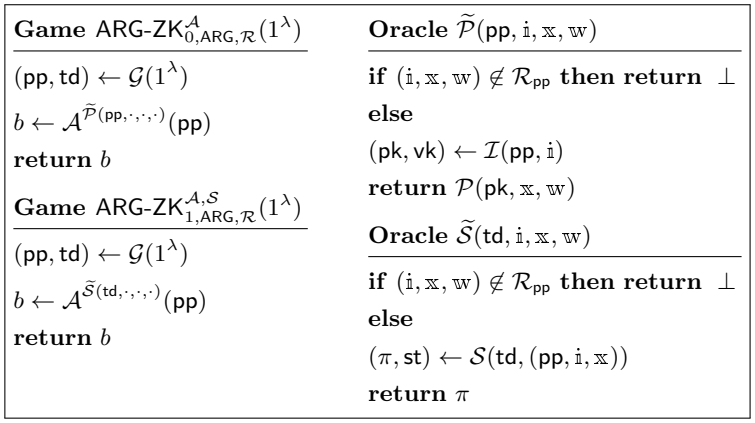

Fig. 7: Zero knowledge for non-interactive arguments.

A.3 Accumulation schemes

We recall the definition of accumulation schemes from [BCMS20].

Definition 10 (Accumulation predicate). *An* accumulation predicate *relative to $\mathcal{O}$ is a pair of PPT oracle algorithms $(\Phi,\mathcal{H})$ with the following syntax:*

- $\mathcal{H}^\theta(1^\lambda) \to \mathsf{pp}_\Phi$. *On input the security parameter 1^λ and with access to $\theta \leftarrow \mathcal{O}(\lambda)$, outputs predicate parameters pp_Φ.*
- $\Phi^\theta(\mathsf{pp}_\Phi,\mathsf{q}) \to b \in \{0,1\}$. *On input the predicate parameters pp_Φ, and query q, outputs a bit b indicating whether the predicate holds.*

Definition 11 (Accumulation scheme). *An* accumulation scheme *for predicate $(\Phi,\mathcal{H})$ relative to $\mathcal{O}$, accepting n predicate inputs and m old accumulators, is a tuple of PPT oracle algorithms $\mathsf{AC} = (\mathrm{G},\mathrm{I},\mathrm{P},\mathrm{V},\mathrm{D})$ with the following syntax:*

- $\mathrm{G}(1^\lambda) \to \mathsf{pp}_{\mathsf{AC}}$. *On input the security parameter 1^λ, outputs public parameters $\mathsf{pp}_{\mathsf{AC}}$.*
- $\mathrm{I}^\theta(\mathsf{pp}_{\mathsf{AC}},\mathsf{pp}_\Phi) \to (\mathsf{apk},\mathsf{avk},\mathsf{dk})$. *On input the public parameters $\mathsf{pp}_{\mathsf{AC}}$, and predicate parameters pp_Φ, deterministically outputs a proving key apk, verifying key avk, and decision key dk.*
- $\mathrm{P}^\theta\left(\mathsf{apk},[\mathsf{q}_i]_{i=1}^n,[\mathsf{acc}_j]_{j=1}^m\right) \to (\mathsf{acc},\mathsf{pf})$. *On input the proving key apk, predicate inputs $[\mathsf{q}_i]_{i=1}^n$, and old accumulators $[\mathsf{acc}_j]_{j=1}^m$, outputs a new accumulator acc and a proof pf.*
- $\mathrm{V}^\theta\left(\mathsf{avk},[\mathsf{q}_i]_{i=1}^n,[\mathsf{acc}_j]_{j=1}^m,\mathsf{acc},\mathsf{pf}\right) \to b \in \{0,1\}$. *On input the verifying key avk, predicate input instances $[\mathsf{q}_i]_{i=1}^n$, old accumulator instances $[\mathsf{acc}_j]_{j=1}^m$, new accumulator instance acc, and proof pf, outputs a bit b indicating accept/reject.*
- $\mathrm{D}^\theta(\mathsf{dk},\mathsf{acc}) \to b' \in \{0,1\}$. *On input the decision key dk and new accumulator acc, outputs a bit b' indicating accept/reject.*

Game $\mathsf{AC\text{-}COM}^{\mathrm{A}}_{\mathsf{AC},\Phi,\mathcal{H}}(1^\lambda)$	**Game** $\mathsf{AC\text{-}S}^{\mathrm{A}}_{\mathsf{AC},\Phi,\mathcal{H}}(1^\lambda)$
$\theta \leftarrow \mathcal{O}(\lambda)$	$\theta \leftarrow \mathcal{O}(\lambda)$
$\mathsf{pp}_{\mathsf{AC}} \leftarrow \mathrm{G}^\theta(1^\lambda); \quad \mathsf{pp}_\Phi \leftarrow \mathcal{H}^\theta(1^\lambda)$	$\mathsf{pp}_{\mathsf{AC}} \leftarrow \mathrm{G}^\theta(1^\lambda); \quad \mathsf{pp}_\Phi \leftarrow \mathcal{H}^\theta(1^\lambda)$
$\left([\mathsf{q}_i]_{i=1}^n, [\mathsf{acc}_j]_{j=1}^m\right) \leftarrow \mathrm{A}^\theta(\mathsf{pp}_{\mathsf{AC}}, \mathsf{pp}_\Phi)$	$\left([\mathsf{q}_i]_{i=1}^n, [\mathsf{acc}_j]_{j=1}^m, \mathsf{acc}, \mathsf{pf}\right) \leftarrow \mathrm{A}^\theta(\mathsf{pp}_{\mathsf{AC}}, \mathsf{pp}_\Phi)$
$(\mathsf{apk}, \mathsf{avk}, \mathsf{dk}) \leftarrow \mathrm{I}^\theta(\mathsf{pp}_{\mathsf{AC}}, \mathsf{pp}_\Phi)$	$(\mathsf{apk}, \mathsf{avk}, \mathsf{dk}) \leftarrow \mathrm{I}^\theta(\mathsf{pp}_{\mathsf{AC}}, \mathsf{pp}_\Phi)$
$(\mathsf{acc}, \mathsf{pf}) \leftarrow \mathrm{P}^\theta\left(\mathsf{apk}, [\mathsf{q}_i]_{i=1}^n, [\mathsf{acc}_j]_{j=1}^m\right)$	$b_{\mathrm{V}} \leftarrow \mathrm{V}^\theta\left(\mathsf{avk}, [\mathsf{q}_i]_{i=1}^n, [\mathsf{acc}_j]_{j=1}^m, \mathsf{acc}, \mathsf{pf}\right)$
$b_{\mathrm{D},j} \leftarrow \mathrm{D}^\theta(\mathsf{dk}, \mathsf{acc}_j) \quad \forall j \in [m]$	$b_{\mathrm{D}} \leftarrow \mathrm{D}^\theta(\mathsf{dk}, \mathsf{acc})$
$b_{\Phi,i} \leftarrow \Phi(\mathsf{pp}_\Phi, \mathsf{q}_i) \quad \forall i \in [n]$	$b_{\mathrm{D},j} \leftarrow \mathrm{D}^\theta(\mathsf{dk}, \mathsf{acc}_j) \quad \forall j \in [m]$
$b_{\mathrm{V}} \leftarrow \mathrm{V}^\theta\left(\mathsf{avk}, [\mathsf{q}_i]_{i=1}^n, [\mathsf{acc}_j]_{j=1}^m, \mathsf{acc}, \mathsf{pf}\right)$	$b_{\Phi,i} \leftarrow \Phi(\mathsf{pp}_\Phi, \mathsf{q}_i) \quad \forall i \in [n]$
$b_{\mathrm{D}} \leftarrow \mathrm{D}^\theta(\mathsf{dk}, \mathsf{acc})$	
$\mathbf{return} \left(\bigwedge_{j=1}^m b_{\mathrm{D},j} \wedge \bigwedge_{i=1}^n b_{\Phi,i} \wedge \neg(b_{\mathrm{V}} \wedge b_{\mathrm{D}})\right)$	$\mathbf{return} \left(b_{\mathrm{V}} \wedge b_{\mathrm{D}} \wedge \neg\left(\bigwedge_{j=1}^m b_{\mathrm{D},j} \wedge \bigwedge_{i=1}^n b_{\Phi,i}\right)\right)$

Fig. 8: Completeness and soundness for accumulation schemes.

We require AC to satisfy *completeness* and *soundness*. The security games for these properties are defined in Figure 8.

– **(Perfect) Completeness.** For every unbounded adversary A, it holds that:

$$\mathbf{Adv}^{\mathsf{AC\text{-}COM}}_{\mathrm{A}}(\lambda) := \Pr\left[\mathsf{AC\text{-}COM}^{\mathrm{A}}_{\mathsf{AC},\Phi,\mathcal{H}}(1^\lambda) = 1\right] = 0.$$

– **Soundness.** AC satisfies soundness if for every PPT adversary A,

$$\mathbf{Adv}^{\mathsf{AC\text{-}S}}_{\mathrm{A}}(\lambda) := \Pr\left[\mathsf{AC\text{-}S}^{\mathrm{A}}_{\mathsf{AC},\Phi,\mathcal{H}}(1^\lambda) = 1\right] \leq \mathsf{negl}(\lambda).$$

A.4 Incrementally verifiable computation

Incrementally verifiable computation (IVC) [Val08] allows for verifying the correctness of a repeated computation by ensuring that each step produces a valid result while also confirming that all preceding steps have been executed correctly. More precisely, given a function F and an initial input z_0, an IVC scheme allows a prover to generate a proof π_i verifying the statement $z_i = F^i(z_0)$ (i.e., applying F iteratively i times on z_0) given a proof π_{i-1} for the statement $z_{i-1} = F^{i-1}(z_0)$ and auxiliary input w. We recall the definition of IVC from [KST22] with the difference that our definition is in the random oracle model.

Definition 12 (IVC). *An incrementally verifiable computation (IVC) scheme consists of PPT algorithms* $\mathsf{IVC} = (\mathbb{G}, \mathbb{I}, \mathbb{P}, \mathbb{V})$ *with the following syntax:*

- $\mathbb{G}(1^\lambda) \to \mathbb{pp}$. *On input security parameter* 1^λ*, outputs public parameters* $\mathbb{pp}$.
- $\mathbb{I}^\theta(\mathbb{pp}, F) \to (\mathbb{pk}, \mathbb{vk})$. *On input the public parameters* $\mathbb{pp}$ *and a polynomial time computable function* F*, deterministically outputs a proving key* $\mathbb{pk}$ *and a verifying key* $\mathbb{vk}$.
- $\mathbb{P}^\theta(\mathbb{pk}, i, z_0, z_i; z_{i-1}, w_{i-1}, \pi_{i-1}) \to \pi_i$. *On input the proving key* $\mathbb{pk}$*, IVC instance* (i, z_0, z_i)*, and IVC witness* $(z_{i-1}, w_{i-1}, \pi_{i-1})$*, outputs an IVC proof* π_i.
- $\mathbb{V}^\theta(\mathbb{vk}, i, z_0, z_i, \pi_i) \to b \in \{0, 1\}$. *On input the verifying key* $\mathbb{vk}$*, IVC instance* (i, z_0, z_i)*, and IVC proof* π_i*, outputs a bit* b *indicating accept/reject.*

We require IVC *to satisfy* completeness *and* knowledge soundness. *We also define an optional* succinctness *property.*

- ***(Perfect) Completeness.*** *For any PPT adversary* $\mathbb{A}$*,*

$$\Pr\left[\mathbb{V}^\theta(\mathbb{vk}, i, z_0, z_i, \pi_i) = 1 \;\middle|\; \begin{array}{r} \theta \leftarrow \mathcal{O}(\lambda); \mathbb{pp} \leftarrow \mathbb{G}(1^\lambda) \\ F, (i, z_0, z_i, z_{i-1}, w_{i-1}, \pi_{i-1}) \leftarrow \mathbb{A}^\theta(\mathbb{pp}) \\ (\mathbb{pk}, \mathbb{vk}) \to \mathbb{I}^\theta(\mathbb{pp}, F) \\ z_i = F(z_{i-1}, w_{i-1}) \\ \mathbb{V}^\theta(\mathbb{vk}, i-1, z_0, z_{i-1}, \pi_{i-1}) = 1 \\ \pi_i \leftarrow \mathbb{P}^\theta(\mathbb{pk}, i, z_0, z_i; z_{i-1}, w_{i-1}, \pi_{i-1}) \end{array}\right] = 1.$$

  where F *is a polynomial time computable function.*
- ***Knowledge soundness.*** *For any PPT adversary* $\mathbb{A}$ *and any constant* $n \in \mathbb{N}$*, there exists an efficient extractor* $\mathbb{E}_\mathbb{A}$ *such that:*

$$\Pr\left[\begin{array}{c} z_n = z \text{ where} \\ z_{i+1} \leftarrow F(z_i, w_i) \\ \forall i \in \{0, \ldots, n-1\} \end{array} \;\middle|\; \begin{array}{r} \theta \leftarrow \mathcal{O}(\lambda); \mathbb{pp} \leftarrow \mathbb{G}(1^\lambda) \\ (F, (z_0, z), \pi) \leftarrow \mathbb{A}^\theta(\mathbb{pp}, r) \\ (w_0, \ldots, w_{n-1}) \leftarrow \mathbb{E}^\theta_\mathbb{A}(\mathbb{pp}, r) \end{array}\right] \approx$$
$$\Pr\left[\mathbb{V}^\theta(\mathbb{vk}, n, z_0, z, \pi) = 1 \;\middle|\; \begin{array}{r} \theta \leftarrow \mathcal{O}(\lambda); \mathbb{pp} \leftarrow \mathbb{G}(1^\lambda) \\ (F, (z_0, z), \pi) \leftarrow \mathbb{A}^\theta(\mathbb{pp}, r) \\ (\mathbb{pk}, \mathbb{vk}) \to \mathbb{I}^\theta(\mathbb{pp}, F) \end{array}\right]$$

  where r *denotes a random tape.*
- ***Succinctness.*** *An* IVC *is called succinct if the size of the IVC proof* π *does not grow with the number of applications* n.

B Instantiating the PKE for PoPCS via CL encryption

Using ElGamal encryption in its basic form allows the receiver to recover only the group encoding of the evaluation. In the context of VSS, this encrypted evaluation represents a share of the secret and may therefore be an arbitrarily large field element, making direct use of ElGamal impractical. While a common

folklore technique mitigates this issue by splitting plaintexts into small fragments and performing brute-force discrete logarithm computations, we instead adopt a different approach: we instantiate the scheme using CL encryption in which discrete logarithms are easy in a designated subgroup, thereby avoiding the need for expensive exhaustive search computations.

B.1 Recap of CL encryption

In [CL15], Castagnos and Laguillaumie introduced the CL framework, which constructs cyclic groups $\mathbb{G}$ where DDH is believed to hold, along with a subgroup $\mathbb{H} \subset \mathbb{G}$ in which the discrete logarithm problem is easy. We use CL encryption to instantiate the PKE in our PoPCS. This framework allows encrypting values "in the exponent" enabling efficient Σ-protocols to prove statements about messages, while also supporting efficient decryption (unlike standard ElGamal encryption). In the following, we recall the definition of CL framework, using multiplicative notation for group operations (in contrast to the additive notation used elsewhere in this paper).

Let λ be a security parameter and let q be a μ-bit prime with $\mu \geq \lambda$. The framework $\mathsf{Gen}_{\mathsf{CL}} = (\mathsf{Gen}, \mathsf{Solve})$ is defined as follows.

- $\mathsf{Gen}(1^\lambda, q)$ outputs public parameters $\mathsf{pp} = (\widehat{\mathrm{G}}, \mathrm{G}, \mathrm{F}, \mathrm{G}^q, \tilde{s}, f, g_q, g)$, where:
  - $\widehat{\mathrm{G}}$ is a finite abelian group of (unknown) order $\widehat{n} = q \cdot \widehat{s}$ with $\gcd(q, \widehat{s}) = 1$, and valid encodings in $\widehat{\mathrm{G}}$ are efficiently recognizable. The factor $\widehat{s}$ is unknown, but we are given an upper bound $\tilde{s} \geq \widehat{s}$.
  - $\mathrm{G} \subseteq \widehat{\mathrm{G}}$ is a cyclic subgroup of (unknown) order $n = q \cdot s$ with $s \mid \widehat{s}$.
  - $\mathrm{F} = \langle f \rangle \subseteq \widehat{\mathrm{G}}$ is the unique cyclic subgroup of order q.
  - $\mathrm{G}^q = \langle g_q \rangle = \{x^q : x \in \mathrm{G}\}$ is the cyclic subgroup of q-th powers of (unknown) order s. Since $\mathrm{F} \subseteq \mathrm{G}$, we have $\mathrm{G} \simeq \mathrm{G}^q \times \mathrm{F}$ with $g := g_q \cdot f$ being a generator of G.
- $\mathsf{Solve}(q, \mathsf{pp}, X)$ is a deterministic polynomial-time algorithm that solves the discrete logarithm problem in F, i.e.,

$$\Pr\left[x = x' \;\middle|\; \begin{array}{r} \mathsf{pp} \leftarrow\!\!\$\ \mathsf{Gen}(1^\lambda, q) \\ x \leftarrow\!\!\$\ \mathbb{Z}/q\mathbb{Z}\ X \leftarrow f^x \\ x' \leftarrow \mathsf{Solve}(q, \mathsf{pp}, X) \end{array}\right] = 1.$$

We recall the hard subgroup membership problem for groups with an easy discrete logarithm subgroup (HSM-CL) from [CLT18]. The HSM-CL assumption states that distinguishing elements of G^q from random elements of G is computationally hard.

Definition 13 (HSM-CL assumption). *Let λ be a security parameter and let $\mathsf{Gen}_{\mathsf{CL}} = (\mathsf{Gen}, \mathsf{Solve})$ be as defined above. Let $\mathcal{D}$ (resp. $\mathcal{D}_q$) be distributions such that $\{g^x :\leftarrow\!\!\$\ \mathcal{D}\}$ (resp. $\{g_q^x : x \leftarrow\!\!\$\ \mathcal{D}_q\}$) are statistically close to uniform over* G *(resp.* G^q*).*

$\mathsf{CL.Setup}(1^\lambda, 1^\mu)$
1. sample a μ-bit prime q
2. $\mathsf{pp} \leftarrow\!\!\$\ \mathsf{Gen}(1^\lambda, q)$
3. return pp

$\mathsf{CL.KeyGen}(\mathsf{pp})$
1. sample $\alpha \leftarrow\!\!\$\ \mathcal{D}_q$ and $Y \leftarrow g_q^\alpha$
2. set $\mathsf{pk} := Y$ and $\mathsf{sk} := \alpha$
3. return $(\mathsf{pk}, \mathsf{sk})$

$\mathsf{CL.Enc}(\mathsf{pk}, m)$
1. sample $\hat{\omega} \leftarrow\!\!\$\ \mathcal{D}_q$
2. return $(g_q^{\hat{\omega}}, f^m \cdot Y^{\hat{\omega}})$

$\mathsf{CL.Dec}(\mathsf{sk}, (c_1, c_2))$
1. compute $M = c_2/c_1^\alpha$
2. return $\mathsf{Solve}(q, \mathsf{pp}, M)$

Fig. 9: Encryption scheme from HSM-CL [CLT18]

The advantage of an adversary $\mathcal{A}$ is

$$\left| \Pr\left[b = b' \;\middle|\; \begin{array}{r} \mathsf{pp} \leftarrow\!\!\$\ \mathsf{Gen}(1^\lambda, q) \\ x \leftarrow\!\!\$\ \mathcal{D};\ x' \leftarrow\!\!\$\ \mathcal{D}_q \\ X_0 \leftarrow g^x;\ X_1 \leftarrow g_q^{x'}; b \leftarrow\!\!\$\ \{0,1\} \\ b' \leftarrow \mathcal{A}(q, \mathsf{pp}, X_b, \mathsf{Solve}(\cdot)) \end{array} \right] - \frac{1}{2} \right| .$$

The HSM-CL assumption holds for $\mathsf{Gen}_{\mathsf{CL}}$ if for every PPT adversary $\mathcal{A}$, $\mathbf{Adv}_{\mathcal{A}}^{\mathsf{hsm\text{-}cl}}(\lambda) = \mathsf{negl}(\lambda)$.

Under the HSM-CL assumption, [CLT18] proposes the following public-key encryption scheme. The plaintext space is $\mathbb{F}_q$ for a μ-bit prime q with $\mu \geq \lambda$, and both the secret key x and the encryption randomness r are sampled from the distribution $\mathcal{D}_q$. As shown in [CLT18, Lemma 4], a practical instantiation of $\mathcal{D}_q$ is the uniform distribution over $\{0, \ldots, S\}$, where $S := 2^{\lambda-2} \cdot \tilde{s}$. The scheme, depicted in fig. 9, is proven to be semantically secure against chosen-plaintext attacks (IND-CPA) under the HSM-CL assumption.

B.2 PoPCS from CL encryptions

To use CL encryption in our instantiation, we observe that we have two objects living in different groups: a polynomial commitment $C = \langle \mathbf{f}, \mathbf{G} \rangle + [\omega]H$, which lives in a cyclic group $\mathbb{G}$ of order q, and a CL ciphertext

$$(c_1, c_2) = (g_q^{\hat{\omega}},\ f^m \cdot Y^{\hat{\omega}}) \in \mathrm{G}^q \times \mathrm{G},$$

whose plaintext is alleged to be $m := \langle \mathbf{f}, \mathbf{z} \rangle$. To incorporate this ciphertext into the commitment, we use a new generator U and create a fresh Pedersen commitment to the CL plaintext as $C_m = [m]U + [\omega']H \in \mathbb{G}$. We then run a Σ-protocol (see [DGK+21], Section B.4) to show that the CL ciphertext contains the same value as C_m. Finally, the argument of knowledge $(\mathsf{Open}, \mathsf{VerifyOpen})$ for the PoPCS can be implemented similarly to the construction in section 3.1, with the difference that the committed value D is now defined as

$$D = \langle \mathbf{f}, \mathbf{G} \rangle + [\omega + \omega']H + [\langle \mathbf{f}, \mathbf{z} \rangle]U.$$

Theorem 3. *The PoPCS constructed in section 3.1, when instantiated with CL encryption, satisfies completeness, extractability, and zero knowledge.*

C Compressed Σ-protocol theory

For a given Pedersen commitment to a long vector $\mathbf{x} \in \mathbb{F}^n$, consider a zero-knowledge proof that shows that the prover knows $\mathbf{x}$, while also revealing the value of $\mathbb{L}(\mathbf{x})$ for any linear form $\mathbb{L}$. A simple Σ-Protocol can be used for this. The idea of the compressed Σ-protocol [AC20] is to replace the final (long) message of the Σ-Protocol with a short (logarithmic) proof that shows the prover knows this message. This approach keeps the proof zero-knowledge but drastically reduces the amount of data needed to be sent. That is, instead of sending a ton of information (linear), only a tiny bit (logarithmic) is required. Below we review the compression technique in [AC20]. We start by recalling the basic Σ-protocol used as the main pivot.

C.1 The basic Σ-protocol

Let $\mathbb{G}$ be a group of prime order q. For $\mathbf{G} \in \mathbb{G}^n$, $\mathbf{x} \in \mathbb{F}_q^n$, and $c \in \mathbb{F}_q$ we define $\langle \mathbf{x}, \mathbf{G} \rangle := \sum [x_i]G_i$, and $[c]\mathbf{G} := ([c]G_0, \ldots, [c]G_{n-1})$. The component-wise group operation between two vectors $\mathbf{G}, \mathbf{H} \in \mathbb{G}^n$ is denoted as $\mathbf{G} + \mathbf{H} := (G_0 + H_0, \ldots, G_{n-1} + H_{n-1})$.

A Pedersen vector commitment is defined by the following algorithms:

Setup. Sample $\mathbf{G} = (G_0, \ldots, G_{n-1}) \leftarrow\!\!\$\ \mathbb{G}^n$ and $H \leftarrow\!\!\$\ \mathbb{G}$.
Commit. Define $\mathsf{Commit} : \mathbb{F}_q^n \times \mathbb{F}_q \to \mathbb{G}$ as

$$\mathsf{Commit}(\mathbf{x}, \gamma) = \langle \mathbf{x}, \mathbf{G} \rangle + [\gamma]H.$$

We assume, in the following, that Pedersen vector commitments are employed in our setting, and we are interested in opening such commitments with respect to a *linear form* $\mathbb{L} : \mathbb{F}_q^n \to \mathbb{F}_q$; that is, in revealing the value $\mathbb{L}(\mathbf{x})$ together with a corresponding proof of validity, without disclosing any additional information about $\mathbf{x}$. More formally, we consider an NP relation defined as

$$\mathcal{R} = \{(C, \mathbb{L}, y; \mathbf{x}, \gamma) : C = \langle \mathbf{x}, \mathbf{G} \rangle + [\gamma]H, y = \mathbb{L}(\mathbf{x})\},$$

where $C \in \mathbb{G}$, $y, \gamma \in \mathbb{F}_q$, $\mathbf{x} \in \mathbb{F}_q^n$, and $\mathbb{L} \in \mathcal{L}(\mathbb{F}_q^n)$ comes from the set of linear forms on $\mathbb{F}_q^n$ defined below.

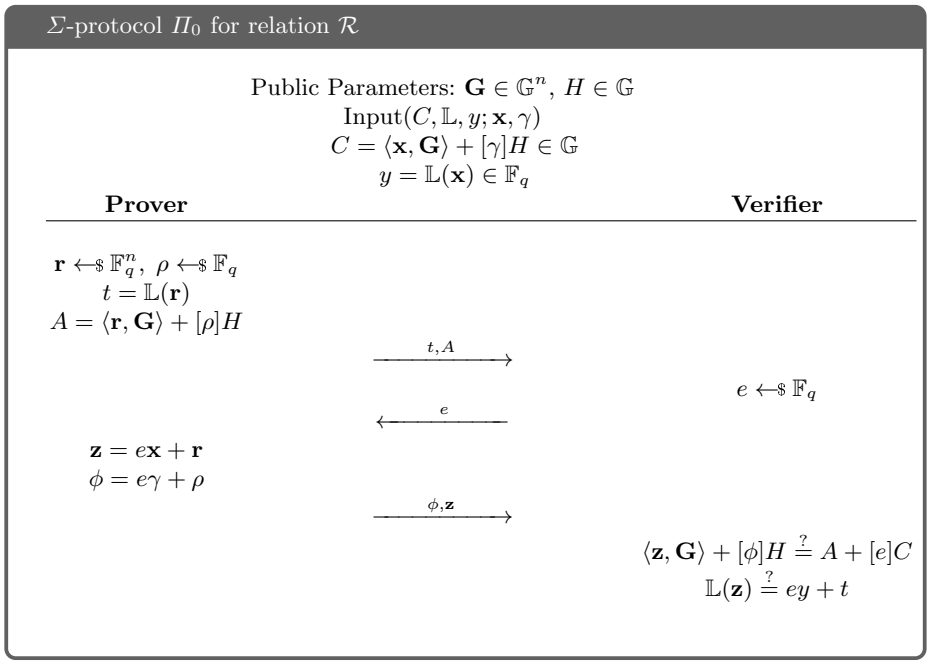

Fig. 10: Σ-protocol Π_0

Definition 14. $\mathcal{L}(\mathbb{F}_q^n) := \{(\mathbb{L} : \mathbb{F}_q^n \to \mathbb{F}_q) : \mathbb{L}$ *is a* $\mathbb{F}_q$*-linear map*$\}$.

We sometimes make the description of linear maps more explicit by representing them as vectors, so that for $\mathbb{L} = (a_1, \ldots, a_n) \in \mathbb{F}_q^n$, we define $\mathbb{L}(\mathbf{x}) = \sum a_i x_i = \langle \mathbf{a}, \mathbf{x} \rangle$.

We start by describing a basic Σ-protocol for $\mathcal{R}$. The protocol is denoted by Π_0 and depicted in Fig. 10.

C.2 Compressing basic Σ-protocol

We now recall the techniques introduced by Bulletproofs [BBB+18] and compressed Σ-protocols [AC20] that enable compressing the described basic Σ-protocol. The key idea is to replace the linear-size vector $(\phi, \mathbf{z})$ in the last round with a logarithmic-size proof of knowledge of $(\phi, \mathbf{z})$. In other words, rather than sending $(\phi, \mathbf{z})$ directly as part of the proof, the prover gives a proof of knowledge for the relation

$$\mathcal{R}_1 = \{(D, \mathbb{L}, \hat{y}; \hat{\mathbf{z}}) : D = \langle \hat{\mathbf{z}}, \hat{\mathbf{G}} \rangle, \hat{y} = \mathbb{L}(\mathbf{z})\},$$

where $\hat{\mathbf{G}} := (\mathbf{G}, H) \in \mathbb{G}^{n+1}$, $\hat{\mathbf{z}} := (\phi, \mathbf{z}) \in \mathbb{F}_q^{n+1}$, $D := A + [e]C$, and $\hat{y} := ey + t$.

The relation enforces two constraints, one on the vector commitment and one on the linear form. As observed in [BBB+18], the two constraints can be

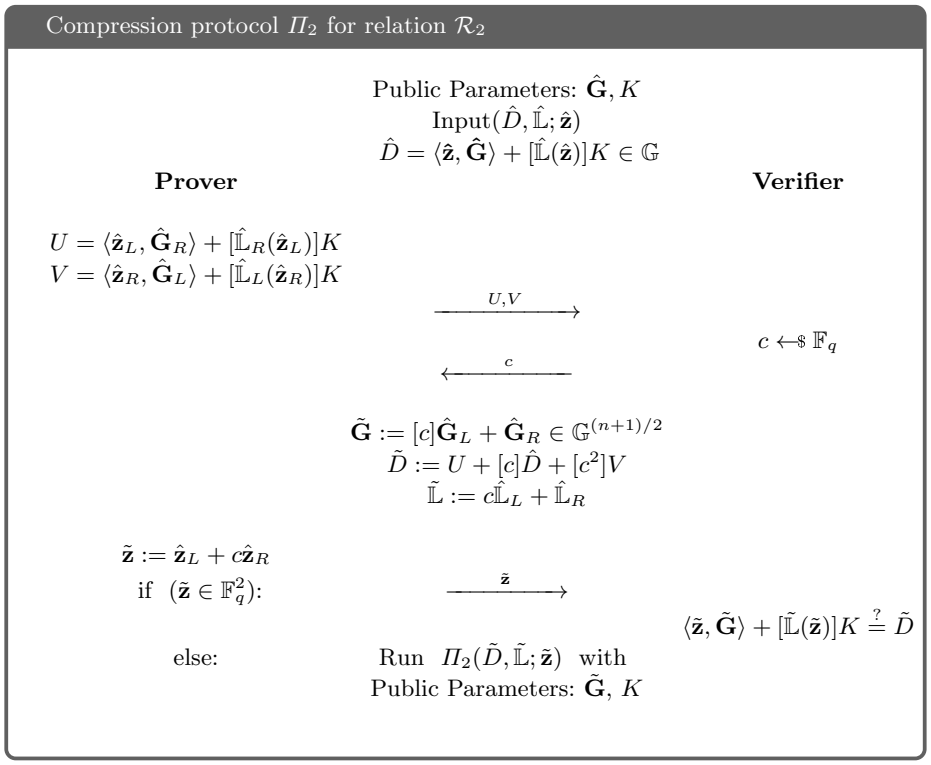

Fig. 11: Compression protocol Π_2

combined by incorporating the linear form into the vector commitment using an additional generator $K \in \mathbb{G}$. This amounts to the relation

$$\mathcal{R}_2 = \{(\hat{D}, \hat{\mathbb{L}}; \hat{\mathbf{z}}) : \hat{D} = \langle \hat{\mathbf{z}}, \hat{\mathbf{G}} \rangle + [\hat{\mathbb{L}}(\hat{\mathbf{z}})]K\},$$

where $\hat{D} := D + [f\hat{y}]K$, and $\hat{\mathbb{L}}(\hat{\mathbf{z}}) := f\mathbb{L}(\mathbf{z})$ for a random challenge $f \in \mathbb{F}_q$ chosen by the verifier. The key contribution of compressed Σ-protocol paper is a compression technique that enables an efficient proof of knowledge protocol Π_2 for $\mathcal{R}_2$. Figure 11 describes this protocol, wherein for simplicity it is assumed that $n + 1$ is a power of 2.

Let us denote by Π_1 the reduction protocol from relation $\mathcal{R}_1$ to relation $\mathcal{R}_2$. It is not hard to see that composing all these protocols as $\Pi = \Pi_2 \circ \Pi_1 \circ \Pi_0$ results in a compressed Σ-protocol for the relation $\mathcal{R}$. For completeness, we describe this protocol in Fig. 12.

C.3 Compressed protocol as a black-box

From now on, we assume we have black-box access to the compressed Σ-protocol above and can open arbitrary linear forms on multiple compact vector com-

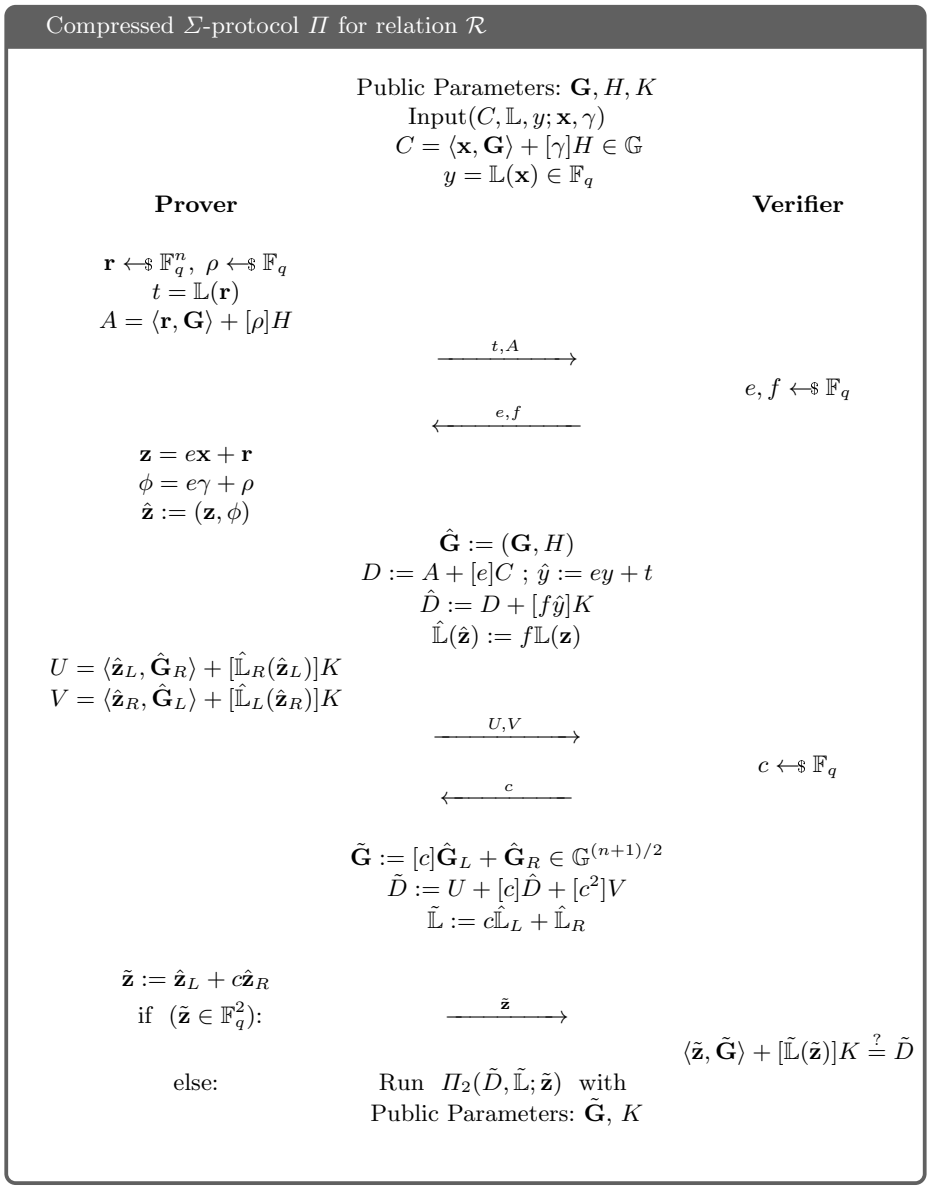

Fig. 12: Compressed Σ-protocol Π

mitments. A particular opening check we will be interested in throughout this work is nullity checks, where the prover claims that $\mathbb{L}(\mathbf{x}) = 0$ for some linear form $\mathbb{L}$. In practice, the prover will likely need to perform numerous nullity checks on the committed vector $\mathbf{x}$. An amortization technique in compressed Σ-

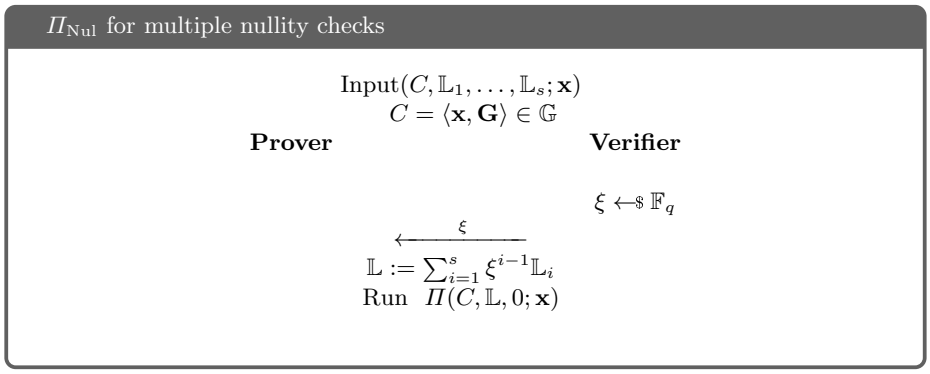

Fig. 13: Π_{Nul} for multiple nullity checks

protocol (CSP) shows how to achieve this: assume the prover wishes to show that $\mathbb{L}_i(\mathbf{x}) = 0, i = 1, \ldots, s$, for linear forms $\mathbb{L}_1, \ldots, \mathbb{L}_s$. The verifier samples a random element $\rho \leftarrow\$ \mathbb{F}_q$ and asks the prover to open the linear form $\mathbb{L}(\mathbf{x}) := \sum_{i=1}^{s} \rho^{i-1} \mathbb{L}_i(\mathbf{x})$. By Schwartz-Zippel lemma, $\mathbb{L}(\mathbf{x}) = 0$ implies that $\mathbb{L}_i(\mathbf{x}) = 0$ for all i with probability at least $1 - \frac{s-1}{q}$, which is close to 1 when q is exponential and s is polynomial in the security parameter. The protocol for multiple nullity checks is presented in Fig. 13. We note that all the discussions and protocols presented above still hold when linear maps are replaced by affine maps. To simplify the explanation, we describe protocols only in terms of linear maps. However, if we need to distinguish between them explicitly, we use the notation $\mathbb{A} := (a_1, \ldots, a_n; b)$ for affine maps; i.e., $\mathbb{A}(\mathbf{x}) = \langle \mathbf{a}, \mathbf{x} \rangle + b$, for $\mathbf{x} \in \mathbb{F}_q^n$.

C.4 Proof of circuit satisfiability

We recall the compressed Σ-protocol for circuit satisfiability in the CSP work. Let $\mathcal{R}_{cs} = \{(F; \mathbf{x} \in \mathbb{F}_q^{\ell_{\mathsf{in}}}, \mathbf{y} \in \mathbb{F}_q^{\ell_{\mathsf{out}}}) : F(\mathbf{x}) = \mathbf{y}\}$ be the circuit satisfiability relation defined over all circuits F with ℓ_{in} inputs and ℓ_{out} output. The key idea in CSP is to reduce the problem of circuit satisfiability into the problem of opening a linear form which we know how to do efficiently from the previous section. The reduction is through a linearization technique that we recall below.

Linearization technique. Let F be an arithmetic circuit over $\mathbb{F}_q$ with ℓ_{in} input wires and ℓ_{out} output wires. We assume F has ℓ_{mul} (fan-in 2) multiplication gates with input-output wires $\{(l_i, r_i, o_i)\}_{i=1\ldots,\ell_{\mathsf{mul}}}$, and a number of addition gates that is immaterial. The prover holds a vector $\mathbf{x}$ and is willing to prove that $F(\mathbf{x}) = \mathbf{y}$. It can be observed that for each i, there are affine forms $u_i, v_i : \mathbb{F}_q^{\ell_{\mathsf{in}}+i-1} \to \mathbb{F}_q$, defined by F, such that $l_i = u_i(\mathbf{x}, o_1, \ldots, o_{i-1})$ and $r_i = v_i(\mathbf{x}, o_1, \ldots, o_{i-1})$. Similarly, there is an affine form $w : \mathbb{F}_q^{\ell_{\mathsf{in}}+\ell_{\mathsf{mul}}} \to \mathbb{F}_q^{\ell_{\mathsf{out}}}$ such that for the resulting output wires $\mathbf{y}$, we have that $\mathbf{y} = w(\mathbf{x}, o_1, \ldots, o_{\ell_{\mathsf{mul}}})$. In other words, a given tuple $(\mathbf{x}, o_1, \ldots, o_{\ell_{\mathsf{mul}}})$ meets the circuit, resulting in the output $\mathbf{y}$ if and only if

$$
\begin{gathered}
w(\mathbf{x}, o_1, \ldots, o_{\ell_{\mathsf{mul}}}) = \mathbf{y} \\
\wedge \\
u_i(\mathbf{x}, o_1, \ldots, o_{i-1}) \cdot v_i(\mathbf{x}, o_1, \ldots, o_{i-1}) = o_i \qquad \forall i = 1, \ldots, \ell_{\mathsf{mul}}
\end{gathered} \tag{2}
$$

To convince the verifier, the prover must show that these constraints hold. However, the prover lacks adequate tools due to the non-linearity introduced by the last constraints. To address this, a linearization technique can be used which works as follows. The prover selects a random polynomial $L(X) \in \mathbb{F}_q[X]$ of degree at most ℓ_{mul} that defines a packed secret sharing of the left wires vector $(l_1, \ldots, l_{\ell_{\mathsf{mul}}})$. Similarly, the prover selects a random polynomial $R(X) \in \mathbb{F}_q[X]$ of degree at most ℓ_{mul} that defines a packed secret sharing of the right wires vector $(r_1, \ldots, r_{\ell_{\mathsf{mul}}})$. Finally, the prover computes the product polynomial $O(X) := L(X)R(X)$ of degree at most $2\ell_{\mathsf{mul}}$. Now, instead of committing to $(\mathbf{x}, o_1, \ldots, o_{\ell_{\mathsf{mul}}})$, the prover commits to a vector containing $\mathbf{x}$ and these polynomials. Since we are selecting $L(X), R(X)$ so that $l_i = L(i)$ and $r_i = R(i)$ for $i = 1, \ldots, \ell_{\mathsf{mul}}$, the prover does not even need to include $\{(L(i), R(i))\}_{i=1,\ldots,\ell_{\mathsf{mul}}}$ in this vector as they can be computed by affine forms over $(\mathbf{x}, O(1), \ldots, O(\ell_{\mathsf{mul}}))$. This results in the prover committing to a vector

$$
\boldsymbol{\mathcal{X}} := (\mathbf{x}, L(0), R(0), O(0), O(1), \ldots, O(2\ell_{\mathsf{mul}})) \in \mathbb{F}_q^{\ell_{\mathsf{in}}+2\ell_{\mathsf{mul}}+3},
$$

and proving that the constraints in (2) hold. Checking the first constraint is simply a nullity check for $w(\mathbf{x}, o_1, \ldots, o_{\ell_{\mathsf{mul}}}) - \mathbf{y} = 0$. To check the non-linear constraints, the verifier selects a random challenge $c \leftarrow\!\!\$\ \mathbb{F}_q \backslash \{1, \ldots, \ell_{\mathsf{mul}}\}$ and sends it to the prover. Next, the prover opens the committed vector $\boldsymbol{\mathcal{X}}$ in $l := L(c)$, $r := R(c)$ and $o := O(c)$, and sends (l, r, o) together with opening proofs to the verifier[12]. The verifier can now check whether $o = lr$. By Schwartz-Zippel lemma, a cheating prover will be caught with probability at least $1 - \frac{2\ell_{\mathsf{mul}}}{(q-\ell_{\mathsf{mul}})}$. The resulting protocol is denoted by Π_{cs} and presented in Fig. 14.

D The SPARK compiler

SPARK, introduced in Spartan [Set20], is a cryptographic compiler that transforms a PCS for dense polynomials into one that efficiently handles sparse multilinear polynomials. Consider a constraint matrix M of dimension $m \times n$ that contains s non-zero entries, where $s \ll m \times n$. A naive multilinear extension of this matrix would require the prover to commit to all mn entries, including the zeros. This results in a prohibitive $\mathcal{O}(mn)$ prover overhead. SPARK circumvents this by translating the matrix into a dense representation. It encodes the matrix M using three dense vectors of length s:

- `val`: The non-zero values of the matrix.
- `row`: The row indices corresponding to each non-zero value.

[12] The linear forms are defined by evaluations of Lagrange polynomials at c, i.e., $\delta_0(c), \delta_1(c), \ldots, \delta_{2\ell_{\mathsf{mul}}}(c)$, where $\delta_i(X) = \prod_{j=0,\ldots,2\ell_{\mathsf{mul}}; j\neq i} \frac{X-j}{i-j}$.

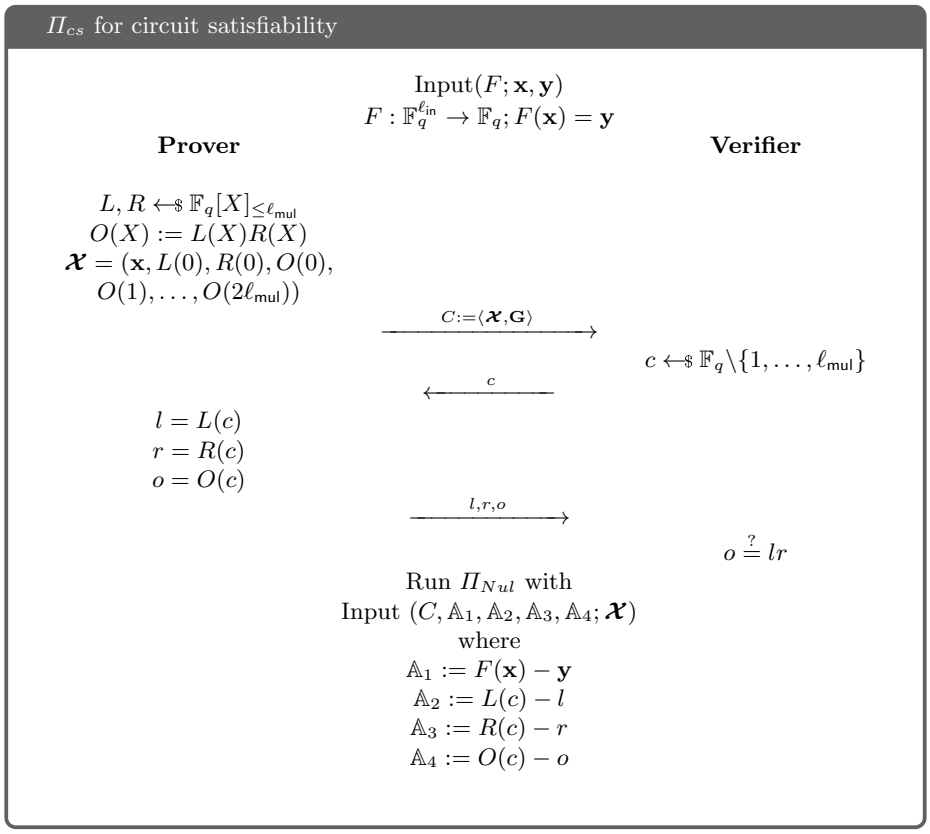

Fig. 14: Π_{cs} for circuit satisfiability

– `col`: The column indices corresponding to each non-zero value.

Instead of committing directly to M, the SPARK prover computes the multilinear extensions of `val`, `row`, and `col` and commits to them using the underlying dense PCS. Because these vectors only have size s, the prover's commitment time and the resulting commitment size scale with $\mathcal{O}(s)$, the number of non-zero entries. When the verifier challenges the prover to evaluate the multilinear extension of M at a random point, the prover does not reconstruct the matrix. Instead, it uses an *offline memory checking* to prove that the non-zero entries were honestly mapped to their correct coordinates during evaluation. Ultimately, SPARK reduces sparse matrix evaluation to the evaluation of a few dense polynomials (proved via the underlying PCS) and a memory consistency check. Consequently, the total prover time for committing and evaluating the matrix is bounded by $\mathcal{O}(s)$ without incurring overhead that scales with M's dimensions.

D.1 Instantiation with a multilinear PCS based on CSP

To instantiate the SPARK compiler within our protocol, we require an underlying multilinear PCS that supports an accumulation scheme. Below, we present such a scheme based on the CSP theory.

Let $\tilde{f}(X_1, \ldots, X_l)$ be an l-variate multilinear polynomial, and let $\mathbf{x} \in \mathbb{F}_q^N$ denote the vector of its evaluations over the boolean hypercube $\{0,1\}^l$, where $N = 2^l$. Evaluating $\tilde{f}$ at a point $\mathbf{a} = (a_1, \ldots, a_l) \in \mathbb{F}_q^l$ can be expressed as a linear form $L_{\mathbf{a}}$ over the evaluation vector $\mathbf{x}$:

$$\tilde{f}(a_1, \ldots, a_l) = L_{\mathbf{a}}(\mathbf{x}) = \langle \mathbf{q_a}, \mathbf{x} \rangle$$

where $\mathbf{q_a} = \big((1-a_1)\cdots(1-a_l), \ldots, a_1 \cdots a_l\big)$ is a structured vector defined as the tensor product $\bigotimes_{i=1}^{l} \binom{1-a_i}{a_i}$. By framing polynomial evaluation as the opening of a linear form, we can construct the PCS directly over a Pedersen vector commitment. Specifically, let $P = \langle \mathbf{x}, \mathbf{G} \rangle$ be the commitment to $\mathbf{x}$ using public generators $\mathbf{G} \in \mathbb{G}^N$. To prove that $\tilde{f}(\mathbf{a}) = v$, the prover must demonstrate that $L_{\mathbf{a}}(\mathbf{x}) = v$ with respect to the commitment P. The prover and verifier engage in an l-round CSP for opening this linear form. In each round $i \in \{1, \ldots, l\}$, the prover sends cross commitments $(A_i, B_i) \in \mathbb{G}^2$, and the verifier replies with a random challenge $e_i \in \mathbb{F}_q$. Using e_i, the evaluation vector $\mathbf{x}$, the generator vector $\mathbf{G}$, and the linear form $L_{\mathbf{a}}$ are all folded in half. After l rounds, these vectors are compressed to length 1, and the prover sends the final folded scalar $\mathbf{z} \in \mathbb{F}_q$ to the verifier.

Accumulation-friendly verification. At the end of the CSP, the verifier must check the relation:

$$[\mathbf{z}]G' + [L'(\mathbf{z})]K \stackrel{?}{=} A + [e]P + [e^2]B \tag{3}$$

where G' is the final folded generator, L' is the final folded linear form, and $K \in \mathbb{G}$ is an independent public generator.

Computing G' and L' requires linear time, which is why the CSP verifier is linear. However, as first observed in Halo [BGH19], the algebraic structure of these elements allows the following amortization:

- *Logarithmic computation of L'.* Because $L_{\mathbf{a}}$ has a tensor product structure $\mathbf{q_a}$, the folded linear form L' evaluated at $\mathbf{z}$ can be computed by the verifier in $\mathcal{O}(\log N)$ time as

$$L'(\mathbf{z}) = \mathbf{z} \cdot \prod_{i=1}^{l} \Big((1 - a_i) + e_i a_i\Big)$$

- *Deferring G' via accumulation.* Directly computing the folded group element G' requires $\mathcal{O}(N)$ group operations. However, G' can be viewed as a polynomial commitment to a degree $N-1$ polynomial h, whose coefficients are multiplicative combinations of the challenges $(e_1, \ldots, e_l)$. As a result, the

verifier does not compute G' explicitly. Instead, the prover constructs G' and proves its correctness (probabilistically) via a PCS opening of h at a challenge point. The resulting opening can then be aggregated with other PCS openings using an accumulator, yielding a single linear-size verification check that is deferred until the very end.

This yields an accumulation-friendly PCS that, when integrated with the SPARK compiler, allows our protocol to efficiently commit to and evaluate sparse multilinear polynomials while maintaining sub-linear costs for the accumulation verifier.