Verifiable Streaming Computation and Step-by-Step Zero-Knowledge

Abtin Afshar, Rishab Goyal*

Abstract

We propose a new incrementally computable proof system, called Incrementally Verifiable *Streaming* Computation (IVsC). IVsC enables computing incremental proofs of correct execution for any RAM program $\mathcal{M}$ on a *streaming* input x. Input x is called a *streaming* input if it is only available on-the-fly as part of an ongoing data generation/streaming process, and not available at once. We also propose a new notion of zero-knowledge features for IVsC that guarantees the proof can be incrementally verified w.r.t. an encrypted digest, where the proof and digest hide the full data stream. We design zero-knowledge IVsC from a wide variety of standard falsifiable assumptions (such as decision-linear/sub-exponential DDH/LWE).

We also introduce a new notion of non-interactive zero-knowledge proofs, that we call *step-by-step* zero-knowledge protocols. Such protocols have strong zero-knowledge guarantees, wherein the prover's entire internal memory is simulatable at any point during proof generation. That is, unlike standard zero-knowledge proofs, where only the final proof is simulatable, we can also simulate prover's state at every step of the computation. Such proof systems will be useful in settings where an adversary could corrupt an honest prover even before it generates the full proof. We show that a zero-knowledge IVsC system can be used (almost) as a black-box to design step-by-step zero-knowledge proof systems, therefore secure under standard assumptions.

*Email: {abtin,rishab}@cs.wisc.edu. Research supported by Wisconsin Alumni Research Foundation.

Contents

1 Introduction

Incrementally verifiable computation (IVC) [Val08] and, its popular generalization, proof carrying data (PCD) [CT10] are powerful succinct argument protocols. They enable incremental proof generation for a distributed computation performed by a network of mutually distrustful parties. PCDs have major theoretical and practical significance [CTV13, CTV15, NT16, Min17, KB23, BMRS20, CCDW20]. They offer strong succinctness and soundness guarantees, which informally state that the proof size is very short and no cheating polytime attacker can create proofs for incorrect computation. In privacy-sensitive applications (e.g., zk-EVM [But22] and more [CTV15, BBB+18, XZC+22, ZGSX23]), it is extremely valuable to support zero-knowledge guarantees as well. That is, the proofs (both, intermediate and final) must hide the sensitive data.

The most common approaches to design PCDs rely on recursively composing succinct non-interactive arguments of knowledge (SNARKs), or related primitives with similar accumulation/folding features. Such PCDs are typically proven secure in oracle models (e.g., Random Oracles), or under non-falsifiable (knowledge) assumptions (e.g., knowledge-of-exponents) [CT10, BCCT13, BCTV14, BGH19, BCMS20, COS20, BDFG21, BCL+21, CY21, CCS22, KST22, KS22, HN23, CCG+23, BC23, CGSY24]. Despite numerous advantages (e.g., transparent setup via ROs, nearly-constant sized proofs, etc), these have a strong limitation. Their security relies on complex heuristics [BR94], or strong non-falsifiable assumptions [Nao03]. As is standard practice in cryptography, proving security under standard (falsifiable) cryptographic assumptions is more desirable.

Unfortunately, there are known strong barriers to designing PCDs from standard falsifiable assumptions [GW11]. In light of these, it is natural to ask whether we could design PCDs, for any subset of general distributed computation, secure under standard assumptions. Recently, two beautiful works by Paneth-Pass [PP22] and Devadas et al. [DGKV22] designed IVC for all *deterministic computation* assuming hardness of standard assumptions. IVCs are a sub-class of PCDs, where distributed computation occurs in a straight-line (i.e., each user hands off partial computation state to the next user). Although not yet practical, these works design the first fully-succinct IVC from standard falsifiable assumptions. In this work, we continue this investigation of designing PCDs (beyond IVC) under standard assumptions.

This work. We introduce the notion of incrementally verifiable *streaming* computation (IVsC), as an improvement over IVCs. We also propose a new notion of privacy for IVsC. Our goal is to capture computations over *'streaming data'* while preserving data privacy. In words, IVsC can be used in applications, where data is not statically available, but comes in via an ongoing generation process. Similar to [PP22, DGKV22], we keep our focus on only *deterministic computation*.

We also introduce a new notion of non-interactive zero-knowledge (NIZK), that we call *step-by-step* zero-knowledge protocols. These protocols are standard NIZK systems, except they have two special features: (1) the NIZK proof can be incrementally computed, (2) the prover's internal state is also simulatable at any step during proof generation. That is, unlike standard NIZKs where only the final proof can be simulated, we can simulate prover's state after every computation step.[1]

We design IVsC (with privacy) from a wide variety of standard assumptions (decision linear, or sub-exponential DDH, or LWE). In addition, we design step-by-step ZK protocols starting from our IVsC protocol, obtaining them from same assumptions. We also briefly discuss another application

[1] E.g., consider a user proving correctness of PRF evaluation ($y = F_K(x)$) computed via the GGM-tree [GGM86], then using a step-by-step ZK protocol we can guarantee that the prover's state only leaks the partial PRF evaluation (i.e., after i PRG computations, only $F_K(x[1:i])$ could get leaked, instead of K).

in developing a new approach to prove distributed computations beyond path-graphs.

Organization. In the next section, we provide a high level overview of our main results and techniques. We discuss some motivating examples highlighting the benefits of supporting streaming inputs and various practical use cases of our designed systems in Section 6. In Section 2 we provide standard cryptographic preliminaries. In Section 3 we formally introduce and design IVsC with privacy. In Section 4 we describe step-by-step ZK and design it via a useful extension of IVsC, called *predicated zero-knowledge* IVsC. Finally, in Section 5 we discuss some more applications.

1.1 Overview

Consider the computation of a RAM machine $\mathcal{M}$ on an input x, where the computation of $\mathcal{M}(x)$ is very long (i.e., cannot be finished by a single user). An IVC protocol enables distribution of this computation across multiple users. In IVC, the $(u+1)$-th user takes the entire intermediate machine configuration cf_t (i.e., computation state and memory) from u-th user along with a succinct proof π_t, and continues the computation for some ℓ steps to generate updated configuration and proof, $(\mathsf{cf}_{t+\ell}, \pi_{t+\ell})$. Here cf_0 denotes the initial configuration (i.e., input x) and π_t proves that $\mathsf{cf}_0 \to \mathsf{cf}_t$ after t steps of $\mathcal{M}$. Importantly, the proof size, $|\pi_t|$, should never grow with t, and the running time for a single user should only grow with the number of computation steps it performs (i.e., ℓ for user $u+1$ in the above). In the context of *deterministic computations*, the verifier gets both, the original and final, configuration (e.g., cf_0, cf_t) along with π_t.

Streaming inputs and introducing IVsC. As described above, IVC requires the full input x to be part of the original machine configuration, cf_0. Thus, it cannot support incrementally certifying long computations, if the entire input is *not* available (or cannot be given) to the first user when the computation begins. To handle dynamic/streaming inputs, we consider a new streaming variant called incrementally verifiable *streaming* computation (IVsC).

In IVsC, each user u gets a local input x_u, in addition to the (intermediate) machine configuration cf_t and proof π_t. Rather than viewing the input x to be an implicit part of the configuration cf, we consider a separate input tape for $\mathcal{M}$ (in addition to its internal memory). Thus, $\mathcal{M}$ can read bits off of the input tape as well as its memory. One significant difference is that π_t now certifies that $\mathsf{cf}_0 \to \mathsf{cf}_t$ w.r.t. inputs $x_1, x_2, \ldots$, where x_i is local data/input available to the u-th user and cf_0 is some fixed initial configuration (e.g., all-zeros). Since we focus on deterministic computations (like [PP22, DGKV22]), thus the verifier receives the inputs $x_1, x_2, \ldots$ to verify the proof.

Prior works in IVC [PP22, DGKV22] (and RAM delegation [CJJ21], more broadly) also considered an explicit Digest algorithm to speed up verification. Digest takes as input a configuration and outputs a short digest. A verifier only needs the digest for the configurations, thus its running time could be independent of the input size. We consider a similar improvement in IVsC. Our Digest procedure produces a short digest for a *streaming* input $x_1, x_2, \ldots$; moreover, the digest is incrementally computable. That is, in addition to proof π_t, each user passes along the input digest (so far), and the next user updates the proof as well as the digest. Thus, π_t can be verified rather quickly (i.e., in time independent of the streaming input size) given the digest. An important aspect of the 'input digest', both in IVC and IVsC, is that the digest is independent of $\mathcal{M}$, thus it can be reused for verifying other computations on the same data. However, a soundness attacker still must output the full input, or otherwise, the computation becomes non-deterministic.

We formalize the IVsC framework in Section 3, and explain our design choices. Succinctness and soundness can be naturally extended from either IVC/PCD literature (for deterministic com-

putation). At a very high level, IVsC can be viewed as a PCD for straight-line graphs. (Note that since IVC does not support local data for intermediate users, thus IVC does not really correspond to PCDs for straight-line graphs.) Next, we sketch the main ideas behind our IVsC design.

IVsC via recursive proof batching. Our starting point is the elegant template developed by Paneth-Pass [PP22] and Devadas et al. [DGKV22]. Both works relied on a common master tool, called rate-1 somewhere extractable batch arguments (seBARGs)[2]. Rate-1 seBARGs are powerful batch argument systems that generate a succinct proof π proving the validity of a batch of k instances $x_1, \ldots, x_k$, given corresponding NP witnesses $\omega_1, \ldots, \omega_k$. These argument systems are very desirable for two reasons: (i) proof size, $|\pi|$, is equal to the size of a single witness, $|\omega_i|$, plus a fixed additive polynomial term and (ii) one can efficiently extract the NP witness, corresponding to a hidden *trapdoor* index, from any accepting proof.

While both [PP22, DGKV22] rely on rate-1 seBARGs, their IVC designs are somewhat different due to diverging intermediate abstractions developed in each work. Paneth-Pass introduced the notion of mergeable proofs of delegation, while Devadas et al. proposed the concept of hashed multi-hop seBARGs. We briefly summarize both of them, and refer the reader to [PP22, DGKV22] for a detailed overview.

Prior intermediate abstractions. Mergeable proofs [PP22] enable merging two delegation proofs into a single succinct proof. That is, given proof π_1, π_2 that certify $\mathsf{cf}_0 \to \mathsf{cf}_1$ and $\mathsf{cf}_1 \to \mathsf{cf}_2$ (resp.), using mergeable proofs these can be accumulated into a single proof π' of fixed polynomial size, certifying $\mathsf{cf}_0 \to \mathsf{cf}_2$. On the other hand, hashed multi-hop seBARGs generalize seBARGs to allow unbounded batching of batch proofs, with a special feature that the verifier only needs the digest of instances for verification, where the prover also generates this digest after each batching. That is, given k proofs $\pi_1, \ldots, \pi_k$ for $x_1, \ldots, x_k$, one generates a proof π and digest h such that verifier only needs (π, h), and k such proof-digest pairs can be further batched polynomially many times. Both works design these two objects by relying on rate-1 seBARGs.

Canonical IVC template. Given these tools, [PP22, DGKV22] follow a somewhat similar template for designing IVC. Each IVC proof contains λ RAM delegation proofs $\pi^1, \ldots, \pi^\lambda$, and λ configurations $\mathsf{cf}^1, \ldots, \mathsf{cf}^\lambda$. Each $\pi^i = \epsilon$ (i.e., empty), or it proves $\mathsf{cf}^i \to \mathsf{cf}^{i+1}$. The special feature of these is that if $\pi^i \neq \epsilon$, then it certifies correctness of exactly 2^i computation steps. That is, $\mathcal{M}$ starts from cf^i and reaches cf^{i+1} after 2^i steps. Now $\{\pi^i\}_i$ together certify $\mathcal{M}$ that $\mathsf{cf}_0 \to \mathsf{cf}_T$ (where cf_0 and cf_T are starting/ending configs). And, the way this works is as: suppose $T = \sum_i t_{i+1} 2^i$ (where t_i is the i-th least significant bit in binary representation of T), then for each i s.t. $t_i = 0$ we have that $\pi^i = \epsilon$, otherwise π^i certifies $\mathsf{cf}^i \to \mathsf{cf}^{i+1}$ (proving correctness of 2^i computation steps).

The advantage of designing IVC proofs as above is that it ensures proofs can be incrementally updated, yet their size does not grow with T (machine run-time). To understand a bit better, suppose a user is given an IVC proof certifying some T computation steps of $\mathcal{M}$, and it wants to run two more steps (i.e., $\mathsf{cf}_T \to \mathsf{cf}_{T+2}$). Given the above architecture, the user can first create a RAM delegation proof π' for certifying $\mathsf{cf}_T \to \mathsf{cf}_{T+2}$, and then do the following:

1. Start with $i = 1$.
2. Check if π^i is empty. If yes, it sets $\pi^i = \pi'$, appropriately updates the underlying configurations, and ends the routine. Else:
   (a) *"Accumulate"* π^i and π'. (Note each of these prove a computation of size 2^i.)

[2]Throughout, by rate-1 we mean almost-rate-1, where the proof to witness ratio is $1 + o(1)$ and not exactly 1.

(b) Update π' with the above accumulated proof. (Now π' proves a computation of size $2^i + 2^i = 2^{i+1}$.)
(c) Update corresponding configurations appropriately, increase i to $i+1$, and go back to step (2).

The above trick is colloquially referred to as the 'powers-of-two' trick and has been used in various contexts in cryptography [GKP+13, GHMR18, GV20]. In the context of IVC, it is extremely useful for incrementally updating RAM delegation proofs, while ensuring that the individual RAM delegation proofs (what we refer to as $\{\pi^i\}_i$) do not get "merged" [PP22] or "batched" [DGKV22] too often. Clearly, at any point when we combine RAM proofs, the underlying configurations go from being 2^i apart to being 2^{i+1} apart. Thus, next time the resulting proof has to be merged will be only after 2^{i+1} more computation steps have happened. This inherently guarantees desired succinctness for the proof size, and ensures updating a proof and verifying it can be done efficiently.

Soundness of IVC. Pass-Paneth execute the above template by using their mergeable proofs for *accumulation*, while Devadas et al. rely on hashed multi-hop seBARGs. The exact designs in both works slightly alter the above template to suit their intermediate object as well as to ensure the soundness proof goes through.

At an abstract level, we can view both their soundness proofs through a unified lens. We informally call it the *sliding window technique*, where the intuition is to view all T steps of computations (and their corresponding configurations) as a long linked list. Now the i-th element/node in the list corresponds to the computation that happens at the i-th step of $\mathcal{M}$'s computation. Recall at any step, $\mathcal{M}$ would read/write something from/on its memory, update the RAM machine state, and move the head along the memory. The core idea behind proving soundness is to fix a *constant-size* continuous sequence of nodes $\mathsf{cf}_i, \ldots, \mathsf{cf}_{i+w}$, and extract the corresponding configurations by using extractability feature of underlying seBARGs. We refer to this sequence of nodes as a *window*, where w is the window size. Given the extracted information, the reduction algorithm can argue that the all these configurations must be consistent[3] w.r.t $\mathcal{M}$ by relying on seBARG's (/intermediate object's) security. The next step is for the reduction to "*slide*" the window slightly in the forward direction. That is, say now it extracts $\mathsf{cf}_{i+1}, \ldots, \mathsf{cf}_{i+w+1}$ (i.e., moves it forward by one). Again, we can argue that these configurations must be consistent as before, and it appears that, by sliding the (extraction) window by one node at a time, we can argue that the starting and final configurations are consistent as per $\mathcal{M}$.

However, there are two caveats that require care. First, we need to guarantee that sliding the extraction window has only negligible drop in adversary's success; and second, when we move across two overlapping windows, then the extracted configurations that appear in both are the same. This is where the works rely on the famous *no-signaling* proof techniques [KRR14, CJJ21], and show that by devising a very specific sequence of hybrids and case analysis, soundness goes through.

While the above is mostly accurate, there is one final technical issue. The configuration size is too big, thus if we want to extract even a single configuration entirely, then the underlying RAM delegation/batch proof size must grow with the (maximum) configuration size. At first, this appears to be a major bottleneck, but it turns out there is rather simple and elegant solution for addressing it. Simply denote each node with a *'psuedo'*-configuration pcf, instead of the full configuration cf. A *psuedo*-configuration can be viewed as a specialized digest of the full configuration. Its

[3]Throughout, when we say two (or more) configurations $\mathsf{cf}_1, \mathsf{cf}_2$ $(, \ldots)$ are consistent, we mean that if you run $\mathcal{M}$ on cf_1, then you will reach cf_2 (and so on).

size is a-priori fixed (thus, independent of $|\mathsf{cf}_i|$), and one can succinctly check if two (adjacent) pseudo-configurations are consistent. For security, they guarantee that an attacker cannot generate two pairs of psuedo-configurations $(\mathsf{pcf}_1, \mathsf{pcf}_2)$ and $(\mathsf{pcf}'_1, \mathsf{pcf}'_2)$ such that they both are consistent, yet $\mathsf{pcf}_1 = \mathsf{pcf}'_1$ and $\mathsf{pcf}_2 \neq \mathsf{pcf}'_2$. To design such pseudo-configurations, prior works use special hash functions [HW15, OPWW15] implemented as Merkle trees. Combining all these ideas, one can avoid extracting the full configuration, and ensure the soundness proof goes through without blowing up the proof size. We refer the reader to [PP22, DGKV22] for a more detailed overview.

Our IVsC template. Our template builds upon the aforementioned IVC template to handle *streaming inputs*. Very briefly, the central reason the above IVC design cannot handle streaming inputs directly is that the input in IVC is programmed/hardwired inside the starting configuration. That is, since the input x is statically available, the first user simply treats it as part of the RAM memory. Thus, if some user (during the incremental computation) receives additional data/input, then it has to somehow merge it with its current configuration *while proving consistency*. The issue is that this significantly changes the pseudo-configuration, thus proving the pseudo-configurations are still consistent (succinctly) is challenging.

Our solution is to maintain a separate input tape, in addition to all other parts of the machine configuration. The advantage of separating this out would be: (1) we can add new data to the input tape dynamically, (2) it will not affect the current pseudo-configuration significantly[4]. However, just splitting (streaming) input and the full machine configuration is not enough. We need to ensure that an appropriate *succinct* digest for the input tape is added to the (new) pseudo-configuration, and this digest can be *incrementally* updated to support new incoming/streaming data. Finally, we also need to be careful that, despite these changes, the two necessary properties for pseudo-configurations still hold – (a) succinctly certifying two pseudo-configurations are adjacent, and (b) computational infeasibility of finding two *consistent* pseudo-configuration pairs with conflicting computation results.

At a high level, we achieve this by including λ input digests $h_1, \ldots, h_\lambda$ as an additional part of the incremental proof. Recall previously, they contained λ RAM delegation proofs and λ configurations. The purpose of maintaining these additional digests is to make sure that streaming inputs can be efficiently accumulated via the *powers-of-two* technique. Thus, the running time of update does not scale with the total size of the streaming input. Additionally, these digests can also be added to each pseudo-configuration pcf, and they will assist in succinctly proving adjacency of two pseudo-configurations. This is because each user (potentially) gets some new data/input, thus these digest values could significantly change when the user accumulates the latest input. To ensure this doesn't cause a problem, we prove that the input digests are also correctly updated by each prover. An important and desirable feature of the above input digests is that these can be reused for certifying different computations on the same data. Thus, if the same sequence of users run another machine $\mathcal{M}'$ on the given data, then the input digests will be consistent. Therefore, a verifier does not need to recompute the digests for a given sequence of streaming inputs every time.

Overall, these ideas could put together to generalize the IVC template to handle streaming inputs. And, although the soundness proof has to be sufficiently expanded (due to the changes in pseudo-configurations and additional tape headers), the core technical ideas still rely on the sliding window technique coupled with no-signaling proof strategies. Next, we discuss how to formalize and achieve a non-trivial notion of *privacy* for IVsC.

[4]There would be some changes which include storing additional tape header for the input tape, etc. But these can be appropriately handled, thus we ignore them for this overview.

Privacy (zero-knowledge) for IVsC. As mentioned earlier, defining privacy (zero-knowledge) for IVsC is a rather delicate task. Recall that a zero-knowledge proof for any deterministic computation is simply ϵ, an empty string. Because a verifier can verify its validity by performing the computation on its own, and an empty proof string is perfectly simulatable (vacuously). Since we study IVsC for deterministic computations, thus it is unclear if there exists a non-trivial notion of zero-knowledge. In words, *what can we hide from a malicious verifier that gets the (streaming) input for the (deterministic) computation?*

At first glance, it seems zero-knowledge for (incremental) deterministic computations is unlikely to be meaningful. However, it turns out that we indeed can formalize this non-trivially for IVsC (even IVC). Recall that a verifier gets a proof π and a short *digest* h as inputs, rather than the full input. This digest can either be statically computed for a fixed input (i.e., in IVC), or incrementally for a streaming input (i.e., in IVsC). Suppose we define privacy for the proof and digest *jointly* as follows:

$$\{(\pi, h, y = \mathcal{M}(x_1, x_2, \ldots)) : (\pi, h) \leftarrow \mathsf{IVsC.Prove}(\mathcal{M}, x_1, x_2, \ldots)\} \approx_c \mathtt{Sim}(\mathcal{M}, y) \tag{1}$$

The above states that an honestly computed proof-digest pair is indistinguishable from a simulated distribution, given the machine description and the output of the computation. Moreover, we can define this to hold *even for intermediate proofs, digests, and configurations.*

In other words, the guarantee is when a particular user passes off the IVsC proof, digests, and configurations after (say) T computation steps to the next user, then that user cannot learn anything about the previous inputs or intermediate configurations of $\mathcal{M}$, more than what is revealed by cf_T, i.e. the configuration of $\mathcal{M}$ after running it for T steps on the (streaming) input so far.

While the above formulation may seem ideal and almost the best-possible privacy guarantee that could be feasibly defined for incremental deterministic computations, there is one technical nuance. If we juxtapose the soundness definition and Eq. (1), then we realize the threat models are not fully overlapping. Soundness states that an adversary cannot generate $y, x_1, x_2, \ldots$ and π such that $y \neq \mathcal{M}(x_1, x_2, \ldots)$ but $(\mathsf{Digest}(x_1, x_2, \ldots), \pi)$ is accepted by the verifier. Notably, soundness attackers must output the *full* input. While Eq. (1) states that simulation only works for the proof and digest, and *not* the full input. This leaves a noticeable gap. Let us explain via an example.

An example. Consider a trusted authority that collects and signs large amounts of certain sensitive data. Due to resource constraints, it delegates the computation of a study/survey on this data to a cloud provider. Suppose it wants the cloud provider to publish the output publicly (e.g., census statistics and results) so that any individual user can verify the output. Now it might appear that we could readily rely on IVsC in this setting. The cloud runs IVsC to incrementally run the computation, generate a proof π, and digest h, and publishes these along with the output. Due to the zero-knowledge guarantee, we know that the proof-digest pair will not reveal too much about the data. But what about soundness!? We cannot rely on IVsC soundness to guarantee that a user will not accept the results, as the user does not have the entire data.

While it appears bypassing this would require all powerful succinct proofs for non-deterministic computations, we point out that (in many such applications) all a user might want to check is that the input digest h conforms to a publicly checkable predicate. That is, h is *well-formed*. E.g., the user just wants to check that every data was signed by the trusted (collection) authority. With this insight, we introduce a new notion that we call *predicate zero-knowledge* property for IVsC.

Predicated zero-knowledge for IVsC. Consider ϕ to be the predicate against which a verifier wants

to check data conformity. That is, the verifier wants to check that

$$\exists x_1, x_2, \ldots \textbf{ s.t. } y = \mathcal{M}(x_1, x_2, \ldots) \wedge (\phi(x_i) = 1)_i$$

Syntactically, we do not make any significant adjustments to the IVsC abstraction, except with one notable difference. Since we require the proof-digest pair (π, h) to hide everything about the streaming inputs, beyond their predicate satisfiability, thus the digest h can no longer be deterministically computed from just the inputs $(x_1, x_2, \ldots)$. Instead, we expect each digest h is accompanied with a predicate proof π_ϕ such that: (1) a verifier can efficiently check the underlying data conforms as per ϕ, and (2) π_ϕ can also be incrementally updated as new streaming data comes in. Soundness and privacy (in the form of zero-knowledge) can be appropriately generalized to fit the abstraction. We refer the reader to the main body for detailed descriptions. We remark that we also keep track of the randomness used in creating/updating these digests. This is useful for defining stronger soundness notions as well as enabling auditability of the digest computation by the data owner.

We believe the above gives us the best possible guarantee for capturing privacy for incremental proofs for deterministic computations. Along the way, it answers an important question left open by Ananth et al. [ADKL19] asking if one could design incremental proofs that enjoy both succinctness and privacy guarantees. Next, we explain the main ideas behind our construction.

Designing IVsC with predicated zero-knowledge. We explain our final template in two steps. First, we design an IVsC protocol with *empty predicates*. That is, we do not check well-formedness of the digest in the first step. While this is not ideal, we fix this in the second step. In the second step, we take our base protocol and extend it for arbitrary non-empty predicates.

Step 1: IVsC *for empty predicates.* In our previous IVsC template (without privacy), each proof contains λ RAM delegation proofs $(\pi_1, \ldots, \pi_\lambda)$, λ pseudo-configurations $(\mathsf{pcf}_1, \ldots, \mathsf{pcf}_\lambda)$, and λ digests $(h_1, \ldots, h_\lambda)$. Let us inspect why this does not already satisfy zero-knowledge as in Eq. (1). There are multiple reasons listed out below:

1. The digests and pseudo-configurations are (deterministic, Merkle-tree-style) hashes of the streaming input and configurations. Thus, they could reveal parts of the inputs and intermediate configurations.
2. The RAM delegation proofs themselves do not have any zero-knowledge guarantees.

In short, every component of the original proof leaks information about the inputs and computation so far. At a high level, our approach is two-fold: (a) commit to every input x_i and each configuration cf_j, and then hash down the corresponding commitments, and (b) equip RAM delegation proofs with necessary zero-knowledge properties by using NIZKs. The intuition is that by committing and then hashing the inputs/configurations, we can rely on hiding property of commitments. Similarly, by using NIZKs (inside RAM delegation proofs) we can prove correctness of each transition between two psuedo-configurations, while hiding the openings for the commitments.

The above serves as a good high level summary of the technical modifications needed. However, the exact composition of these have to done very carefully so that the soundness proof goes through. Notably, using NIZKs and commitments, we make all these digest and pseudo-configurations "*lossy*". By lossy, we mean that they may no longer satisfy statistical binding properties that are necessary for executing the sliding-window-style proof strategy. Moreover, once we plug in NIZKs, then recursive composition of RAM delegation proofs can *not* be done succinctly. Because

even if the NIZK proofs are very short, then we need to compose them recursively where the instance size grows significantly with each composition. However, as we detail in the main body, these issues can be handled by relying on hiding commitments with somewhere-extractability guarantees. And, more importantly, we use NIZKs to only prove consistency of two pseudo-configurations that are 2 steps apart. In other words, we realize that it is sufficient to just have the first RAM delegation proof, π_1, satisfy zero-knowledge properties. And, rest of the RAM delegation proofs can still be composed as before, without using NIZKs. The proof of zero-knowledge is much more simple, since we can start by simulating NIZK proof that show consistency of all pseudo-configurations that are 2 steps apart. Once we change these, we can accumulate them as an honest prover does, and we can finish the proof by using hiding security of all (extractable) commitments.

More details are provided later in Section 3. We remark that we could feasibly introduce a new intermediate abstraction (similar to mergeable proofs/hashed multi-hop BARGs [PP22, DGKV22]), equip it with stronger zero-knowledge-style guarantees, and then use ideas from our IVsC template described earlier. Instead, we decided to give a more direct construction for IVsC by relying on rate-1 seBARGs (and related primitives) in the main body. This is mostly because, unlike prior works [DGKV22, PP22], we focus on privacy of incremental proofs. And, capturing a sufficient notion of zero-knowledge for those intermediate abstractions turned out problematic. Briefly, as we hinted above, we need two different RAM delegation proofs (one with zero-knowledge property and one without for ensuring optimal accumulation), thus any possible intermediate abstraction would be unfortunately quite complicated and make the soundness proof a highly non-black-box process.

Step 2: *Extending to general predicates.* Finally, we show that we can upgrade the above IVsC protocol to generate predicate proofs along with the incremental proofs. Note that each predicate proof is almost like another incremental proof. That is, consider each prover computes two parallel IVsC proofs, where the first IVsC proof certifies computation of $\mathcal{M}(x_1, \ldots)$, while the second proof certifies computation of $((\phi(x_1) = 1) \wedge (\phi(x_2) = 1) \ldots)$. Note that if we generate both of these proofs using our above IVsC construction with empty predicates, then this offers a generic approach for designing IVsC for general predicates. We need some additional care in ensuring the soundness proof goes through. Because of the fact that if these two proofs are generated independently, then we cannot argue that their corresponding input digests are the same (since they are not deterministic). Thus, a black-box approach for extending to general predicates does fail. However, as we detail in Section 3.5, if we reuse the digests carefully, and rely on similar incremental computation techniques (as we do for our empty predicate IVsC), then the proofs of soundness, well-formedness of digest, and zero-knowledge work out. Moreover, we consider even more general predicates, where different portions of the streaming data could have separate predicates, and those could even be provided in a streaming fashion. Refer to Section 3.5 for more details.

The above concludes the overview of our IVsC protocols, and next we introduce the notion of step-by-step zero-knowledge protocols and discuss our main design ideas.

Step-by-step zero knowledge. The prover's running time in any NIZK proof system (for NP) is always larger than the time it takes to compute $\mathcal{R}(x, \omega)$, where $\mathcal{R}$ is the corresponding NP relation[5]. Consider a setting where $\mathcal{R}$ is a really long computation, or x and ω are being generated as part of a streaming computation. The question that we are interested in studying is: *can zero-knowledge proofs be incrementally computed?* Suppose the NIZK proof will be computed by more than one user (due to large runtime of $\mathcal{R}$), then can we obtain any *intermediary* zero-knowledge guarantees

[5]Otherwise, the NIZK system will not satisfy either correctness, soundness, or zero-knowledge.

for the computation state that is passed to the next prover? Or, can we obtain any protections against attackers that can corrupt a prover in the middle of the computation?

To address these, we introduce a new notion of NIZK systems, that we call step-by-step zero-knowledge protocols. We design an *incrementally computable zero-knowledge* protocol (ICZK) that gives us desired step-by-step ZK guarantees. We define ICZK as a regular NIZK system with two special features: (a) the NIZK proof is computed as an incremental proof (i.e., the proof size does not scale with runtime of $\mathcal{R}$), and (b) they satisfy *incremental zero-knowledge*. By incremental ZK, we mean that the following distributions (for any T) are indistinguishable:

$$\{\mathsf{state} : \mathsf{state} \leftarrow \mathsf{Prove}(x, \omega, T)\} \approx_c \{\mathtt{Sim}(\mathsf{cf}) : \mathsf{cf} = \mathcal{R}(x, w; T)\}$$

In the above, $\mathcal{R}(x, w; T)$ denotes running $\mathcal{R}$ on (x, ω) for T time steps, and state refers to the prover's state after running T steps of the NP validation relation, $\mathcal{R}$. It is straightforward to see that the above tightly captures the intuition behind a step-by-step zero-knowledge protocol. Because we are quite literally defining simulatability at every step of the computation. As we show next, we can design ICZK quite naturally from our (predicated) zero-knowledge IVsC protocols.

From zk-IVsC to ICZK. A natural idea would be to directly use zk-IVsC to incrementally run $\mathcal{R}$ on (x, ω) as the joint input. While the resulting scheme would satisfy our incremental ZK property, we are unable to prove its soundness. The reason is that there is no obvious 'meaningful' predicate that we could check on (x, ω) to ensure a cheating prover is really using a satisfying *witness* as part of the input, without revealing ω as part of the proof.

At a high level, our plan is to include a redundant encoding of the witness ω as an additional component of the proof, and then define the predicate to check that the witness portion of the input (in the zk-IVsC proof) is consistent with this encoding. To preserve zero-knowledge, we set the redundant encoding to be an encryption of ω. That is, the prover starts by encrypting ω to create a ciphertext $\mathsf{ct} = \mathsf{Enc}(\omega; R)$ for some randomness R. Then, we run the zk-IVsC prover while setting the predicate ϕ to check that ct is an encryption of the input witness ω. However, there is a major hurdle in executing this strategy. The predicate ϕ_{ct} is no longer a deterministic function. Encryption must be randomized for preserving zero-knowledge, thus we need to include the encryption randomness R as part of the zk-IVsC input.

Combining the above ideas, the ICZK prover starts by encrypting ω as $\mathsf{ct} = \mathsf{Enc}(\omega; R)$, and then sets (x, ω, R) as the input for zk-IVsC. With this, the ICZK prover runs the zk-IVsC protocol for computing $\mathcal{R}$ on (x, ω) portion of the input, and sets the predicate as $\phi_{\mathsf{ct}}(x, \omega, R) = (\mathsf{ct} \stackrel{?}{=} \mathsf{Enc}(\omega; R))$. While this change makes the predicate deterministic, and we can prove soundness of the resulting system, we have made it difficult to prove (incremental) zero-knowledge. This is because the randomness R that we use to encrypt ω is used as an input in the zk-IVsC protocol. Now to use semantic security of encryption, we have to argue that R is not leaked by the zk-IVsC portion of the proof. However, to argue that R is not leaked by the zk-IVsC proof, we need to make sure the predicate ϕ_{ct} is independent of its input (x, ω, R).

This circularity prevents us from proving (incremental) zero-knowledge for the above design. Our final observation is that we can break this circularity by relying on two special PKE schemes, one that is leakage-resilient and satisfies randomness recovery[6] [PW08], and other that supports randomness-dependent-message security [BCPT13]. The first type of PKE schemes is used to encrypt ω, while the second type is used to commit to the input (x, ω, R) as part of zk-IVsC. Together

[6]We also need it to be rate-1, but as we show later, the rate can be improved quite easily.

these both guarantee that our zk-IVsC protocol satisfies a leakage-resilient-style zero-knowledge guarantee, and the semantic security of ct can be used despite the circularity. Fortunately, we observe that these PKE can be designed from lossy trapdoor functions [PW08], thus we are able to instantiate our template under a wide variety of standard assumption. Although there are some more technical subtleties that we need to handle along the way, the above nicely summarizes the key challenges and circular issues. We refer to Section 4 for a more in-depth discussion.

New trade-offs in certifying general distributed computation via IVsC. Finally, we also briefly discuss another application for IVsC in certifying distributed computations beyond path graphs. While we do not design general PCDs for deterministic computations, we show that IVsC can be deployed in a wide variety of distributed computation networks to enable a non-trivial trade-off. We show that if we start with any arbitrary distributed computation network with say N computation nodes, then using IVsC we can certify the computation with a proof size of $\mathsf{poly}(\lambda, \log N)$. However, the total running time of the computation would grow linearly with N, rather than with the network diameter/depth (which could be as small as $\log N$). Our idea is to modify the (more general) distributed network by inserting a few artificial communication edges between different nodes, and use those to convert it into an almost path graph by relying on an appropriate topological sorting. We believe this could be interesting model for future research, thus sketch a construction in Section 5.

Related work. While the scope of PCD constructions from standard falsifiable assumptions [Nao03] is limited to IVC protocols for *deterministic computation* [DGKV22, PP22], there are several PCD constructions such as [CT10, BDFG21, CY21, KS22, HN23] for general distributed computations from *non-falsifiable* assumptions. These works can be split to two different templates: (1) PCDs via recursive composition of succinct non-interactive arguments of knowledge (SNARKs) [BCCT13, BCTV14, BSCTV17, COS20], and (2) more efficient PCDs via accumulation schemes in the random oracle model [BCMS20, BCL+21, KST22, BC23, BC24, BMNW24][7]. Moreover, there are a few works that construct PCDs with a transparent setup [BGH19, BCL+21, CCS22, CCG+23] without heuristically instantiating the random oracle in a non-black-box way.

2 Preliminaries

Notation We use overline for sets (e.g., for inputs to the RAM machine we let $\overline{z} = (z_k)_{k\in[K]}$ and $\overline{z}_i = (z_k)_{k\in[i]}$).

2.1 RAM Programs

A RAM program is typically defined as a deterministic machine $\mathcal{M}$ represented using a fixed polynomial-sized set of states that has random access to a large memory (where the memory includes an explicit input and rest of it is initialized as zeros). In this work, we use the following representation for RAM machines as it enables a simpler exposition of our main ideas.

Any RAM machine $\mathcal{M}$ that receives n bits of explicit input $x = \{0,1\}^n$, is associated with a work tape of size $S = S(n)$ and a fixed set of states Q that the machine can be in at any point during machine execution. Formally, a machine $\mathcal{M}$ is associated with following components:

[7] All these works additionally rely on public-key assumption with a notable exception of [BMNW24] which is limited to bounded depth accumulation.

- A set of machine states Q.
- A memory M of size $n + S$. Without loss of generality, we assume that the explicit input $x \in \{0,1\}^n$ is written in the first n cells of the memory (i.e., $x = (\mathsf{M}_1, \ldots, \mathsf{M}_n)$), and the rest of the memory (i.e., $(\mathsf{M}_{n+1}, \ldots, \mathsf{M}_{n+S})$) corresponds to the work-tape of the machine. For ease of exposition, we also use $\mathsf{W} = (\mathsf{W}[i])_{i\in[S]}$ to denote the work-tape. In our model, we view the work tape to be initialized as all zeros.
- A state transition function δ with the following syntax:

$$\delta : \mathrm{Q} \times \{0,1\} \to \mathrm{Q} \times \{0,1\}^{\log(n+S)} \times \{0,1\}^{\log S} \times \{0,1\}$$

  Here the transition function defines the execution of machine $\mathcal{M}$ at each time step.

There is a circuit $\mathbb{C}_{\mathcal{M}}$ that computes the next step as follows:

$$\mathbb{C}_{\mathcal{M}}(\mathrm{q}, \mathsf{rbit}) = (\mathrm{q}', \mathsf{ridx}', \mathsf{widx}, \mathsf{wbit})$$

We define the "configuration" of the RAM machine as the work part of memory, the current state, and the next reading index. Namely, we have $\mathsf{cf} = (\mathsf{W}, \mathrm{q}, \mathsf{ridx})$. Note that $|\mathsf{cf}| = S + \log|\mathrm{Q}| + \log(n+S)$.

We let out to be the output of the RAM machine and require the machine to write out at the first $n_{\mathsf{out}} = |\mathsf{out}|$ locations of the work tape W. Namely, we have $\mathsf{out} = \mathsf{W}[1, \ldots, n_{\mathsf{out}}] = \mathsf{cf}[1, \ldots, n_{\mathsf{out}}]$.

In this work, our focus is on streaming inputs $\overline{z} = (z_k)_{k\in[K]}$ for RAM machines. More specifically the machine is run by K users on their inputs where each user runs the machine for T steps. We use $\mathsf{cf}_{k,t}$ for the configuration of the machine at timestamp (k,t), i.e., after running for t steps by user k. We let $\mathsf{cf}_{1,0}$ be the initial configuration of the machine and $\mathsf{cf}_{k,0} = \mathsf{cf}_{k-1,T}$ for $k > 1$.

We denote by $\mathcal{M}(\mathsf{cf}_{k,t}, z_k; 1^{t'})$ running the machine $\mathcal{M}$ starting from the intermediate configuration $\mathsf{cf}_{k,t}$, on input z_k for t' steps. It follows by induction that

$$\mathsf{cf}_{k,t} = \mathcal{M}\left(\mathcal{M}\left(\cdots \mathcal{M}\left(\mathcal{M}(\mathsf{cf}_{1,0}, z_1; 1^T), z_2; 1^T\right)\cdots\right), z_k; 1^t\right).$$

We additionally use $\mathcal{M}((z_k)_{k\in[K]}; 1^T) = \mathsf{cf}_{K,T}$ for a complete computation of machine by K users.

The language of machine $\mathcal{M}$ is defined as follows:

$$\mathcal{L}_{\mathcal{M},T} = \left\{ (\overline{z}, \mathsf{out}) : \begin{array}{l} \mathsf{cf}_{1,0} = \text{initial config}, \mathsf{cf}_{K,T}[1, \ldots, n_{\mathsf{out}}] = \mathsf{out}, \\ \forall k \in [K] : \left(|z_k| = n \ \wedge \ \mathcal{M}(\mathsf{cf}_{k,0}, z_k; 1^T) = \mathsf{cf}_{k,T}\right) \end{array} \right\} \quad (2)$$

Remark 2.1. We let w.l.o.g. the initial configuration $\mathsf{cf}_{1,0}$ to be all zeros.

2.2 Rate-1 Block Somewhere Extractable Hash (SEH) Families

In what follows we recall a *simplified* definition of a block somewhere extractable hash (SEH) family that suffices for our applications.

Syntax. A block somewhere extractable hash family SEH consists of the following polynomial time algorithms:

$\mathsf{Gen}(1^\lambda, N, \Sigma, I) \to (\mathsf{hk}, \mathsf{td})$. This is a probabilistic setup algorithm that takes as input a security parameter 1^λ in unary, a number of blocks N, an alphabet Σ, and a subset $I \subseteq [N]$. It outputs a hash key hk along with trapdoor td.

$\mathsf{Hash}(\mathsf{hk}, x) \to \mathsf{v}$. This is a deterministic algorithm that takes as input a hash key hk and an input $x \in \Sigma^N$, and outputs a hash value v.

$\mathsf{Extract}(\mathsf{td}, \mathsf{v}, j) \to y$. This is a deterministic extraction algorithm that takes as input a trapdoor td, a hash value v, and an index $j \in [|I|]$ and outputs a block y.

We use the notation $\mathsf{Extract}(\mathsf{td}, \mathsf{v}) = (\mathsf{Extract}(\mathsf{td}, \mathsf{v}, j))_{j \in [|I|]}$.

Definition 2.2 (SEH)**.** A somewhere extractable hash family $\mathsf{SEH} = (\mathsf{Gen}, \mathsf{Hash}, \mathsf{Extract})$ is required to satisfy the following properties:

Efficiency. The size of the hash key hk and the hash value v is at most $|I| \cdot |\Sigma| \cdot \mathsf{poly}(\lambda)$.

Index hiding. For any PPT adversary $\mathcal{A}$, any polynomial $N = N(\lambda)$, and any $I_0, I_1 \subseteq [N]$ such that $|I_0| = |I_1|$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for every $\lambda \in \mathbb{N}$,

$$\Pr\left[\mathcal{A}(\mathsf{hk}) = b : \begin{array}{l} b \leftarrow \{0,1\} \\ (\mathsf{hk}, \mathsf{td}) \leftarrow \mathsf{Gen}(1^\lambda, N, \Sigma, I_b) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda),$$

Extraction correctness. For any $\lambda \in \mathbb{N}$, any $N \leq 2^\lambda$, any subset $I \subseteq [N]$, and any $x \in \Sigma^N$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for every $\lambda \in \mathbb{N}$,

$$\Pr\left[\ (x_i)_{i \in I} \neq \mathsf{Extract}(\mathsf{td}, \mathsf{v}, I) \ : \begin{array}{l} (\mathsf{hk}, \mathsf{td}) \leftarrow \mathsf{Gen}(1^\lambda, N, \Sigma, I) \\ \mathsf{v} \leftarrow \mathsf{Hash}(\mathsf{hk}, x) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

Remark 2.3. Note that the extraction correctness of a SEH family, implies the following properties:

Somewhere binding. For any $\lambda \in \mathbb{N}$, any $N \leq 2^\lambda$, any subset $I \subseteq [N]$, any index $i^* \in I$, and any (all powerful) adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for every $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l} \mathsf{Hash}(\mathsf{hk}, x) = \mathsf{v} \\ \wedge\ x_{i^*} \neq y_{i^*} \end{array} : \begin{array}{l} (\mathsf{hk}, \mathsf{td}) \leftarrow \mathsf{Gen}(1^\lambda, N, \Sigma, I), \\ (\mathsf{v}, x) \leftarrow \mathcal{A}(\mathsf{hk}), \\ (y_i)_{i \in I} = \mathsf{Extract}(\mathsf{td}, \mathsf{v}) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

Somewhere binding w.r.t. path opening. For any $\lambda \in \mathbb{N}$, $N = 2$, any $b \in [2]$, and any $x \in \Sigma^2$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for every $\lambda \in \mathbb{N}$ and any $\ell \in [\lambda]$,

$$\Pr\left[\begin{array}{l} \mathsf{VerifyAcc}(\overline{\mathsf{hk}}, \mathsf{v}, \rho, b) = 1 \\ \wedge\ \mathsf{v}_{0,b} \neq \mathsf{v}_0 \end{array} : \begin{array}{l} \big((\mathsf{hk}_d, \mathsf{td}_d) \leftarrow \mathsf{Gen}(1^\lambda, 2, \Sigma, b)\big)_{d \in [\ell]}, \\ (\mathsf{v}, \rho) \leftarrow \mathcal{A}(\overline{\mathsf{hk}}), \\ \mathsf{v}_0 = \mathsf{Extract}'((\mathsf{td}_d)_{d \in [\ell]}, \mathsf{v}) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

where $\overline{\mathsf{hk}} = (\mathsf{hk}_d)_{d \in [\ell]}$, $\rho = ((\mathsf{v}_{0,1}, \mathsf{v}_{0,2}), \ldots, (\mathsf{v}_{\ell-1,1}, \mathsf{v}_{\ell-1,2}))$, and we define $\mathsf{VerifyAcc}(\overline{\mathsf{hk}}, \mathsf{v}, \rho, b) = 1$ if for $d \in [\ell]$, $\mathsf{SEH.Hash}(\mathsf{hk}_d, (\mathsf{v}_{d-1,1}, \mathsf{v}_{d-1,2})) = \mathsf{v}_{d,b}$ where $\mathsf{v}_{\ell,b} = \mathsf{v}$. Additionally define $\mathsf{Extract}'((\mathsf{td}_d)_{d \in [\ell]}, \mathsf{v})$ to recursively compute $\mathsf{Extract}(\mathsf{td}_d, \mathsf{v}_d) = \mathsf{v}_{d-1}$ for $d \in [\ell]$, where $\mathsf{v}_\ell = \mathsf{v}$, and outputs v_0.

Definition 2.4 (Almost Rate-1 SEH)**.** A SEH scheme is said to be almost rate-1 if the hash value generated by $\mathsf{Hash}(\mathsf{hk}, x)$ is of length $(1 + c/\lambda) \cdot |I| \cdot |\Sigma| + \mathsf{poly}(\lambda)$.

Remark 2.5 ([CG])**.** Assuming almost rate-1 seBARGs there exists almost rate-1 SEH.

2.3 Rate-1 Somewhere Extractable Batch Arguments (seBARGs)

Syntax. A (publicly verifiable and non-interactive) somewhere extractable batch argument scheme seBARG for an NP language $\mathcal{L}$ (decided by relation $\mathcal{R}$) consists of the following polynomial time algorithms:

$\mathsf{Setup}(1^\lambda, k, n, i^*) \to \mathsf{crs}$. This is a probabilistic setup algorithm that takes as input a security parameter 1^λ, number of instances k, input length n, and an index $i^* \in [k]$. It runs in time at most $\mathsf{poly}(\lambda, n, \log k)$ and outputs a common reference string crs.

$\mathsf{Prove}(\mathsf{crs}, x_1, \ldots, x_k, w_1, \ldots, w_k) \to \pi$. This is a prover algorithm takes as input a crs, k instances $x_1, \ldots, x_k$ and corresponding witnesses $w_1, \ldots, w_k$, and outputs a proof π.

$\mathsf{Verify}(\mathsf{crs}, x_1, \ldots, x_k, \pi) \to 0/1$. The verification algorithm takes as input a common reference string crs, k instances x_i for $i \in [k]$, and a proof π. It outputs 0 (reject) or 1 (accept).

Definition 2.6 (seBARG)**.** A somewhere-extractable batch argument scheme $\mathsf{seBARG} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ for $\mathcal{L}$ is required to satisfy the following properties:

Efficiency. The size of the CRS and the proof is at most $\mathsf{poly}(\lambda, \log k, n, m)$, where m is the witness length.

Completeness. For any $\lambda \in \mathbb{N}$, and any $k = k(\lambda)$, $n = n(\lambda)$ of size at most 2^λ, any k instances $x_1, \ldots, x_k \in \mathcal{L}$, and their corresponding witnesses $w_1, \ldots, w_k \in \{0,1\}^m$, and any index $i^* \in [k]$,

$$\Pr\left[\mathsf{Verify}(\mathsf{crs}, x_1, \ldots, x_k, \pi) = 1 : \begin{array}{l} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, k, n, i^*), \\ \pi \leftarrow \mathsf{Prove}(\mathsf{crs}, x_1, \ldots, x_k, w_1, \ldots, w_k) \end{array}\right] = 1.$$

Index hiding. For any PPT adversary $\mathcal{A}$, any polynomials $k = k(\lambda)$ and $n = n(\lambda)$, and any indices $i_0, i_1 \in [k]$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that for every $\lambda \in \mathbb{N}$,

$$\Pr\left[b \leftarrow \mathcal{A}(\mathsf{crs}) : \begin{array}{l} b \leftarrow \{0,1\}, \\ \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, k, n, i_b) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

Somewhere Extraction. There exists a stateful PPT extractor $\mathcal{E}$ such that for any PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for any polynomials $k = k(\lambda)$ and $n = n(\lambda)$, and any index $i^* \in [k]$, for every $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l} \mathsf{Verify}(\mathsf{crs}, x_1, \ldots, x_k, \pi) = 1 \\ \wedge\ \mathcal{R}(x_{i^*}, w^*) \neq 1 \end{array} : \begin{array}{l} (\mathsf{crs}, \mathsf{td}) \leftarrow \mathcal{E}(1^\lambda, k, n, i^*) \\ (x_1, \ldots, x_k, \pi) \leftarrow \mathcal{A}(\mathsf{crs}) \\ w^* \leftarrow \mathcal{E}\left(\mathsf{td}, \{x_i\}_{i \in [k]}, \pi\right) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

Remark 2.7. We note that the somewhere extraction property implies the following *semi-adaptive soundness* property which asserts that for any PPT adversary $\mathcal{A}$, any polynomials $k = k(\lambda)$ and $n = n(\lambda)$, and any index $i^* \in [k]$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for every $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l} \mathsf{Verify}(\mathsf{crs}, x_1, \ldots, x_k, \pi) = 1 \\ \wedge\ x_{i^*} \notin \mathcal{L} \end{array} : \begin{array}{l} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, k, n, i^*) \\ (x_1, \ldots, x_k, \pi) \leftarrow \mathcal{A}(\mathsf{crs}) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

Definition 2.8 (Almost Rate-1 seBARG)**.** An seBARG scheme $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ is said to be almost rate-1 if the proof generated by $\mathsf{Prove}(\mathsf{crs}, x_1, \ldots, x_k, w_1, \ldots, w_k)$ is of length $(1 + c/\lambda)m + \mathsf{poly}(\lambda)$ for some constant c.

Remark 2.9 ([DGKV22, PP22, KLVW23])**.** Assuming either LWE/DLIN/sub-exponential DDH (and QR), there exists almost rate-1 seBARGs.

2.4 2-to-1 Collision Resistant Hash Function

Syntax. A 2-to-1 collision resistant hash function (CRHF) consists of the following polynomial time algorithms:

$\mathsf{Gen}(1^\lambda) \to \mathsf{hk}$**.** This is a probabilistic key generation algorithm that takes as input a security parameter 1^λ, and outputs a hash key hk.

$\mathsf{Hash}(\mathsf{hk}, x) \to \mathsf{h}$**.** This is the hashing algorithm that takes as input the hash key hk and an input $x \in \{0,1\}^{2\lambda}$, and outputs a hash value $\mathsf{h} \in \{0,1\}^\lambda$.

Definition 2.10 (CRHF)**.** A CRHF $\mathsf{H} = (\mathsf{Gen}, \mathsf{Hash})$ is required to satisfy the following property:

Collision Resistance. For any PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for every $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l} \mathsf{Hash}(x_1) = \mathsf{Hash}(x_1) \wedge \\ x_1 \neq x_2 \end{array} : \begin{array}{l} (\mathsf{hk}) \leftarrow \mathsf{Gen}(1^\lambda), \\ (x_1, x_2) \leftarrow \mathcal{A}(\mathsf{hk}) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

2.5 Hash Tree

Syntax. A hash tree consists of the following polynomial time algorithms:

$\mathsf{Gen}(1^\lambda) \to \mathsf{hk}$**.** This is a probabilistic key generation algorithm that takes as input a security parameter 1^λ, and outputs a hash key hk.

$\mathsf{Hash}(\mathsf{hk}, x) \to (\mathsf{rt}, \mathsf{tree})$**.** This is the hashing algorithm that takes as input the hash key hk and an input $x \in \{0,1\}^*$, and outputs a hash root rt and a hash tree tree.

$\mathsf{Read}(\mathsf{hk}, \mathsf{tree}, i) \to (b, \mathsf{rop})$**.** This is the hash opening algorithm that takes as input the hash key hk, a hash tree tree, and an index i, and outputs a bit b and a reading opening rop.

$\mathsf{Write}(\mathsf{hk}, \mathsf{tree}, i, b) \to (\mathsf{rt}', \mathsf{tree}', \mathsf{wop})$**.** This is the writing algorithm that takes as input the hash key hk, a hash tree tree, an index i and a bit b, and outputs a hash root rt', a hash tree tree', a writing opening wop.

$\mathsf{VfyRd}(\mathsf{hk}, \mathsf{rt}, i, b, \mathsf{rop}) \to 0/1$**.** This is a read-verification algorithm that takes as input the hash key hk, a hash root rt, an index i, a bit b and a reading opening rop, and outputs 0 (reject) or 1 (accept).

$\mathsf{VfyWt}(\mathsf{hk}, \mathsf{rt}, i, b, \mathsf{rt}', \mathsf{wop}) \to 0/1$**.** This is a write-verification algorithm that takes as input the hash key hk, a hash root rt, an index i, a bit b, a new hash root rt' and a writing opening wop, and outputs 0 (reject) or 1 (accept).

Definition 2.11 (Hash Tree)**.** A hash tree $\mathsf{HT} = (\mathsf{Gen}, \mathsf{Hash}, \mathsf{Read}, \mathsf{Write}, \mathsf{VfyRd}, \mathsf{VfyWt})$ is required to satisfy the following properties:

Efficiency. The size of hash key hk is at most $\mathsf{poly}(\lambda)$, the size of hash root rt is λ, and the size of openings is at most $\mathsf{poly}(\lambda, \log n)$ where n is the input size.

Reading Soundness. For any PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for every $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l|ll} \mathsf{VfyRd}(\mathsf{hk}, \mathsf{rt}, i, b^{(1)}, \mathsf{rop}^{(1)}) = 1 \wedge & : & \mathsf{hk} \leftarrow \mathsf{Gen}(1^\lambda), \\ \mathsf{VfyRd}(\mathsf{hk}, \mathsf{rt}, i, b^{(2)}, \mathsf{rop}^{(2)}) = 1 \wedge & & (\mathsf{rt}, i, b^{(1)}, \mathsf{rop}^{(1)}, b^{(2)}, \mathsf{rop}^{(2)}) \leftarrow \mathcal{A}(\mathsf{hk}) \\ b^{(1)} \neq b^{(2)} & & \end{array}\right] \leq \mathsf{negl}(\lambda).$$

Writing Soundness. For any PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for every $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l|ll} \mathsf{VfyWt}(\mathsf{hk}, \mathsf{rt}, i, b, \mathsf{rt}^{(1)}, \mathsf{wop}^{(1)}) = 1 \wedge & : & (\mathsf{hk}) \leftarrow \mathsf{Gen}(1^\lambda), \\ \mathsf{VfyWt}(\mathsf{hk}, \mathsf{rt}, i, b, \mathsf{rt}^{(2)}, \mathsf{wop}^{(2)}) = 1 \wedge & & (\mathsf{rt}, i, b, \mathsf{rt}^{(b)}, \mathsf{wop}^{(b)}, \mathsf{rt}^{(2)}, \mathsf{wop}^{(2)}) \leftarrow \mathcal{A}(\mathsf{hk}) \\ \mathsf{rt}^{(1)} \neq \mathsf{rt}^{(2)} & & \end{array}\right] \leq \mathsf{negl}(\lambda).$$

Remark 2.12. For all the algorithms and definitions above, instead of a single bit b at location i we can consider a set $(b_{i_j})_{i_j \in I}$ at locations in the set $I = (i_1, \ldots, i_t)$. Then the size of opening will grow linearly with t.

Remark 2.13. Reading soundness implies collision resistance of the hash tree.

Remark 2.14 ([Mer87])**.** Hash trees can be constructed using any family $\mathcal{F} : \{0,1\}^{2\lambda} \to \{0,1\}^\lambda$ of collision-resistance hash functions.

2.6 Non-Interactive Zero-Knowledge Arguments (NIZK)

Consider an NP language $\mathcal{L} = \{x \mid \exists w : \mathcal{R}(x, w) = 1\}$ defined w.r.t. a relation $\mathcal{R}$.

Syntax. A non-interactive zero-knowledge (NIZK) argument consists of the following polynomial time algorithms:

$\mathsf{Setup}(1^\lambda, 1^{n_x}, 1^{n_w}) \to \mathsf{crs}$**.** The probabilistic setup algorithm takes as input a security parameter λ, an instance length n_x, an witness length n_w, and outputs a common reference string crs.

$\mathsf{Prove}(\mathsf{crs}, x, w) \to \pi$**.** The prover algorithm takes as input a common reference string crs, an instance x, and a witness w and outputs a proof π.

$\mathsf{Verify}(\mathsf{crs}, x, \pi) \to 0/1$**.** The verifier algorithm takes as input a common reference string crs, an instance x, and a proof π. It outputs 0 (reject) or 1 (accept).

Definition 2.15 (NIZK)**.** A non-interactive zero-knowledge proof $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ for $\mathcal{L}$ is required to satisfy the following properties:

Completeness. For all $\lambda, n_x, n_w \in \mathbb{N}$ and $(x, w) \in \mathcal{R}$ where $|x| = n_x$ and $|w| = n_w$ we have:

$$\Pr[\mathsf{Verify}(\mathsf{crs}, x, \pi) = 1 : \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^{n_x}, 1^{n_w}), \pi \leftarrow \mathsf{Prove}(\mathsf{crs}, x, w)] = 1.$$

Adaptive Soundness. For any PPT adversary $\mathcal{A}$, there is a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda, n_x \in \mathbb{N}$:

$$\Pr\left[\begin{array}{l|ll} \mathsf{Verify}(\mathsf{crs}, x, \pi) = 1 \wedge & : & \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^{n_x}, 1^{n_w}), \\ x \notin \mathcal{L} & & (x, \pi) \leftarrow \mathcal{A}(\mathsf{crs}), |x| = n_x \end{array}\right] \leq \mathsf{negl}(\lambda).$$

Zero-Knowledge. There exists a stateful PPT simulator $\mathcal{S}$ such that for any PPT adversary $\mathcal{A}$, there is a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda, n_x, n_w \in \mathbb{N}$:

$$\begin{aligned} |\Pr[\mathcal{A}^{\mathsf{Prove}(\mathsf{crs},\cdot,\cdot)}(\mathsf{crs}) = 1 \ : \ \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^{n_x}, 1^{n_w})] - \\ \Pr[\mathcal{A}^{\mathcal{O}^{\mathcal{S}}(\cdot,\cdot)}(\mathsf{crs}) = 1 \ : \ \mathsf{crs} \leftarrow \mathcal{S}(1^\lambda, 1^{n_x}, 1^{n_w})]| \leq \mathsf{negl}(\lambda) \end{aligned}$$

where $\mathcal{O}^{\mathcal{S}}(x, w)$ outputs $\mathcal{S}(x)$ if $x \in \mathcal{L}$ and $\perp$ otherwise.

Knowledge Extractor. There exists a stateful PPT extractor $\mathcal{E}$ such that for any non-uniform PPT adversary $\mathcal{A}$, there is a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda, n_x, n_w \in \mathbb{N}$:

$$\Pr\left[\begin{array}{l|ll} \mathsf{Verify}(\widetilde{\mathsf{crs}}, x, \pi) = 1, & : & (\widetilde{\mathsf{crs}}, \mathsf{td}) \leftarrow \mathcal{E}(1^\lambda, 1^{n_x}, 1^{n_w}), \\ \wedge\ (\mathcal{R}(x, w) = 0 \ \vee \ |w| > n_w) & & (x, \pi) \leftarrow \mathcal{A}(\widetilde{\mathsf{crs}}), \\ & & |x| = n_x, \\ & & w \leftarrow \mathcal{E}(\mathsf{td}, x, \pi) \end{array}\right] \leq \mathsf{negl}(\lambda),$$

and $\widetilde{\mathsf{crs}}$ and $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^{n_x}, 1^{n_w})$ are computationally indistinguishable.

Remark 2.16 ([CW23, BWW23, BKP+23])**.** Assuming PKE and seBARGs there exists NIZKs.

2.7 (Customized) Public-Key Encryption System

Syntax. A public key encryption (PKE) scheme for the message space $\mathcal{M} = \{\mathcal{M}_\lambda\}_{\lambda \in \mathbb{N}}$ consists of the following polynomial time algorithms.

$\mathsf{Setup}(1^\lambda) \to (\mathsf{pk}, \mathsf{sk})$**.** The probabilistic setup algorithm takes as input a security parameter 1^λ and outputs the public and secret key pair $(\mathsf{pk}, \mathsf{sk})$.

$\mathsf{Enc}(\mathsf{pk}, m) \to \mathsf{ct}$**.** The probabilistic encryption algorithm takes as input the public key pk, a message $m \in \mathcal{M}_\lambda$, and outputs the ciphertext ct.

$\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) \to m'$**.** The decryption algorithm takes as input secret key sk, ciphertext ct, and outputs m'.

Definition 2.17 (PKE)**.** A public-key encryption system $(\mathsf{Setup}, \mathsf{Enc}, \mathsf{Dec})$ for $m \in \mathcal{M}_\lambda$ is required to satisfy the following properties:

Correctness. For any $\lambda \in \mathbb{N}$, $m \in \mathcal{M}_\lambda$, we have that $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) = m$ where $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, m)$ and $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\lambda)$.

Security. For any stateful PPT adversary $\mathcal{A}$, there is a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$:

$$\left|\Pr\left[1 \leftarrow \mathcal{A}^{\mathsf{Enc}(\mathsf{pk},\cdot)}(1^\lambda, \mathsf{pk})\right] - \Pr\left[1 \leftarrow \mathcal{A}^{\mathsf{Enc}(\mathsf{pk},0^{|m|})}(1^\lambda, \mathsf{pk})\right]\right| \leq \mathsf{negl}(\lambda)$$

where $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\lambda)$.

2.7.1 leakage resilient/RDM/randomness recoverable/rate-1 PKEs

Definition 2.18 (PKE with Randomness Recovery)**.** A public-key encryption system $(\mathsf{Setup}, \mathsf{Enc}, \mathsf{Dec})$ for $m \in \mathcal{M}_\lambda$ is said to have randomness recovery if the decryption algorithm additionally outputs the randomness, i.e., $\mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, m; r)) \to (m, r)$.

Definition 2.19 (Leakage Resilient PKE)**.** A public-key encryption system $(\mathsf{Setup}, \mathsf{Enc}, \mathsf{Dec})$ for $m \in \mathcal{M}_\lambda$ is said to be leakage resilient if for any stateful PPT adversary $\mathcal{A}$, there is a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$:

$$\left|\Pr\left[1 \leftarrow \mathcal{A}^{\mathsf{Enc}'(\mathsf{pk},\cdot,\cdot)}(1^\lambda, \mathsf{pk})\right] - \Pr\left[1 \leftarrow \mathcal{A}^{\mathsf{Enc}'(\mathsf{pk},0^{|m|},\cdot)}(1^\lambda, \mathsf{pk})\right]\right| \leq \mathsf{negl}(\lambda)$$

where $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\lambda)$ and $\mathsf{Enc}'(\mathsf{pk}, m, f)$ outputs $(\mathsf{Enc}(\mathsf{pk}, m; r), f(r))$.

Remark 2.20 ([PW08], §4.1)**.** Assuming lossy trapdoor functions (LTDFs) there exists a leakage resilient PKE with randomness recovery. The construction given in section 4.1 of [PW08] immediately implies randomness recovery. Moreover, in their security analysis, in addition to c_1 we can consider more leakage on the randomness, thus the min entropy of the randomness goes down by the length of the additional leakage. Therefore, we only need to increase the randomness size by the leakage size n_ℓ (λ bits in our applications) and use $(n + n_\ell, k + n_\ell)$-LTDFs. Note that LTDFs with better parameter can be constructed by combining two LTDFs.

Remark 2.21 ([PW08, FGK+10])**.** Assuming LWE, DLIN, and DDH there exists lossy trapdoor functions. Thus under the same set of assumptions there exists a leakage resilient PKE with randomness recovery.

Definition 2.22 (Rate-1 PKE)**.** A PKE scheme $(\mathsf{Setup}, \mathsf{Enc}, \mathsf{Dec})$ is said to be rate-1 if the ciphertext generated by $\mathsf{Enc}(\mathsf{pk}, m)$ is of length $|m| + \mathsf{poly}(\lambda)$

Remark 2.23 (Rate-1 leakage resilient PKE with randomness recovery.)**.** Assuming PRGs and PKEs, there exists rate-1 PKEs. Furthermore, if we start with a leakage resilient PKE with randomness recovery, we get a rate-1 leakage resilient PKE with randomness recovery. The construction works as follows: (1) Setup compute $\mathsf{pk}' = (\mathsf{pk}, G)$ where $\mathsf{pk} \leftarrow \mathsf{PKE.Setup}(\lambda)$ and G is a PRG with 3λ-bits seeds (that is secure if s has high entropy), (2) the encryption algorithm samples a PRG seed s, and computes $\mathsf{ct}_1 \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}, s)$, $\mathsf{ct}_2 = G(s) \oplus m$, and (3) the decryption algorithm first decrypts ct_1 to find s and then computes $m = \mathsf{ct}_2 \oplus G(s)$.

The security follows from the security of the underlying PKE and PRG. Additionally, if the underlying PKE is randomness recoverable, since the decryption algorithm also finds s, the resulting algorithm is also randomness recoverable. For leakage resilience we rely on leakage resilience of the underlying PKE, and high entropy of s given the leakage. Moreover, since the encrypted message is 3λ-bits long, the required randomness for PKE is $\mathsf{poly}(\lambda)$, so the resulting scheme is rate-1.

Definition 2.24 (Randomness-Dependent Message (RDM) PKE)**.** A public-key encryption system $(\mathsf{Setup}, \mathsf{Enc}, \mathsf{Dec})$ for $m \in \mathcal{M}_\lambda$ is said to be randomness-dependent message if for any stateful PPT adversary $\mathcal{A}$, there is a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$:

$$\left|\Pr\left[1 \leftarrow \exp_0^{\mathcal{A}}(1^\lambda, \mathsf{pk})\right] - \Pr\left[1 \leftarrow \exp_1^{\mathcal{A}}(1^\lambda, \mathsf{pk})\right]\right| \leq \mathsf{negl}(\lambda)$$

where definitions of $\exp_0^{\mathcal{A}}$ and $\exp_1^{\mathcal{A}}$ are provided in Figure 1.

$\exp_0^{\mathcal{A}}(1^\lambda, \mathsf{pk})$**.** This is the real experiment where $\mathcal{A}$ first receives $\mathsf{pk} \leftarrow \mathsf{Setup}(1^\lambda)$ from the challenger. Then $\mathcal{A}$ iteratively sends f_i to the challenger, receives $f_i(r)$, then sends m_i and receives $\mathsf{Enc}(\mathsf{pk}, m_i; r)$. After this, $\mathcal{A}$ outputs guess b'. Output b'.
$\exp_1^{\mathcal{A}}(1^\lambda, \mathsf{pk})$**.** This is the ideal experiment where $\mathcal{A}$ first receives $\mathsf{pk} \leftarrow \mathsf{Setup}(1^\lambda)$ from the challenger. Then $\mathcal{A}$ iteratively sends f_i to the challenger, receives $f_i(r)$, then sends m_i and receives $\mathsf{Enc}(\mathsf{pk}, 0^{|m_i|}; r)$. After this, $\mathcal{A}$ outputs guess b'. Output b'.

Figure 1: Real and ideal experiments for RDM-PKE security.

Remark 2.25 ([BCPT13])**.** Assuming PKE there exists an RDM-PKE.

3 Zero-Knowledge Incrementally Verifiable Streaming Computation (zk-IVsC)

In this section we first formally define $\mathsf{zk\text{-}IVsC}$, and construct and analyze our $\mathsf{zk\text{-}IVsC}$. We finally discuss how to further extend and optimize our $\mathsf{zk\text{-}IVsC}$.

3.1 Definition.

Syntax. A zero-knowledge incrementally verifiable streaming computation ($\mathsf{zk\text{-}IVsC}$) scheme for a RAM machine $\mathcal{M}$ consists of the following PPT algorithms:

$\mathsf{Gen}(1^\lambda, n, S, T) \to \mathsf{crs}$**.** The probabilistic setup algorithm takes as input a security parameter 1^λ, the input length n, the maximum configuration size S, and the running time of machine T, and outputs a common reference string crs.

$\mathsf{Update}(\mathsf{crs}, z_k, \mathsf{st}_{k-1}, \mathsf{hz}_{k-1}, \Pi_{k-1}; \mathsf{ps}_k) \to (\mathsf{st}_k, \mathsf{hz}_k, \Pi_k)$**.** The update algorithm takes as input a CRS crs, an input z_k, an intermediate state st_{k-1}, an intermediate digest hz_{k-1}, a proof Π_{k-1}, and some private state/randomness ps_k. It outputs an updated state st_k, an updated digest hz_k, and an updated proof Π_k. Note that this algorithm has RAM access to the memory.

NOTE. The update algorithm takes ps_k as explicit randomness, and it could use more randomness. However, we are not making the entire randomness explicit for simplicity.

$\mathsf{Digest}(\mathsf{crs}, \overline{z}, \overline{\mathsf{ps}}) \to \mathsf{hz}$. The digest algorithm takes as input a CRS crs, a set of inputs $\overline{z} = (z_k)_{k\in[K]}$, and a set of private states $\overline{\mathsf{ps}} = (\mathsf{ps}_k)_{k\in[K]}$, and outputs a hash digest hz.

$\mathsf{Verify}(\mathsf{crs}, (\mathsf{hz}, \mathsf{out}), \Pi) \to 0/1$. The verifier algorithm takes as input a CRS crs, a hash digest hz, an output out, and a proof Π, and outputs a bit to signal whether the proof is valid or not.

Remark 3.1 (Prover)**.** We will define the prover $\mathsf{Prove}(\mathsf{crs}, (z_k)_{k\in[K]}; (\mathsf{ps}_k)_{k\in[K]})$ for a zk-IVsC inductively as follows (where st_0 is the initial state and $\Pi_0 = \epsilon$):

If $K = 1$ then output $(\mathsf{st}_1, \mathsf{hz}_1, \Pi_1) \leftarrow \mathsf{Update}(\mathsf{crs}, z_1, \mathsf{st}_0, \Pi_0; \mathsf{ps}_1)$, otherwise inductively compute $(\mathsf{st}_{K-1}, \mathsf{hz}_{K-1}, \Pi_{K-1}) = \mathsf{Prove}(\mathsf{crs}, (z_k)_{k\in[K-1]}; \overline{\mathsf{ps}}_{K-1})$ and output $(\mathsf{st}_K, \mathsf{hz}_K, \Pi_K)$ that is computed as $(\mathsf{st}_K, \mathsf{hz}_K, \Pi_K) \leftarrow \mathsf{Update}(\mathsf{crs}, z_K, \mathsf{st}_{K-1}, \mathsf{hz}_{K-1}, \Pi_{K-1}; \mathsf{ps}_K)$.

Informal definition. Informally speaking, zk-IVsC requires that for a RAM computation $\mathcal{M}(\overline{z}) = \mathsf{cf}$ where $\mathsf{out} = \mathsf{cf}[1, \ldots, n_{\mathsf{out}}]$, there exists an efficient prover who generates a proof whose size is independent of computation time and space. Moreover, no PPT adversary can find (1) a collision: i.e. two different inputs whose digests are the same, (2) a valid proof w.r.t. an *honest* digest of $\overline{z}$ for some incorrect output out, (3) valid proofs $\Pi^{(0)}$ and $\Pi^{(1)}$ w.r.t. a unique digest hz for $\mathsf{out}^{(0)} \neq \mathsf{out}^{(1)}$, (4) any information about $\overline{z}$ from (hz, Π) more than what is revealed by cf.

Definition 3.2 (zk-IVsC)**.** A zero-knowledge incrementally verifiable streaming computation scheme $(\mathsf{Gen}, \mathsf{Update}, \mathsf{Digest}, \mathsf{Verify})$ for $\mathcal{M}$ has to satisfy the following properties:

Completeness. For every $\lambda, n, S, T \in \mathbb{N}$, any $\overline{z} = (z_k)_{k\in[K]}$ s.t. $z_k \in \{0,1\}^n$,

$$\Pr\left[\begin{array}{ll} (\overline{z}, \mathsf{out}) \in \mathcal{L}_{\mathcal{M},T} \wedge & \mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda, n, S, T) \\ \mathsf{hz} = \mathsf{Digest}(\mathsf{crs}, \overline{z}, \overline{\mathsf{ps}}) \wedge & : \ (\mathsf{st}, \mathsf{hz}, \Pi) \leftarrow \mathsf{Prove}(\mathsf{crs}, \overline{z}; \overline{\mathsf{ps}}) \\ \mathsf{Verify}(\mathsf{crs}, (\mathsf{hz}, \mathsf{out}), \Pi) = 1 & \text{find } \mathsf{cf} \in \mathsf{st}, \mathsf{out} = \mathsf{cf}[1, \ldots, n_{\mathsf{out}}] \end{array}\right] = 1.$$

Efficiency. For the completeness experiment above,

- Size of crs, and setup running time are $\mathsf{poly}(\lambda, \log S, \log n)$.
- Size of Π, and its verification time are $\mathsf{poly}(\lambda, \log S, \log n, \log T, \log K)$.
- Update's running time is $\mathsf{poly}(\lambda, n) + \mathsf{poly}(\lambda, S) + T \cdot \mathsf{poly}(\lambda, \log S, \log n, \log T, \log K)$. (We also consider - and later achieve - a more efficient update that runs in $T \cdot \mathsf{poly}(\lambda, \log S, \log n, \log T, \log K)$, when it is allowed to preprocess the n-bit input.)

Collision Resistance. For any stateful PPT adversary $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that for every $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{ll} \mathsf{hz}^{(0)} = \mathsf{hz}^{(1)} \wedge & (n, S, T) \leftarrow \mathcal{A}(1^\lambda), \mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda, n, S, T) \\ \overline{z}^{(0)} \neq \overline{z}^{(1)} & : \ (\overline{z}^{(b)}, \overline{\mathsf{ps}}^{(b)})_{b\in\{0,1\}} \leftarrow \mathcal{A}(\mathsf{crs}) \\ & (\mathsf{hz}^{(b)} = \mathsf{Digest}(\mathsf{crs}, \overline{z}^{(b)}, \overline{\mathsf{ps}}^{(b)}))_{b\in\{0,1\}} \end{array}\right] \leq \mathsf{negl}(\lambda).$$

Soundness. For any admissible stateful PPT adversary $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that for every $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{ll} (\overline{z}, \mathsf{out}) \notin \mathcal{L}_{\mathcal{M},T} \wedge & (n, S, T) \leftarrow \mathcal{A}(1^\lambda), \mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda, n, S, T) \\ \mathsf{Verify}(\mathsf{crs}, (\mathsf{hz}, \mathsf{out}), \pi) = 1 & : \ (\overline{z}, \mathsf{out}, \overline{\mathsf{ps}}, \Pi, 1^{K\cdot T}) \leftarrow \mathcal{A}(\mathsf{crs}), \mathsf{hz} = \mathsf{Digest}(\mathsf{crs}, \overline{z}, \overline{\mathsf{ps}}) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

where an adversary is admissible if $\forall k \in [K], z_k \in \{0,1\}^n$.

Strong Soundness. For any stateful PPT adversary $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that for every $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{ll} \mathsf{Verify}(\mathsf{crs},(\mathsf{hz},\mathsf{out}^{(0)}),\Pi^{(0)})=1 \wedge & (n,S,T)\leftarrow\mathcal{A}(1^\lambda),\\ \mathsf{Verify}(\mathsf{crs},(\mathsf{hz},\mathsf{out}^{(1)}),\Pi^{(1)})=1 \wedge & \mathsf{crs}\leftarrow\mathsf{Gen}(1^\lambda,n,S,T),\\ \mathsf{out}^{(0)}\neq\mathsf{out}^{(1)} & (\mathsf{hz},(\mathsf{out}^{(b)},\Pi^{(b)})_{b\in\{0,1\}},1^{K\cdot T})\leftarrow\mathcal{A}(\mathsf{crs}) \end{array}\right]\leq\mathsf{negl}(\lambda).$$

Remark 3.3. Strong soundness implies the normal soundness by having the challenger run $\mathcal{M}$ on $((z_k)_{k\in[K]})$ (received from the adversary), and generate (out^*,Π^*) corresponding to the correct computation. Additionally, strong soundness implies collision resistance property.

Zero-Knowledge. There exists a stateful PPT simulator $\mathcal{S}$ such that for any PPT adversary $\mathcal{A}$, there is a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{ll} b=b' & (n,S,T)\leftarrow\mathcal{A}(1^\lambda),b\leftarrow\{0,1\}\\ & \mathsf{crs}_0\leftarrow\mathsf{Gen}(1^\lambda,n,S,T),\mathsf{crs}_1\leftarrow\mathcal{S}(1^\lambda,n,S,T),\\ & \overline{z}\leftarrow\mathcal{A}(\mathsf{crs}_b),\mathsf{cf}=\mathcal{M}(\overline{z};1^T)\\ & (\mathsf{st}_0,\mathsf{hz}_0,\Pi_0)\leftarrow\mathsf{Prove}(\mathsf{crs}_0,\overline{z}),(\mathsf{st}_1,\mathsf{hz}_1,\Pi_1)\leftarrow\mathcal{S}(\mathsf{crs}_1,\mathsf{cf}),\\ & b'\leftarrow\mathcal{A}(\mathsf{st}_b,\mathsf{hz}_b,\Pi_b) \end{array}\right]\leq\mathsf{negl}(\lambda).$$

3.2 Construction

In this section we first define the notation, then present an overview of our construction, followed by the building blocks and the construction itself.

Notations. We implement our algorithms in the RAM model and to indicate a RAM access, we use *underline* in the construction. For ease of exposition (and w.l.o.g.) we assume T is a power of 2, every user has an input of size n, and the work tape is of size S.

For a RAM machine $\mathcal{M}$ we use $\mathsf{cf}=(\mathsf{W},\mathrm{q},\mathsf{ridx})$ to denote a configuration and $\mathsf{pcf}=(\mathsf{rt},\mathrm{q},\mathsf{ridx})$ to denote a pseudo-configuration, where W is the work tape, rt is a short digest of the work tape W (root of the hash tree computed on W), q is the state of the RAM machine, and ridx (reading index) is the pointer on the memory. For a configuration (resp. pseudo-configuration) that corresponds to user k after t steps of the computation, we use $\mathsf{cf}_{k,t}=(\mathsf{W}_{k,t},\mathrm{q}_{k,t},\mathsf{ridx}_{k,t})$ (resp. $\mathsf{pcf}_{k,t}=(\mathsf{rt}_{k,t},\mathrm{q}_{k,t},\mathsf{ridx}_{k,t})$). Note that we are splitting the memory into two parts, input tape and work tape, and often we denote the digest of the input tape by $\mathsf{rt}_{\mathsf{in}}$. Moreover, if $\mathsf{ridx}\leq n$ then $\mathcal{M}$ is reading the $\mathsf{ridx}^{\text{th}}$ index from the input tape, otherwise, $\mathcal{M}$ is reading the $(\mathsf{ridx}-n)^{\text{th}}$ index of the work tape.

Construction overview. We break our zk-IVsC proof to small steps, and show how each step builds upon the previous step. Our starting point is the fact the hash trees allow one to succinctly prove a single step of RAM computation. Namely, consider a RAM transition $\mathsf{cf}\xrightarrow{z}\mathsf{cf}'$ which consists of a single-bit reading from input/work tape z_i/W_i and a single-bit writing in the work tape. One can succinctly prove such a transition by generating a reading proof rop and a writing proof wop corresponding to the digest of the input/work tapes. Hence, $(\mathsf{rop},\mathsf{wop})$ is a proof for the RAM transition $\mathsf{pcf}\xrightarrow{\mathsf{rt}_{\mathsf{in}}}\mathsf{pcf}'$. Namely, we use $\mathsf{is\text{-}nxt\text{-}cnfg}_{\mathcal{M}}(\mathsf{hk}_{\mathsf{ht}},\mathsf{rt}_{\mathsf{in}},\mathsf{pcf},\mathsf{pcf}',\mathsf{rbit},\mathsf{rop},\mathsf{wop})\rightarrow 0/1$ (see Fig. 2) (where $\mathsf{hk}_{\mathsf{ht}}$ is a hash-key for a hash tree) as a function that efficiently checks the adjacency of two [consecutive] pseudo-configurations w.r.t. their corresponding set of reading and writing proofs $(\mathsf{rop},\mathsf{wop})$.

$\mathsf{is\text{-}nxt\text{-}cnfg}_{\mathcal{M}}(\mathsf{hk}_{\mathsf{ht}}, \mathsf{rt}_{\mathsf{in}}, \mathsf{pcf}, \mathsf{pcf}', \mathsf{rbit}, \mathsf{rop}, \mathsf{wop}) \to 0/1$

Compute the RAM step function as $\mathbb{C}_{\mathcal{M}}(\mathsf{q}, \mathsf{rbit}) = (\mathsf{q}^*, \mathsf{ridx}^*, \mathsf{widx}^*, \mathsf{wbit}^*)$. Then output 1 if all of following checks pass; output 0 otherwise.

Check reading correctness. Namely, rop certifies rbit is $\mathsf{ridx}^{\text{th}}$ index of the input/work tape:

- If $\mathsf{ridx} \leq n$ then $\mathsf{HT.VfyRd}(\mathsf{hk}_{\mathsf{ht}}, \mathsf{rt}_{\mathsf{in}}, \mathsf{ridx}, \mathsf{rbit}, \mathsf{rop}) = 1$, o.w. $\mathsf{HT.VfyRd}(\mathsf{hk}_{\mathsf{ht}}, \mathsf{rt}, \mathsf{ridx} - n, \mathsf{rbit}, \mathsf{rop}) = 1$.

Check writing correctness. Namely, wop certifies $\mathsf{pcf} \to \mathsf{pcf}'$:

- $(\mathsf{q}', \mathsf{ridx}') = (\mathsf{q}^*, \mathsf{ridx}^*)$ and $\mathsf{HT.VfyWt}(\mathsf{hk}_{\mathsf{ht}}, \mathsf{rt}, \mathsf{widx}^*, \mathsf{wbit}^*, \mathsf{rt}', \mathsf{wop}) = 1$.

Figure 2: Description of $\mathsf{is\text{-}nxt\text{-}cnfg}_{\mathcal{M}}$.

Next, we show how to make such a proof zero-knowledge by encrypting the pseudo-configurations and input digest and generating a NIZK proof π_{zk} for such a transition. Namely, we let $(\mathsf{ct}_{\mathsf{in}}, \mathsf{epcf}, \mathsf{epcf}')$ be encryptions of $(\mathsf{rt}_{\mathsf{in}}, \mathsf{pcf}, \mathsf{pcf}')$, and by $\mathsf{epcf} \xrightarrow{\mathsf{ct}_{\mathsf{in}}} \mathsf{epcf}'$ we denote there exist $(\mathsf{rt}_{\mathsf{in}}, \mathsf{pcf}, \mathsf{pcf}')$, s.t. $\mathsf{pcf} \xrightarrow{\mathsf{rt}_{\mathsf{in}}} \mathsf{pcf}'$. Thus we define $\mathcal{L}_{\mathsf{zk}}$ (see Fig. 3) to be the NIZK language for such a transition which checks the correctness of all the encryptions (w.r.t. an encryption public-key pk and the corresponding randomnesses), in addition to checking whether $\mathsf{pcf} \xrightarrow{\mathsf{rt}_{\mathsf{in}}} \mathsf{pcf}'$ holds.

Language $\mathcal{L}_{\mathsf{zk}}$

Hardwired: $\mathsf{pk}, \mathsf{hk}_{\mathsf{ht}}$
Instance: $x = (\mathsf{ct}_{\mathsf{in}}, \mathsf{epcf}, \mathsf{epcf}')$
Witness: $w = (\mathsf{rt}_{\mathsf{in}}, r_{\mathsf{in}}, \mathsf{pcf}, r, \mathsf{pcf}', r', \mathsf{rbit}, \mathsf{rop}, \mathsf{wop})$
Membership: w is a valid witness for $x \in \mathcal{L}_{\mathsf{zk}}$ if the following hold:

$\mathsf{ct}_{\mathsf{in}}$ **is the encryption of input root:** $\mathsf{PKE.Enc}(\mathsf{pk}, \mathsf{rt}_{\mathsf{in}}; r_{\mathsf{in}}) = \mathsf{ct}_{\mathsf{in}}$.
epcf **is the encryption of** pcf**:** $\mathsf{PKE.Enc}(\mathsf{pk}, \mathsf{pcf}; r) = \mathsf{epcf}$.
epcf' **is the encryption of** pcf'**:** $\mathsf{PKE.Enc}(\mathsf{pk}, \mathsf{pcf}'; r') = \mathsf{epcf}'$.
(rop, wop) **certify** $\mathsf{pcf} \to \mathsf{pcf}'$**:** $\mathsf{is\text{-}nxt\text{-}cnfg}_{\mathcal{M}}(\mathsf{hk}_{\mathsf{ht}}, \mathsf{rt}_{\mathsf{in}}, \mathsf{pcf}, \mathsf{pcf}', \mathsf{rbit}, \mathsf{rop}, \mathsf{wop}) = 1$.

Figure 3: Description of language $\mathcal{L}_{\mathsf{zk}}$.

We let $(\mathsf{v} = (\mathsf{epcf}, \mathsf{epcf}'), \pi_{\mathsf{zk}})$ to be a level 0 proof (w.r.t. $\mathsf{ct}_{\mathsf{in}}$), i.e. $\pi_0 \in \Pi$ in our zk-lVsC construction. Next we show how to iteratively merge any two proofs $\pi_{\ell-1,1}$ and $\pi_{\ell-1,2}$ (each for 2^ℓ steps of the computation) to get a proof π_ℓ (for $2^{\ell+1}$ steps of the computation). First note that by our choice of T (being a power of 2), a proof for T steps of the computation (performed by any user k) would only be a proof $\pi_{\log T-1}$. Thus until level $\ell-2$ we use the same $\mathsf{ct}_{\mathsf{in}}$ to generate/check the proofs, and only after that the $\mathsf{ct}_{\mathsf{in}}$ values are being merges. Therefore, in our construction, we only see h in the proofs π_ℓ for $\ell \geq \log T - 1$ (where $\mathsf{h} = \mathsf{ct}_{\mathsf{in}}$ at level $\log T - 1$). Moreover, every proof has two path opening from v to its leftmost and rightmost childs/epcfs. Since the leftmost and rightmost childs of v at level 0 are part of v itself, we do not consider path openings for π_0. Next we show how to merge two level $\ell-1$ proofs for $\ell \geq \log T$. Note that this is the more general case and includes all the checks we need to consider for merging.

First note that our construction samples separate SEH keys and seBARG CRSs for every level of merging, hence we use $\mathsf{crs}_{\mathsf{bp},\ell}$ and $\mathsf{hk}_{\mathsf{cf},\ell}$ for merging level $\ell-1$ proofs. Now consider two proofs $\pi_{\ell-1,1} = (\mathsf{v}_1, \rho_{1,1}, \rho_{1,2}, \mathsf{h}_1, \hat{\pi}_1)$ and $\pi_{\ell-1,2} = (\mathsf{v}_2, \rho_{2,1}, \rho_{2,2}, \mathsf{h}_2, \hat{\pi}_2)$. To merge such proofs, (1) we use a CRHF to hash $(\mathsf{h}_1, \mathsf{h}_2)$ to get t-h, (2) we use SEH to hash $(\mathsf{v}_1, \mathsf{v}_2)$ to get t-v, (3) we compute an

seBARG proof $\mathsf{t}\text{-}\pi^*$ for $\mathcal{L}_{\mathsf{bp},\ell}$ (see Fig. 4) that proves (a) the correctness of $\mathsf{t}\text{-}\mathsf{h}$ w.r.t. $(\mathsf{h}_1, \mathsf{h}_2)$, (b) the correctness of $\mathsf{t}\text{-}\mathsf{v}$ w.r.t. $(\mathsf{v}_1, \mathsf{v}_2)$, (c) the validity of $(\mathsf{v}_i, \mathsf{h}_i)$ w.r.t. $\hat{\pi}_i$, (d) the validity of $\rho_{i,1}$ and $\rho_{i,2}$ w.r.t. v, and (e) the equality of v_1 and v_2 at their intersection, i.e. the rightmost child of v_1 is the same as the leftmost child of v_2, and (4) we extend $\rho_{1,1}$ (resp. $\rho_{2,2}$) to get $\mathsf{t}\text{-}\rho_1$ (resp. $\mathsf{t}\text{-}\rho_2$), path openings to the leftmost (resp. rightmost) child of $\mathsf{t}\text{-}\mathsf{v}$.

Language $\mathcal{L}_{\mathsf{bp},\ell}$

Hardwired: $\mathsf{pk}, \mathsf{hk}_{\mathsf{ht}}, \mathsf{hk}_{\mathsf{ch}}, \mathsf{crs}_{\mathsf{zk}}, (\mathsf{crs}_{\mathsf{bp},j})_{j\in[\ell-1]}, (\mathsf{hk}_{\mathsf{cf},j})_{j\in[\ell]}$.
Instance: $x = (\mathsf{h}, \mathsf{v}, i)$.
Witness: $w = (\mathsf{v}_1, \mathsf{v}_2, \hat{w})$.
Membership: w is a valid witness for $x \in \mathcal{L}_{\mathsf{bp},\ell}$ if the following hold:

v **is the hash of** $(\mathsf{v}_1, \mathsf{v}_2)$**.** Namely, $\mathsf{SEH.Hash}(\mathsf{hk}_{\mathsf{cf},\ell}, (\mathsf{v}_1, \mathsf{v}_2)) = \mathsf{v}$.

$\hat{w}$ **certifies the validity of** v_i**.** Namely:

- If $\ell = 1$, then parse $\mathsf{v}_i = (\mathsf{epcf}, \mathsf{epcf}')$ and $\hat{w} = \pi$, and check $\mathsf{NIZK.Verify}(\mathsf{crs}_{\mathsf{zk}}, (\mathsf{h}, \mathsf{epcf}, \mathsf{epcf}'), \pi) = 1$.
- If $1 < \ell \leq \log T - 1$, then parse $\hat{w} = (\rho_1, \rho_2, \pi)$, and check $\mathsf{BARG.Verify}(\mathsf{crs}_{\mathsf{bp},\ell-1}, (\mathsf{h}, \mathsf{v}_i, j)_{j\in[2]}, \pi) = 1$.
- If $\log T \leq \ell$, then parse $\hat{w} = (\rho_1, \rho_2, \mathsf{h}_1, \mathsf{h}_2, \pi)$, and check $\mathsf{BARG.Verify}(\mathsf{crs}_{\mathsf{bp},\ell-1}, (\mathsf{h}_i, \mathsf{v}_i, j)_{j\in[2]}, \pi) = 1$ and $\mathsf{CRHF.Hash}(\mathsf{hk}_{\mathsf{ch}}, (\mathsf{h}_1, \mathsf{h}_2)) = \mathsf{h}$.

$\rho_i \in \hat{w}$ **is a valid path from** v_i**.** Namely, if $\ell > 1$ check $\mathsf{SEH.VerifyAcc}((\mathsf{hk}_{\mathsf{cf},d})_{d\in[\ell-1]}, \mathsf{v}_i, \rho_i, 3 - i)$.

Intersection of v_1 **and** v_2 **is equal.** Namely, for $i \in [2]$, if $\ell = 1$ parse $\mathsf{v}_i = (\mathsf{v}'_{i,1}, \mathsf{v}'_{i,2})$, otherwise, parse $\rho_i = ((\mathsf{v}_{0,i,1}, \mathsf{v}_{0,i,2}), \ldots, (\mathsf{v}_{\ell-2,i,1}, \mathsf{v}_{\ell-2,i,2}))$ and $\mathsf{v}_{0,i,3-i} = (\mathsf{v}'_{i,1}, \mathsf{v}'_{i,2})$. Then check $\mathsf{v}'_{1,2} = \mathsf{v}'_{2,1}$.

Figure 4: Description of language $\mathcal{L}_{\mathsf{bp},\ell}$.

Now we put everything together to get $\mathsf{t}\text{-}\pi = (\mathsf{t}\text{-}\mathsf{v}, \mathsf{t}\text{-}\rho_1, \mathsf{t}\text{-}\rho_2, \mathsf{t}\text{-}\mathsf{h}, \mathsf{t}\text{-}\pi^*)$, which in the next iteration, (if $\pi_\ell \in \Pi$ is not empty,) will be parsed as $\pi_{\ell,2}$ and get further merged with the already existing level ℓ proof in Π. This iterative process ends when $\pi_\ell \in \Pi$ is empty and then we let $(\epsilon, \ldots, \epsilon, \mathsf{t}\text{-}\pi, \pi_{\ell+1}, \ldots, \pi_\lambda)$ to be the new proof. The last piece of the puzzle is to generate a proof π_{out} for the actual output out of $\mathcal{M}$. We do this by generating a hash tree reading proof w.r.t. the final pcf in the computation/proof.

The verification checks the validity of all the proofs, and verifies every consecutive non-empty hash values $\mathsf{v} \in \pi_\ell$ and $\mathsf{v}' \in \pi_{\ell'}$ are equal in their intersecting epcf (w.r.t. their corresponding path openings), and the starting epcf in the proof is consistent with the initial configuration and the output is consistent with the last epcf in the proof (w.r.t. π_{out}). We summarize our notation in Table 1.

Building blocks and parameters. Let HT be a hash tree with output root of size λ on any input, CRHF be a collision resistant $\mathsf{Hash} : \{0,1\}^{2\lambda} \to \{0,1\}^{\lambda}$, seBARG be an almost rate-1 seBARG, SEH be an almost rate-1 SEH, NIZK be a NIZK, and PKE be a public-key encryption. We use the following parameters to instantiate such primitives. Let $|\mathsf{pcf}| = \lambda + |\mathsf{q}| + \log(n + S)$, $|\mathsf{epcf}| = |\mathsf{PKE.Enc}(\mathsf{pk}, 0^{|\mathsf{pcf}|})|$, and $n_{\mathsf{cf},0} = 2 \cdot |\mathsf{epcf}|$. For $\ell \in [\lambda]$, let $n_{\mathsf{cf},\ell} = |\mathsf{SEH.Hash}(\mathsf{hk}_{\mathsf{cf},\ell}, 0^{2 \cdot n_{\mathsf{cf},\ell-1}})|$, $n_{\mathsf{bp},\ell} = \lambda + n_{\mathsf{cf},\ell} + 1$, and $\Sigma^{\ell}_{\mathsf{cf}} = \{0,1\}^{n_{\mathsf{cf},\ell}}$ be the alphabet for block SEH. Moreover, let $|\mathsf{ct}_{\mathsf{in}}| = |\mathsf{PKE.Enc}(\mathsf{pk}, 0^{\lambda})|$, $|r|$ be the size of randomness used by PKE, and $n_{\mathsf{zk},x} = |\mathsf{ct}_{\mathsf{in}}| + 2|\mathsf{epcf}|$ and $n_{\mathsf{zk},w} = \lambda + 2|\mathsf{pcf}| + 3|r| + 1 + |\mathsf{rop}| + |\mathsf{wop}|$.

Table 1: zk-IVsC Noatation Summary

pk/sk	public-key/secret-key for encryption
$\mathsf{hk}_{\mathsf{ht}}/\mathsf{hk}_{\mathsf{ch}}$	hash-key for hash tree/collision-resistant hash function
$\mathsf{crs}_{\mathsf{zk}}/\mathsf{td}_{\mathsf{zk}}$	CRS/trapdoor for the NIZK scheme
$\mathsf{crs}_{\mathsf{bp},\ell}/\mathsf{td}_{\mathsf{bp},\ell}$	CRS/trapdoor for seBARG used to generate proofs at level ℓ
$\mathsf{hk}_{\mathsf{cf},\ell}/\mathsf{td}_{\mathsf{cf},\ell}$	hash-key/trapdoor for SEH used to hash level $\ell-1$ hashes of epcfs
$z/\mathsf{W}/\mathrm{q}$	RAM machine's input tape/work tape/state
$\mathsf{rbit}/\mathsf{ridx}$	RAM reading bit/reading index
$\mathsf{widx}/\mathsf{wbit}$	RAM writing bit/writing index
cf	configuration $= (\mathsf{W}, \mathrm{q}, \mathsf{ridx})$
pcf	pseudo-configuration $= (\mathsf{rt}, \mathrm{q}, \mathsf{ridx})$ where rt is hash root of W
epcf	encrypted pseudo-configuration $\leftarrow \mathsf{PKE.Enc}(\mathsf{pk}, \mathsf{pcf})$
$\mathsf{pcf}_{1,0}$	the universal starting pseudo-configuration
$\mathsf{rop}/\mathsf{wop}$	reading/writing proof w.r.t. hash roots
$\mathsf{rt}_{\mathsf{in}}/\mathsf{tree}_{\mathsf{in}}$	hash root/hash tree of RAM input z
$r_{\mathsf{in}}/\mathsf{ct}_{\mathsf{in}}$	randomness/ciphertext of $\mathsf{rt}_{\mathsf{in}}$, i.e. $\mathsf{ct}_{\mathsf{in}} = \mathsf{PKE.Enc}(\mathsf{pk}, \mathsf{rt}_{\mathsf{in}}; r_{\mathsf{in}})$
h	powers-of-2 hashes of $\mathsf{ct}_{\mathsf{in}}$ values, i.e. at level $\log T - 1$, $\mathsf{h} = \mathsf{ct}_{\mathsf{in}}$ and at any level $\ell \geq \log T$, $\mathsf{h} = \mathsf{CRHF.Hash}(\mathsf{hk}_{\mathsf{ch}}, (\mathsf{h}_1, \mathsf{h}_2))$ for level $\ell-1$ hashes h_1 and h_2
v	powers-of-2 hashes of epcf values, i.e. at level 0, $\mathsf{v} = (\mathsf{epcf}, \mathsf{epcf}')$ and at any level $\ell \geq 1$, $\mathsf{v} = \mathsf{SEH.Hash}(\mathsf{hk}_{\mathsf{cf},\ell}, (\mathsf{v}_1, \mathsf{v}_2))$ for level $\ell-1$ hashes v_1 and v_2
ρ	a path opening from hash value v to its rightmost or leftmost child
$\pi_\ell \in \Pi$	$= (\mathsf{v}, \rho_1, \rho_2, \mathsf{h}, \pi')$, a level ℓ proof in the powers-of-2 structure where h is empty if $\ell < \log T - 1$ and if $\ell = 0$, π' is a NIZK proof, o.w. it is a seBARG proof

Construction 3.4. In this construction, we always parse $\mathsf{hz} = (\mathsf{h}_{\log T-1}, \cdots, \mathsf{h}_\lambda)$, $\mathsf{crs} = (\mathsf{pk}, \mathsf{hk}_{\mathsf{ht}}, \mathsf{hk}_{\mathsf{ch}}, \mathsf{crs}_{\mathsf{zk}}, (\mathsf{crs}_{\mathsf{bp},\ell}, \mathsf{hk}_{\mathsf{cf},\ell})_{\ell\in[\lambda]}, \mathsf{pcf}_{1,0})$, $\Pi = (\pi_{\mathsf{out}}, \mathsf{epcf}, \pi_0, \ldots, \pi_\lambda)$, $\mathsf{pcf} = (\mathsf{rt}, \mathrm{q}, \mathsf{ridx})$, and $\mathsf{st}_i = (\mathsf{cf}_i, \mathsf{pcf}_i, r_i)$ wherever needed. Moreover, we let $\mathsf{st}_0 = (0^{|\mathsf{cf}|}, \mathsf{pcf}_{1,0}, 0^{|r|})$, and $\Pi_0 = (\epsilon, \mathsf{epcf}_{1,0}, \epsilon, \cdots, \epsilon)$, $\mathsf{hz}_0 = (\epsilon, \cdots, \epsilon)$ where $\mathsf{pcf}_{1,0}$ is as defined in Gen and $\mathsf{epcf}_{1,0} = \mathsf{PKE.Enc}(\mathsf{pk}, \mathsf{pcf}_{1,0}; 0^{|r|})$.

$\mathsf{Gen}(1^\lambda, n, S, T) \to \mathsf{crs}$**.** This algorithm lets $\alpha, \beta = 1^\lambda$[8] and does the following:

1. Sample parameters: $\mathsf{hk}_{\mathsf{ht}} \leftarrow \mathsf{HT.Gen}(1^\lambda)$, $\mathsf{hk}_{\mathsf{ch}} \leftarrow \mathsf{CRHF.Gen}(1^\lambda)$, $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{PKE.Gen}(1^\lambda)$, $\mathsf{crs}_{\mathsf{zk}} \leftarrow \mathsf{NIZK.Setup}(1^\lambda, 1^{n_{\mathsf{zk},x}}, 1^{n_{\mathsf{zk},w}})$ for $\mathcal{L}_{\mathsf{zk}}$, $(\mathsf{hk}_{\mathsf{cf},\ell}, \mathsf{td}_{\mathsf{cf},\ell}) \leftarrow \mathsf{SEH.Gen}(1^\lambda, 2, \Sigma_{\mathsf{cf}}^{\ell-1}, \alpha[\ell])$ for $\ell \in [\lambda]$, and $(\mathsf{crs}_{\mathsf{bp},\ell}, \mathsf{td}_{\mathsf{bp},\ell}) \leftarrow \mathsf{seBARG.Setup}(1^\lambda, 2, n_{\mathsf{bp},\ell}, \beta[\ell])$ for $\ell \in [\lambda]$.
2. *Get the honest starting pseudo-configuration:* $\mathsf{pcf}_{1,0} = (\mathsf{HT.Hash}(\mathsf{hk}_{\mathsf{ht}}, 0^S), 0^{|\mathrm{q}|}, 0^{|\mathsf{ridx}|})$.
3. Output $\mathsf{crs} = (\mathsf{pk}, \mathsf{hk}_{\mathsf{ht}}, \mathsf{hk}_{\mathsf{ch}}, \mathsf{crs}_{\mathsf{zk}}, (\mathsf{crs}_{\mathsf{bp},\ell}, \mathsf{hk}_{\mathsf{cf},\ell})_{\ell\in[\lambda]}, \mathsf{pcf}_{1,0})$.

$\underline{\mathsf{Update}}(\mathsf{crs}, z_k, \mathsf{st}_{k-1}, \mathsf{hz}_{k-1}, \Pi_{k-1}; \mathsf{ps}_k) \to (\mathsf{st}_k, \mathsf{hz}_k, \Pi_k)$**.** This algorithm does the following:

1. *Define the starting state for user k = last state for user $k-1$:* $(\mathsf{st}_{k,0}, \Pi_{k,0}) = (\mathsf{st}_{k-1}, \Pi_{k-1})$.

[8] While both α and β have fixed values in the construction, we specifically define them in the construction as we will change these values in the security proofs.

2. *Compute the work/input tape roots:* let $\mathsf{W}_{k,0} \in \mathsf{st}_{k,0}$, $(\mathsf{rt}_{k,0}, \mathsf{tree}_{k,0}) = \mathsf{HT.Hash}(\mathsf{hk}_{\mathsf{ht}}, \mathsf{W}_{k,0})$ and $(\mathsf{rt}_{\mathsf{in}}, \mathsf{tree}_{\mathsf{in}}) = \mathsf{HT.Hash}(\mathsf{hk}_{\mathsf{ht}}, z_k)$.
3. *Encrypt input root:* let $r_{\mathsf{in}} = \mathsf{ps}_k$ and compute $\mathsf{ct}_{\mathsf{in}} = \mathsf{PKE.Enc}(\mathsf{pk}, \mathsf{rt}_{\mathsf{in}}; r_{\mathsf{in}})$.
4. *Define input auxiliary information required for proof generation:* $\mathsf{aux}_{\mathsf{in}} = (\mathsf{rt}_{\mathsf{in}}, \mathsf{tree}_{\mathsf{in}}, r_{\mathsf{in}}, \mathsf{ct}_{\mathsf{in}})$.
5. *Run $\mathcal{M}$ and update its corresponding proof for T steps:* for $t \in [T]$ run $(\mathsf{st}_{k,t}, \mathsf{tree}_{k,t}, \Pi_{k,t}) = \underline{\mathsf{UpdateSingleStep}}(\mathsf{crs}, \mathsf{aux}_{\mathsf{in}}, \mathsf{st}_{k,t-1}, \mathsf{tree}_{k,t-1}, \Pi_{k,t-1})$.[9]
6. *Update the input digest:* $\mathsf{hz}_k = \mathsf{nxt\text{-}hz}(\mathsf{hk}_{\mathsf{ch}}, \mathsf{hz}_{k-1}, \mathsf{ct}_{\mathsf{in}})$.
7. output $(\mathsf{st}_{k,T}, \mathsf{hz}_k, \Pi_{k,T})$.

$\mathsf{Digest}(\mathsf{crs}, (z_k)_{k\in[K]}, (\mathsf{ps}_k)_{k\in[K]}) \to \mathsf{hz}$**.** This algorithm does the following:

1. Let $\mathsf{hz}_0 = (\mathsf{h}_{\log T-1}, \cdots, \mathsf{h}_\lambda)$ where h_ℓ is empty for $\ell \in [\log T - 1, \lambda]$.
2. For $k \in [K]$ compute root of z_k, then encrypt the root and update hz by adding the encrypted root to it; namely, compute the following:
   (a) $(\mathsf{rt}_k, \mathsf{tree}_k) = \mathsf{HT.Hash}(\mathsf{hk}_{\mathsf{ht}}, z_k)$, and $\mathsf{ct}_{\mathsf{in}} = \mathsf{PKE.Enc}(\mathsf{pk}, \mathsf{rt}_k; \mathsf{ps}_k)$.
   (b) $\mathsf{hz}_k = \mathsf{nxt\text{-}hz}(\mathsf{hk}_{\mathsf{ch}}, \mathsf{hz}_{k-1}, \mathsf{ct}_{\mathsf{in}})$.
3. Output $\mathsf{hz} = (\mathsf{h}_{\log T-1}, \cdots, \mathsf{h}_\lambda)$.

$\mathsf{nxt\text{-}hz}(\mathsf{hk}_{\mathsf{ch}}, \mathsf{hz}, \mathsf{t\text{-}h}) \to \mathsf{hz}'$

Let $\mathsf{hz} = (\mathsf{h}_{\log T-1}, \cdots, \mathsf{h}_\lambda)$, $\ell = \log T - 1$ and while h_ℓ is not empty iteratively do the following (1) let $\mathsf{h}^* = (\mathsf{h}_\ell, \mathsf{t\text{-}h})$, (2) compute $\mathsf{t\text{-}h} = \mathsf{CRHF.Hash}(\mathsf{hk}_{\mathsf{ch}}, \mathsf{h}^*)$, and (3) increment ℓ. Output $\mathsf{hz}' = (\epsilon, \ldots, \epsilon, \mathsf{t\text{-}h}, \mathsf{h}_{\ell+1}, \ldots, \mathsf{h}_\lambda)$.

Figure 5: Description of nxt-hz.

$\mathsf{Verify}(\mathsf{crs}, (\mathsf{hz}, \mathsf{out}), \Pi) \to 0/1$**.** This algorithm does the following:

1. Parse $\Pi = (\pi_{\mathsf{out}}, \mathsf{epcf}, \pi_0, \cdots, \pi_\lambda)$, $\mathsf{hz} = (\mathsf{h}_{\log T-1}, \cdots, \mathsf{h}_\lambda)$.
2. Verify that π_{out} proves out is the corresponding output of epcf. Namely, let $\pi_{\mathsf{out}} = (\mathsf{pcf}, r, \pi^*)$, and $\mathsf{pcf} = (\mathsf{rt}, \mathsf{q}, \mathsf{ridx})$, and check $\mathsf{HT.VfyRd}(\mathsf{hk}_{\mathsf{ht}}, \mathsf{rt}, [n_{\mathsf{out}}], \mathsf{out}, \pi^*) = 1$ and $\mathsf{PKE.Enc}(\mathsf{pk}, \mathsf{pcf}; r) = \mathsf{epcf}$; output 0 otherwise.
3. First verify all the λ proofs w.r.t. input digest hz, and then check the consistency of v_i values with the starting/ending/intersecting encrypted pseudo-configurations. Track such encrypted pseudo-configurations by epcf^* and at the beginning let $\mathsf{epcf}^* = \mathsf{epcf}$, i.e. the last encrypted pseudo-configuration in the proof. Then for $\ell \in [\log T - 1, \lambda]$, if π_ℓ is not empty do the following and output 0 if any of the checks fail:
   (a) Let $\pi_\ell = (\mathsf{v}, \rho_1, \rho_2, \mathsf{h}, \hat{\pi})$. Check $\mathsf{h} = \mathsf{h}_\ell$, $\mathsf{seBARG.Verify}(\mathsf{crs}_{\mathsf{bp},\ell}, (\mathsf{v}, \mathsf{h}, i)_{i\in[2]}, \hat{\pi}) = 1$, and $\mathsf{SEH.VerifyAcc}((\mathsf{hk}_{\mathsf{cf},d})_{d\in[\ell]}, \mathsf{v}, \rho_i, i)$ for $i \in [2]$.
   (b) For i$\in$ [2] parse $\rho_i = ((\mathsf{v}_{0,i,1}, \mathsf{v}_{0,i,2}), \ldots, (\mathsf{v}_{\ell-1,i,1}, \mathsf{v}_{\ell-1,i,2}))$, $\mathsf{v}_{0,i,i} = (\mathsf{epcf}_{i,1}, \mathsf{epcf}_{i,2})$, and check $\mathsf{epcf}_{2,2} = \mathsf{epcf}^*$. Let $\mathsf{epcf}^* = \mathsf{epcf}_{1,1}$.
4. Output 1 if $\mathsf{epcf}^* = \mathsf{PKE.Enc}(\mathsf{pk}, \mathsf{pcf}_{1,0}; 0^{|r|})$ and 0 otherwise.

[9] Recall that by underlining an algorithm we mean that it can be even more efficiently implemented as a RAM algorithm. We explain this further in Section 3.3.

$\underline{\mathsf{UpdateSingleStep}}(\mathsf{crs}, \mathsf{aux}_{\mathsf{in}}, \mathsf{st}_{t-1}, \mathsf{tree}_{t-1}, \Pi_{t-1}) \to (\mathsf{st}_t, \mathsf{tree}_t, \Pi_t)$. This algorithm does the following:

1. Parse $\Pi_{t-1} = (\pi_{\mathsf{out}}, \mathsf{epcf}_{t-1}, \pi_0, \cdots, \pi_\lambda)$, and $\mathsf{aux}_{\mathsf{in}} = (\mathsf{rt}_{\mathsf{in}}, \mathsf{tree}_{\mathsf{in}}, r_{\mathsf{in}}, \mathsf{ct}_{\mathsf{in}})$.
2. Find $(\mathsf{cf}_t, \mathsf{pcf}_t)$, and the reading/writing proofs $(\mathsf{rop}, \mathsf{wop})$ for RAM transition $\mathsf{pcf}_{t-1} \to \mathsf{pcf}_t$ as follows:
   (a) *Get read-proof from input tape:* $(\mathsf{rbit}, \mathsf{rop}) = \underline{\mathsf{HT.Read}}(\mathsf{hk}_{\mathsf{ht}}, \mathsf{tree}_{\mathsf{in}}, \mathsf{ridx}_{t-1})$ if $\mathsf{ridx}_{t-1} \le n$.
   (b) *Get read-proof from work tape:* $(\mathsf{rbit}, \mathsf{rop}) = \underline{\mathsf{HT.Read}}(\mathsf{hk}_{\mathsf{ht}}, \mathsf{tree}_{t-1}, \mathsf{ridx}_{t-1} - n)$ if $\mathsf{ridx}_{t-1} > n$.
   (c) *Run the RAM step function:* $\mathbb{C}_{\mathcal{M}}(\mathrm{q}_{t-1}, \mathsf{rbit}) = (\mathrm{q}_t, \mathsf{ridx}_t, \mathsf{widx}, \mathsf{wbit})$.
   (d) *Get write-proof for work tape:* $(\mathsf{rt}_t, \mathsf{tree}_t, \mathsf{wop}) = \underline{\mathsf{HT.Write}}(\mathsf{hk}_{\mathsf{ht}}, \mathsf{tree}_{t-1}, \mathsf{widx}, \mathsf{wbit})$.
   (e) *Update the work tape:* $\underline{\text{write}}$ wbit at $\mathsf{widx}^{\text{th}}$ index of W_{t-1} to get W_t.
   (f) *Find* $(\mathsf{cf}_t, \mathsf{pcf}_t)$*:* Let $\mathsf{cf}_t = (\mathsf{W}_t, \mathrm{q}_t, \mathsf{ridx}_t)$ and $\mathsf{pcf}_t = (\mathsf{rt}_t, \mathrm{q}_t, \mathsf{ridx}_t)$.
3. *Encrypt* pcf_t*:* sample r_t and compute $\mathsf{epcf}_t = \mathsf{PKE.Enc}(\mathsf{pk}, \mathsf{pcf}_t; r_t)$.
4. *Compute a NIZK proof for* $\mathsf{epcf}_{t-1} \to \mathsf{epcf}_t$*:* $\pi_{\mathsf{zk}} = \mathsf{NIZK.Prove}\left(\mathsf{crs}_{\mathsf{zk}}, x = (\mathsf{ct}_{\mathsf{in}}, \mathsf{epcf}_{t-1}, \mathsf{epcf}_t), w\right)$ for language $\mathcal{L}_{\mathsf{zk}}$ where $w = (\mathsf{rt}_{\mathsf{in}}, r_{\mathsf{in}}, \mathsf{pcf}_{t-1}, r_{t-1}, \mathsf{pcf}_t, r_t, \mathsf{rbit}, \mathsf{rop}, \mathsf{wop})$.
5. *Define the temporary proof and input hash:* let $\mathsf{t}\text{-}\pi = \left(\mathsf{t}\text{-}\mathsf{v} = (\mathsf{epcf}_{t-1}, \mathsf{epcf}_t), \mathsf{t}\text{-}\hat{w} = \pi_{\mathsf{zk}}\right)$, and $\mathsf{t}\text{-}\mathsf{h} = \mathsf{ct}_{\mathsf{in}}$. Note that at the moment these values correspond to a level 0 proof in the power-of-2-structure, but they will be treated as proofs for level ℓ, in the next iterative process.
6. *Iteratively merge the proofs:* let $\ell = 0$ and while $\pi_\ell \in \Pi$ is not empty run the following iterative process to merge the proofs π_ℓ and $\mathsf{t}\text{-}\pi$ (which are at the same level) and increment ℓ at the end.
   (a) Parse $\mathsf{t}\text{-}\pi$ and π_ℓ as follows:
   - If $\ell = 0$ then $\pi_\ell = (\mathsf{v}_1, \hat{w}_1)$ and $\mathsf{t}\text{-}\pi = (\mathsf{v}_2, \hat{w}_2)$.
   - If $0 < \ell < \log T - 1$ then $\pi_\ell = (\mathsf{v}_1, \rho_{1,1}, \rho_{1,2}, \hat{\pi}_1)$ and $\mathsf{t}\text{-}\pi = (\mathsf{v}_2, \rho_{2,1}, \rho_{2,2}, \hat{\pi}_2)$.
   - If $\log T - 1 \le \ell$ then $\pi_\ell = (\mathsf{v}_1, \rho_{1,1}, \rho_{1,2}, \mathsf{h}_1, \hat{\pi}_1)$ and $\mathsf{t}\text{-}\pi = (\mathsf{v}_2, \rho_{2,1}, \rho_{2,2}, \mathsf{h}_2, \hat{\pi}_2)$.

   (b) *Merge level* ℓ *hashes to get the next temp* v*:* $\mathsf{t}\text{-}\mathsf{v} = \mathsf{SEH.Hash}(\mathsf{hk}_{\mathsf{cf},\ell+1}, (\mathsf{v}_1, \mathsf{v}_2))$.
   (c) *Merge level* ℓ *input hashes to get the next temp* h*:* if $\ell \ge \log T - 1$ compute $\mathsf{t}\text{-}\mathsf{h} = \mathsf{CRHF.Hash}(\mathsf{hk}_{\mathsf{ch}}, (\mathsf{h}_1, \mathsf{h}_2))$.
   (d) *Find partial witness* $\hat{w}_i$ *for merging BARG proofs*: for $i \in [2]$, if $1 \le \ell < \log T$ then $\hat{w}_i = (\rho_{1,2}, \rho_{2,1}, \hat{\pi}_i)$, and if $\log T \le \ell$ then $\hat{w}_i = (\rho_{1,2}, \rho_{2,1}, \mathsf{h}_i, \hat{\pi}_i)$.
   (e) *Merge NIZK/BARG proofs:* $\mathsf{t}\text{-}\pi^* = \mathsf{seBARG.Prove}\left(\mathsf{crs}_{\mathsf{bp},\ell+1}, \begin{array}{l}(x_i = \mathsf{t}\text{-}\mathsf{h}, \mathsf{t}\text{-}\mathsf{v}, i)_{i\in[2]} \\ (w_i = \mathsf{v}_1, \mathsf{v}_2, \hat{w}_i)_{i\in[2]}\end{array}\right)$ for the language $\mathcal{L}_{\mathsf{bp},\ell+1}$ in Fig. 4.
   (f) *Extend the path openings to get the next temp path openings:* for $i \in [2]$ if $\ell = 0$ then $\mathsf{t}\text{-}\rho_i = (\mathsf{v}_1, \mathsf{v}_2)$, and if $\ell \ge 1$ then $\mathsf{t}\text{-}\rho_i = (\rho_{i,i}, (\mathsf{v}_1, \mathsf{v}_2))$.
   (g) *Put everything together to get the next temp* π*:* if $\ell < \log T - 2$ then $\mathsf{t}\text{-}\pi = (\mathsf{t}\text{-}\mathsf{v}, \mathsf{t}\text{-}\rho_1, \mathsf{t}\text{-}\rho_2, \mathsf{t}\text{-}\pi^*)$, otherwise, $\mathsf{t}\text{-}\pi = (\mathsf{t}\text{-}\mathsf{v}, \mathsf{t}\text{-}\rho_1, \mathsf{t}\text{-}\rho_2, \mathsf{t}\text{-}\mathsf{h}, \mathsf{t}\text{-}\pi^*)$.
7. *Compute output-proof w.r.t.* epcf_t*:* $(\mathsf{out}_t, \pi_t^*) = \mathsf{HT.Read}(\mathsf{hk}_{\mathsf{ht}}, \mathsf{tree}_t, [n_{\mathsf{out}}])$ and $\pi_{\mathsf{out}} = (\mathsf{pcf}_t, r_t, \pi_t^*)$.
8. Output $(\mathsf{st}_t, \mathsf{tree}_t, \Pi_t = (\pi_{\mathsf{out}}, \mathsf{epcf}_t, \epsilon, \ldots, \epsilon, \mathsf{t}\text{-}\pi, \pi_{\ell+1}, \ldots, \pi_\lambda))$.

3.3 Analysis

Theorem 3.5. Assuming that seBARG is an almost rate-1 seBARG (Definitions 2.6 and 2.8), NIZK is a secure NIZK protocol (Definition 2.15), PKE is a secure PKE system (Definition 2.17), SEH is an almost rate-1 SEH (Definitions 2.2 and 2.4), HT is a hash tree (Definition 2.11), and CRHF is a 2-to-1 collision resistant hash function (Definition 2.10), Construction 3.4 is a zk-IVsC (Definition 3.2).

Corollary 3.6. Assuming almost rate-1 seBARGs and PKEs, Construction 3.4 is a zk-IVsC.

Corollary 3.7. Assuming either LWE, or DLIN, or sub-exponential DDH, Construction 3.4 is a zk-IVsC.

The zero-knowledge property of our construction follows by NIZK zero-knowledge and PKE security in a straightforward manner, however, the soundness proof is rather a delicate task. Next, we provide an overview of the strong soundness of our construction and refer the reader to Appendix A for a complete analysis of our construction.

Strong soundness overview. Define $\exp_{\alpha,\beta}$ to be an experiment w.r.t. the choice of α and β (where SEH (resp. seBARG) is binding/extractable on the path α (resp. β)), that also outputs the trapdoors and secret keys. Note that α uniquely determines a tuple $(\mathsf{cf}_{k,t-1}, \mathsf{cf}_{k,t})$ in the computation (and its corresponding path in the proof). Define an extraction algorithm $\mathsf{Extract}^*$ that by (recursively) using the SEH trapdoors and the PKE secret-key, extracts some $(\mathsf{pcf}^{(b)}_{k,t-1}, \mathsf{pcf}^{(b)}_{k,t})$ from proof $\Pi^{(b)}$. Now note that any $\mathsf{epcf}_{k,t}$ is in exactly two different path openings, one that branches out to right and one that branches out to left. Now, define two other extraction algorithms $\mathsf{Extract}_r$ (resp. $\mathsf{Extract}_l$) that first recursively use the seBARG trapdoors, then use path openings from right (resp. left), and finally uses the PKE secret-key to extract some $\mathsf{pcf}^{(b)}_r$ (resp. $\mathsf{pcf}^{(b)}_l$) from proof $\Pi^{(b)}$, where (k,t) is also defined by the value of α. Our goal is to inductively show that if $\mathsf{pcf}^{(0)}_{k,t-1} = \mathsf{pcf}^{(1)}_{k,t-1}$, then $\mathsf{pcf}^{(0)}_{k,t} = \mathsf{pcf}^{(1)}_{k,t}$, and next increment α and β while holding the invariant that $\mathsf{pcf}^{(0)}_{k,t} = \mathsf{pcf}^{(1)}_{k,t}$. More specifically, we prove the following:

(1) *In* $\exp_{1,1}$ *we have* $\mathsf{pcf}^{(0)}_{1,0} = \mathsf{pcf}^{(1)}_{1,0}$ *(see. Claim A.2).* We rely on the SEH binding on the path to $\mathsf{epcf}_{1,0}$, PKE correctness, and the verifier's equality check with $\mathsf{pcf}_{1,0} \in \mathsf{crs}$.

(2) *In* $\exp_{\alpha,\alpha}$ *if* $\mathsf{pcf}^{(0)}_{k,t-1} = \mathsf{pcf}^{(1)}_{k,t-1}$ *then* $\mathsf{pcf}^{(0)}_{k,t} = \mathsf{pcf}^{(1)}_{k,t}$ *(see Claim A.3).* To prove this we use an induction on ℓ, the level of the proof $\pi^{(b)}_\ell = (\mathsf{v}^{(b)}, \rho^{(b)}_1, \rho^{(b)}_2, \mathsf{h}^{(b)}, \hat{\pi}^{(b)})$ that includes a proof for $\mathsf{pcf}_{k,t-1} \to \mathsf{pcf}_{k,t}$. We first extract a witness from $\hat{\pi}^{(b)}$, then using SEH binding property argue the extracted witness is consistent with the extracted hash value from $\mathsf{v}^{(b)}$. Then starting with hashed inputs $\mathsf{h}^{(b)}$ that are consistent with hz, by relying on collision resistance of CRHF we get that the hashed input values in the witness are also consistent with hz. Then we rely on the induction step. For the induction base, we additionally rely on the knowledge soundness of NIZK, together with reading and writing soundness of HT (for $\mathsf{is\text{-}nxt\text{-}cnfg}_{\mathcal{M}}$), to conclude the claim.

(3) *If* $\mathsf{pcf}^{(0)}_{k,t} = \mathsf{pcf}^{(1)}_{k,t}$ *in* $\exp_{\alpha,\alpha}$ *then* $\mathsf{pcf}^{(0)}_{k,t} = \mathsf{pcf}^{(1)}_{k,t}$ *in* $\exp_{\alpha,\alpha+1}$ *(see Claim A.4).* Note that the statement depends only on SEH trapdoors and PKE secret-key, thus we can rely on the index hiding property of seBARG to change the binding path from α to $\alpha+1$.

(4) *If* $\mathsf{pcf}^{(0)}_{k,t} = \mathsf{pcf}^{(1)}_{k,t}$ *in* $\exp_{\alpha,\alpha+1}$ *then* $\mathsf{pcf}^{(0)}_{k,t} = \mathsf{pcf}^{(1)}_{k,t}$ *in* $\exp_{\alpha+1,\alpha+1}$ *(see Claim A.5).* To prove this we take the following steps – (a) in $\exp_{\alpha,\alpha+1}$ if $\mathsf{pcf}^{(0)}_{k,t} = \mathsf{pcf}^{(1)}_{k,t}$ then $\mathsf{pcf}^{(0)}_l = \mathsf{pcf}^{(1)}_l$; this holds as the path on which $\mathsf{pcf}^{(b)}_l$ is computed is the same as the path on which $\mathsf{pcf}^{(b)}_{k,t}$ is extracted, (b) in $\exp_{\alpha,\alpha+1}$ if $\mathsf{pcf}^{(0)}_l = \mathsf{pcf}^{(1)}_l$ then $\mathsf{pcf}^{(b)}_r = \mathsf{pcf}^{(b)}_r$; this holds either by the verifier's check or the seBARG proof check at some level j, (c) if $\mathsf{pcf}^{(b)}_r = \mathsf{pcf}^{(b)}_r$ in $\exp_{\alpha,\alpha+1}$ then $\mathsf{pcf}^{(b)}_r = \mathsf{pcf}^{(b)}_r$ in $\exp_{\alpha+1,\alpha+1}$; this holds by the SEH index hiding and the fact that statement depends only on seBARG trapdoors and PKE secret key, (d) in $\exp_{\alpha+1,\alpha+1}$ if $\mathsf{pcf}^{(b)}_r = \mathsf{pcf}^{(b)}_r$ then $\mathsf{pcf}^{(0)}_{k,t} = \mathsf{pcf}^{(1)}_{k,t}$; this holds as the path on which $\mathsf{pcf}^{(b)}_r$ is computed is the same as the path on which $\mathsf{pcf}^{(b)}_{k,t}$ is extracted.

(5) *In* $\exp_{KT,KT}$ *if* $\mathsf{pcf}^{(0)}_{N,T} = \mathsf{pcf}^{(1)}_{N,T}$ *then the adversary cannot generate valid proofs* $\Pi^{(0)}$ *and* $\Pi^{(1)}$ *for two different final outputs* $\mathsf{out}^{(0)}$ *and* $\mathsf{out}^{(1)}$ *(see Claim A.6).* To prove this claim we rely on the SEH binding and PKE correctness to show $\mathsf{pcf}^{(b)}_{K,T} = \mathsf{pcf}^{(b)}$ where $\mathsf{pcf}^{(b)} \in \pi^{(b)}_{\mathsf{out}}$, which implies that $\mathsf{pcf}^{(0)} = \mathsf{pcf}^{(1)}$, and finally rely on reading soundness of HT to prove $\mathsf{out}^{(0)} = \mathsf{out}^{(1)}$.

3.4 Further Optimizing zk-IVsC

In this section we explain how to further optimize our zk-IVsC.

What if output is large? Consider a RAM machine $\mathcal{M}$ for which the output size n_{out} is large, e.g., $n_{\mathsf{out}} \simeq S$. In this scenario, as our proof size and verification time grows with n_{out}, our construction becomes inefficient. In this case we alter the proof to not compute π_{out}, and let the verification to only verify a digest of the final configuration (which is already included in our construction) and conclude the proof by relying on collision resistance of HT. Then the efficiency analysis of our construction becomes independent of n_{out}, making the construction efficient even if the output size is large.

More efficient prover. Suppose we don't want the running time of the prover (and Update) to grow with the size of cf. Note that the only part of the computation that grows with $|\mathsf{cf}| = S$ is the computation of the hash tree given the configuration $\mathsf{cf}_{k,0}$. To avoid this step, we can consider the computation of the RAM machine to keep a tree of the cf in its memory. Thus we update the state st such that instead of $\mathsf{st}_i = (\mathsf{cf}_i, \mathsf{pcf}_i, r_i)$, we have $\mathsf{st}_i = (\mathsf{cf}_i, \mathsf{pcf}_i, \mathsf{tree}_i, r_i)$. Note that the size of st only suffers from a (multiplicative) logarithmic overhead on the size of cf. However, Update algorithm will be relieved from the computation of $\mathsf{tree}_{k,0}$ each time a new user starts to run the machine. Therefore, the new running time of Update is $\mathsf{poly}(\lambda, n) + T \cdot \mathsf{poly}(\lambda, \log S, \log n, \log T, \log k, n_{\mathsf{out}})$ and the running time of the new prover is $K\left(\mathsf{poly}(\lambda, n) + T \cdot \mathsf{poly}(\lambda, \log S, \log n, \log T, \log K, n_{\mathsf{out}})\right)$.

Succinct communication to the next user. Note that each time a user finishes running the machine, it has to pass the final state/configuration to the next user (so that they can start running the machine). We highlight that if the amount of information needed to start running the machine by the next user is succinct, then there is a succinct way of sending the last configuration to the next user. Namely, user k will only write the information that needs to be passed on at the beginning of the work tape and let the rest of the work tape be all zeros (this will only have an additive overhead of size at most S on the running time of the machine, and w.l.o.g. we can assume $S \leq T$). Then the user will only send the non-zero part of the memory to the next user and the next user will be able to reconstruct the entire state/configuration. Therefore, we can always assume that for all $k \in [N]$, $\mathsf{st}_{k,T}$ has a short description (e.g., of size $\mathsf{poly}(\lambda, \log S)$).

3.5 Predicated zk-IVsC

3.5.1 Definition

In this section, we extend zk-IVsC definition to further impose a well-formedness property on the input to the machine, and then propose a construction that achieves such property.

Syntax. A predicated zero-knowledge incrementally verifiable streaming computation (predicated $\mathsf{zk\text{-}IVsC}$) scheme for a RAM machine $\mathcal{M}$ is defined *similarly* to $\mathsf{zk\text{-}IVsC}$, except that, first the Gen algorithm additionally takes a predicate size n_f as input, and second, the primitive also has the following PPT algorithms for proving and verifying the well-formedness of the inputs:

$\mathsf{PredicateUpdate}(\mathsf{crs}, f_k, z_k, \mathsf{ps}_k, \overline{f}_{k-1}, \mathsf{hz}_{k-1}, \Sigma_{k-1}) \to (\mathsf{hz}_k, \Sigma_k)$ The predicate update algorithm takes as input a CRS crs, a predicate f_k, an input z_k, set of predicates $\overline{f}_{k-1} = (f_j)_{j\in[k-1]}$, a private state ps_k, a digest hz_{k-1}, and a proof Σ_{k-1} and outputs an updated digest hz_k and an updated proof Σ_k.

$\mathsf{PredicateVerify}(\mathsf{crs}, \overline{f}, \mathsf{hz}, \Sigma) \to 0/1$ The predicate verifier algorithm takes as input a CRS crs, a set of predicates $\overline{f} = (f_k)_{k\in[K]}$, a digest hz, and a proof Σ, and outputs a bit to signal whether the proof is valid or not.

Remark 3.8 (Predicate Prover)**.** We will define the predicate prover algorithm $\mathsf{PredicateProve}(\mathsf{crs}, \overline{f}, \overline{z}, \overline{\mathsf{ps}})$ for a predicate $\mathsf{zk\text{-}IVsC}$ inductively as follows (where $\mathsf{hz}_0, \Sigma_0 = \epsilon$):

If $K = 1$ then output $(\mathsf{hz}_1, \Sigma_1) \leftarrow \mathsf{PredicateUpdate}(\mathsf{crs}, f_1, z_1, \mathsf{ps}_1, \overline{f}_0, \mathsf{hz}_0, \Sigma_0)$, otherwise inductively compute $(\mathsf{hz}_{K-1}, \Sigma_{K-1}) = \mathsf{PredicateProve}(\mathsf{crs}, \overline{f}_{K-1}, \overline{z}_{K-1}, \overline{\mathsf{ps}}_{K-1})$ and output

$$(\mathsf{hz}_K, \Sigma_K) \leftarrow \mathsf{PredicateUpdate}(\mathsf{crs}, f_K, z_K, \mathsf{ps}_K, \overline{f}_{K-1}, \mathsf{hz}_{K-1}, \Sigma_{K-1}).$$

Definition 3.9 (Predicated $\mathsf{zk\text{-}IVsC}$)**.** A predicated zero-knowledge incrementally verifiable streaming computation scheme $(\mathsf{Gen}, \mathsf{Update}, \mathsf{Digest}, \mathsf{Verify}, \mathsf{PredicateProve}, \mathsf{PredicateVerify})$ for $\mathcal{M}$ has to satisfy the following properties:

Completeness. For every $\lambda, n, S, T \in \mathbb{N}$, any $\overline{z} = (z_k)_{k\in[K]}$ s.t. $z_k \in \{0,1\}^n$,

$$\Pr\left[\begin{array}{l|l} (\overline{z}, \mathsf{out}) \in \mathcal{L}_{\mathcal{M},T} \ \wedge \ f(z_k) = 1 \ \wedge & \mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda, n, S, T, n_f) \\ \mathsf{hz} = \mathsf{hz}' = \mathsf{Digest}(\mathsf{crs}, \overline{z}, \overline{\mathsf{ps}}) \ \wedge & (\mathsf{st}, \mathsf{hz}, \Pi) \leftarrow \mathsf{Prove}(\mathsf{crs}, \overline{z}, \overline{\mathsf{ps}}) \\ \mathsf{Verify}(\mathsf{crs}, (\mathsf{hz}, \mathsf{out}), \Pi) = 1 & \text{find } \mathsf{cf} \in \mathsf{st}, \mathsf{out} = \mathsf{cf}[1, \ldots, n_{\mathsf{out}}] \\ \mathsf{PredicateVerify}(\mathsf{crs}, \overline{f}, \mathsf{hz}, \Sigma) = 1 & (\mathsf{hz}', \Sigma) \leftarrow \mathsf{PredicateProve}(\mathsf{crs}, \overline{f}, \overline{z}, \overline{\mathsf{ps}}) \end{array}\right] = 1.$$

Efficiency. For the completeness experiment above,

- The size of the crs and setup running time are $\mathsf{poly}(\lambda, \log S, n, n_f)$.
- The size of Π, and update/verifier's running time are the same as in IVsC
- The size of Σ is $\mathsf{poly}(\lambda, n, n_f, \log K)$, and predicate update/verifier's running time are $\mathsf{poly}(K) + \mathsf{poly}(\lambda, n, n_f, \log K)$. (We also consider - and later achieve - a more efficient predicate update/verifier that run in $\mathsf{poly}(\lambda, n, n_f, \log K)$, when they are allowed to preprocess the set $\overline{f}$..

Collision Resistance, Soundness, and Strong Soundness. Similar to $\mathsf{zk\text{-}IVsC}$.

Predicate Soundness. There exists PPT extractor $\mathcal{E}$ and algorithms TGen and $\mathsf{HashCheck}$ such that for any stateful PPT adversary $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that for every $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l|c|l} \mathsf{PredicateVerify}(\mathsf{crs},\overline{f},\mathsf{hz},\Sigma)=1 \wedge & & (n,S,T,n_f,k^*)\leftarrow\mathcal{A}(1^\lambda),\\ (f_{k^*}(z^*)=0 \vee & : & (\mathsf{crs},\mathsf{td})\leftarrow\mathsf{TGen}(1^\lambda,n,S,T,k^*),\\ \mathsf{HashCheck}(\mathsf{crs},\mathsf{hz},z^*,k^*,\mathsf{op})=0) & & (\overline{f},\mathsf{hz},\Sigma)\leftarrow\mathcal{A}(\mathsf{crs})\\ & & (z^*,\mathsf{op})\leftarrow\mathcal{E}(\mathsf{td},\overline{f},\mathsf{hz},\Sigma) \end{array}\right]\leq \mathsf{negl}(\lambda),$$

and crs and $\mathsf{crs}' \leftarrow \mathsf{Gen}(1^\lambda, n, S, T)$ are computationally indistinguishable even given k^*.

Zero-Knowledge. There exists a stateful PPT simulator $\mathcal{S}$ such that for any PPT adversary $\mathcal{A}$, there is a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l|c|l} & & (n,S,T,n_f)\leftarrow\mathcal{A}(1^\lambda), b\leftarrow\{0,1\}\\ & & \mathsf{crs}_0\leftarrow\mathsf{Gen}(1^\lambda,n,S,T,n_f), \mathsf{crs}_1\leftarrow\mathcal{S}(1^\lambda,n,S,T,n_f),\\ & & (\overline{z},\overline{f})\leftarrow\mathcal{A}(\mathsf{crs}_b), (f_k(z_k)=1)_{k\in[K]}, \mathsf{cf}=\mathcal{M}(\overline{z};1^T)\\ b=b' & : & (\mathsf{st}_0,\overline{\mathsf{hz}}_0,\Pi_0)\leftarrow\mathsf{Prove}(\mathsf{crs}_0,\overline{z},\overline{\mathsf{ps}})\\ & & (\mathsf{hz}_{0,K},\Sigma_0)\leftarrow\mathsf{PredicateProve}(\mathsf{crs},\overline{f},\overline{z},\overline{\mathsf{ps}})\\ & & (\mathsf{st}_1,\overline{\mathsf{hz}}_1,\Pi_1,\Sigma_1)\leftarrow\mathcal{S}(\mathsf{crs}_1,\mathsf{cf})\\ & & b'\leftarrow\mathcal{A}(\mathsf{st}_b,\overline{\mathsf{hz}}_b,\Pi_b,\Sigma_b) \end{array}\right]\leq \mathsf{negl}(\lambda),$$

where we consider a variant of Prover that additionally outputs all the intermediate hz values, i.e., $\overline{\mathsf{hz}}_b = (\mathsf{hz}_{b,k})_{k\in[K]}$.

3.5.2 Construction

Notations. We implement our algorithms in the RAM model and to indicate a RAM access, we use *underline* in the construction. For ease of exposition (and w.l.o.g.) we assume T is a power of 2. We let $r_{1,0}$ to be all zeros.

By $\overline{f}^* = (\overline{f}_1^*, \overline{f}_2^*)$ we denote $\overline{f}^* = (f_k)_{k\in[1,2t]}$, $\overline{f}_1^* = (f_k)_{k\in[1,t]}$, and $\overline{f}_2^* = (f_k)_{k\in[t+1,2t]}$. We say γ corresponds to z_k (or just k) if $(\gamma_{\log T},\ldots,\gamma_\lambda)$ is the unique path from root to $k^{*\text{th}}$ leaf in a full binary tree of depth $\lambda - \log T + 1$.

Construction overview. The $\mathsf{PredicateUpdate}$ works similarly to Update, that is, it builds a powers-of-2 structure of proofs, generates proofs at the base level using a NIZK, and recursively merges the proofs using $\mathsf{seBARGs}$. Moreover, the construction relies on a decomposition algorithm that decomposes the set of all predicates to their corresponding powers-of-2 structure. More specifically we define $\mathsf{TreeDecompose}(\overline{f}) = (\overline{f}_{\mathsf{dc},0},\ldots,\overline{f}_{\mathsf{dc},\lambda})$ to be an algorithm that receives as input a set $\overline{f}$ of size k and decomposes it to disjoint sets $\overline{f}_{\mathsf{dc},i}$ as follows: first write k as a summation of powers of 2, i.e. $k = \sum_{i=0}^{\lambda} b_i \cdot 2^i$ where $b_i \in \{0,1\}$ and let $s_i = \sum_{j=0}^{i} b_j \cdot 2^j$ for $i \in [-1,\lambda]$. Finally it outputs $(\overline{f}_{\mathsf{dc},0},\ldots,\overline{f}_{\mathsf{dc},\lambda})$ where $\overline{f}_{\mathsf{dc},i} = (f_k)_{k\in[K-s_i+1,K-s_{i-1}]}$ for $i \in [0,\lambda]$.

Language $\mathcal{L}_{\mathsf{zk},f}$

Hardwired: $\mathsf{pk}, \mathsf{hk}_{\mathsf{ht}}$
Instance: $x = (f, \mathsf{ct}_{\mathsf{in}})$
Witness: $w = (z, \mathsf{rt}_{\mathsf{in}}, r)$
Membership: w is a valid witness for $x \in \mathcal{L}_{\mathsf{zk},f}$ if the following hold:

$\mathsf{ct}_{\mathsf{in}}$ **is the encryption of** $\mathsf{rt}_{\mathsf{in}}$**.** Namely, $\mathsf{PKE.Enc}(\mathsf{pk}, \mathsf{rt}_{\mathsf{in}}; r) = \mathsf{ct}_{\mathsf{in}}$.

$\mathsf{rt}_{\mathsf{in}}$ **is the input root.** Namely, $(\mathsf{rt}_{\mathsf{in}}, \mathsf{tree}_{\mathsf{in}}) = \mathsf{HT.Hash}(\mathsf{hk}_{\mathsf{ht}}, z)$.

Input satisfies the predicate. Namely, Check $f(z) = 1$.

Figure 6: Description of language $\mathcal{L}_{\mathsf{zk},f}$.

Language $\mathcal{L}_{\mathsf{bp},f,\ell}$

Hardwired: $\mathsf{hk}_{\mathsf{ch}}, \mathsf{crs}_{\mathsf{zk},f}, (\mathsf{crs}_{\mathsf{bp},f,j})_{j\in[\ell-1]}$
Instance: $x = (\overline{f}^* = (\overline{f}_1^*, \overline{f}_2^*), \mathsf{h}, i)$
Witness: $w = (\mathsf{h}_1, \mathsf{h}_2, \pi)$
Membership: w is a valid witness for $x \in \mathcal{L}_{\mathsf{bp},f,\ell}$ if the following hold:

h **is the hash of** $(\mathsf{h}_1, \mathsf{h}_2)$**.** Namely, $\mathsf{CRHF.Hash}(\mathsf{hk}_{\mathsf{ch}}, (\mathsf{h}_1, \mathsf{h}_2)) = \mathsf{h}$.

π **certifies well-formedness of** h_i **w.r.t.** $\overline{f}_i^*$**.** Namely, if $\ell = \log T$ then $\mathsf{NIZK.Verify}(\mathsf{crs}_{\mathsf{zk},f}, (\overline{f}_i^*, \mathsf{h}_i), \pi) = 1$, otherwise, $\mathsf{seBARG.Verify}(\mathsf{crs}_{\mathsf{bp},f,\ell-1}, (\overline{f}_i^*, \mathsf{h}_i, j)_{j\in[2]}, \pi) = 1$.

Figure 7: Description of language $\mathcal{L}_{\mathsf{bp},f,\ell}$.

Building blocks and parameters. We use the same primitives as in zk-IVsC. In addition to the sizes in zk-IVsC, let $n_f = |f_k|$ for $k \in [K]$, and $n_{\mathsf{bp},f,\ell} = 2^{\ell-\log T+1} \cdot n_f + \lambda + 1$. Let $|r|$ and $|\mathsf{ct}|$ be respectively the size of randomness and ciphertext of PKE, and let $n_{\mathsf{zk},f,x} = n_f + |\mathsf{ct}|$ and $n_{\mathsf{zk},f,w} = n + \lambda + |r|$.

Construction 3.10. We use the same parsing notation as in zk-IVsC, except that we modify crs parsing as $\mathsf{crs} = (\mathsf{pk}, \mathsf{hk}_{\mathsf{ht}}, \mathsf{hk}_{\mathsf{ch}}, \mathsf{crs}_{\mathsf{zk}}, \mathsf{crs}_{\mathsf{zk},f}, (\mathsf{crs}_{\mathsf{bp},\ell}, \mathsf{hk}_{\mathsf{cf},\ell})_{\ell\in[\lambda]}, (\mathsf{crs}_{\mathsf{bp},f,\ell})_{\ell\in[\log T,\lambda]}, \mathsf{pcf}_{1,0})$, and we additionally parse $\Sigma = (\pi_{f,\log T-1}, \cdots, \pi_{f,\lambda})$ wherever needed. Update , Digest, and Verify algorithms remain the same as in zk-IVsC.

$\mathsf{Gen}(1^\lambda, n, S, T, n_f) \to \mathsf{crs}$**.** This algorithm is the same as in zk-IVsC except that it additionally samples $\mathsf{crs}_{\mathsf{zk},f} = \mathsf{NIZK.Setup}(1^\lambda, 1^{n_{\mathsf{zk},f,x}}, 1^{n_{\mathsf{zk},f,w}})$, lets $\gamma = 1^\lambda$ and for all $\ell \in [\log T, \lambda]$, it samples $(\mathsf{crs}_{\mathsf{bp},f,\ell}, \mathsf{td}_{\mathsf{bp},f,\ell}) \leftarrow \mathsf{seBARG.Gen}(1^\lambda, 2, n_{\mathsf{bp},f,\ell}, \gamma[\ell - \log T + 1])$, and outputs $\mathsf{crs} = (\mathsf{pk}, \mathsf{hk}_{\mathsf{ht}}, \mathsf{hk}_{\mathsf{ch}}, \mathsf{crs}_{\mathsf{zk}}, \mathsf{crs}_{\mathsf{zk},f}, (\mathsf{crs}_{\mathsf{bp},\ell}, \mathsf{hk}_{\mathsf{cf},\ell})_{\ell\in[\lambda]}, (\mathsf{crs}_{\mathsf{bp},f,\ell})_{\ell\in[\log T,\lambda]}, \mathsf{pcf}_{1,0})$.

$\mathsf{PredicateUpdate}(\mathsf{crs}, f_k, z_k, \mathsf{ps}_k, \overline{f}_{k-1}, \mathsf{hz}_{k-1}, \Sigma_{k-1}) \to (\mathsf{hz}_k, \Sigma_k)$**.** This algorithm does the following:

1. Let $\mathsf{hz}_{k-1} = (\mathsf{h}_{\log T-1}, \cdots, \mathsf{h}_\lambda)$, and $\Sigma_{k-1} = (\pi_{f,\log T-1}, \cdots, \pi_{f,\lambda})$.
2. *Decompose the predicates* $\overline{f}_{k-1}$ *to powers-of-2:* $(\overline{f}_{\mathsf{dc},0}, \ldots, \overline{f}_{\mathsf{dc},\lambda}) = \mathsf{TreeDecompose}(\overline{f}_{k-1})$.
3. *Encrypt the root of input:* $(\mathsf{rt}_{\mathsf{in}}, \mathsf{tree}_{\mathsf{in}}) = \mathsf{HT.Hash}(\mathsf{hk}_{\mathsf{ht}}, z_k)$ and $\mathsf{ct}_{\mathsf{in}} = \mathsf{PKE.Enc}(\mathsf{pk}, \mathsf{rt}_{\mathsf{in}}; \mathsf{ps}_k)$.
4. *Compute a NIZK proof for* $f_k(z_k) = 1$*:* $\pi_{\mathsf{zk},f} = \mathsf{NIZK.Prove}(\mathsf{crs}_{\mathsf{zk},f}, x = (f_k, \mathsf{ct}_{\mathsf{in}}), w)$ for the language $\mathcal{L}_{\mathsf{zk},f}$ where $w = (z_k, \mathsf{rt}_{\mathsf{in}}, \mathsf{ps}_k)$.

5. *Define the temporary proof, input hash, and predicates:* Let $\mathsf{t\text{-}h} = \mathsf{ct}_{\mathsf{in}}$, $\mathsf{t\text{-}}f = (f_k)$, $\mathsf{t\text{-}}\pi = \pi_{\mathsf{zk},f}$. Note that at the moment these values correspond to a level 0 proof in the power-of-2-structure, but they will be treated as proofs for level ℓ, in the next iterative process.
6. *Iteratively merge the proofs:* let $\ell = \log T - 1$ and while $\pi_{f,\ell}$ is not empty run the following iterative process and increment ℓ at the end:
   (a) Let $\mathsf{h} = \mathsf{h}_\ell$, $\mathsf{h}' = \mathsf{t\text{-}h}$, $\pi_1 = \pi_{f,\ell}$, $\pi_2 = \mathsf{t\text{-}}\pi$, $\overline{f}_1^* = \mathsf{t\text{-}}f$, $\overline{f}_2^* = \overline{f}_{\mathsf{dc},\ell-\log T+1}$.
   (b) *Merge level ℓ input hashes to get the next temp* h*:* $\mathsf{t\text{-}h} = \mathsf{CRHF.Hash}(\mathsf{hk}_{\mathsf{ch}}, (\mathsf{h}, \mathsf{h}'))$.
   (c) *Merge level ℓ predicates to get the next temp predicates:* $\mathsf{t\text{-}}f = (\overline{f}_1^* \cup \overline{f}_2^*)$.
   (d) *Merge NIZK/BARG proofs:* $\mathsf{t\text{-}}\pi = \mathsf{seBARGProve}\left(\mathsf{crs}_{\mathsf{bp},f,\ell+1}, \begin{matrix}(\mathsf{t\text{-}}f, \mathsf{t\text{-}h}, i)_{i\in[2]},\\ (\mathsf{h}, \mathsf{h}', \pi_i)_{i\in[2]}\end{matrix}\right)$ for the language $\mathcal{L}_{\mathsf{bp},f,\ell+1}$.
7. Output $(\mathsf{hz}_k = (\epsilon, \ldots, \epsilon, \mathsf{t\text{-}h}, \mathsf{h}_{\ell+1}, \ldots, \mathsf{h}_\lambda), \Sigma_k = (\epsilon, \ldots, \epsilon, \mathsf{t\text{-}}\pi, \pi_{f,\ell+1}, \ldots, \pi_{f,\lambda}))$.

$\mathsf{PredicateVerify}(\mathsf{crs}, \overline{f}, \mathsf{hz}, \Sigma) \to 0/1$**.** The verifier first decomposes the predicates $\overline{f}$ and then verifies NIZK/BARG proofs. Namely, let $(\overline{f}_{\mathsf{dc},0}, \ldots, \overline{f}_{\mathsf{dc},\lambda}) = \mathsf{TreeDecompose}(\overline{f})$, if $\pi_{f,\log T-1}$ is not empty then check $\mathsf{NIZK.Verify}(\mathsf{crs}_{\mathsf{zk},f}, (\overline{f}_{\mathsf{dc},0}, \mathsf{h}_{\log T-1}), \pi_{f,\log T-1}) = 1$, and for $\ell \in [\log T, \lambda]$ if $\pi_{f,\ell}$ is not empty then check $\mathsf{seBARG.Verify}(\mathsf{crs}_{\mathsf{bp},f,\ell}, (\overline{f}_{\mathsf{dc},\ell-\log T+1}, \mathsf{h}_\ell, i)_{i\in[2]}, \pi_{f,\ell}) = 1$. Output 1 if all the checks pass and 0 o.w.

3.5.3 Analysis

Theorem 3.11. Assuming that seBARG is an almost rate-1 seBARG (Definitions 2.6 and 2.8), NIZK is a secure NIZK protocol (Definition 2.15), PKE is a secure PKE system (Definition 2.17), SEH is an almost rate-1 SEH (Definitions 2.2 and 2.4), HT is a hash tree (Definition 2.11), and CRHF is a 2-to-1 collision resistant hash function (Definition 2.10), Construction 3.10 is a predicated zk-IVsC (Definition 3.9).

Corollary 3.12. Assuming almost rate-1 seBARGs and PKEs, Construction 3.10 is a predicated zk-IVsC.

Corollary 3.13. Assuming either LWE, or DLIN, or sub-exponential DDH, Construction 3.10 is a predicated zk-IVsC.

The analysis of the predicate proof is very similar to the analysis of zk-IVsC. We refer the reader to Appendix B for a complete analysis of our construction.

4 Incrementally Computable Zero-Knowledge Arguments (ICZK)

4.1 Definition

Consider an NP language $\mathcal{L} = \{\overline{x} = (x_1, \ldots, x_K) \mid \exists \overline{w} = (w_1, \ldots, w_K) : \mathcal{R}(\overline{x}, \overline{w}) = 1\}$ defined w.r.t. a relation $\mathcal{R}$.

Syntax. An incrementally computable zero-knowledge (ICZK) argument consists of the following polynomial time algorithms:

$\mathsf{Setup}(1^\lambda, n_x, n_w) \to \mathsf{crs}$**.** The probabilistic setup algorithm takes as input a security parameter λ, a single instance length n_x, a single witness length n_w, and outputs a common reference string crs.

$\mathsf{Update}(\mathsf{crs}, x_k, w_k, \mathsf{st}_{k-1}, \Pi_{k-1}) \to (\mathsf{st}_k, \Pi_k)$**.** The update algorithm takes as input a common reference string crs, an instance x_k, and a witness w_k, a state st_{k-1}, and a proof Π_{k-1}, and outputs a state st_k and a proof Π_k.

$\mathsf{Verify}(\mathsf{crs}, \overline{x}, \Pi) \to 0/1$**.** The verifier algorithm takes as input a common reference string crs, an instance $\overline{x}$, and a proof Π. It outputs 0 (reject) or 1 (accept).

Remark 4.1 (Prover.)**.** We will define the prover $\mathsf{Prove}(\mathsf{crs}, \overline{x}, \overline{w}) \to \pi$ algorithm for an ICZK inductively as follows (where st_0 is the initial state and $\Pi_0 = \epsilon$):

If $K = 1$ then output $(\mathsf{st}_1, \Pi_1) \leftarrow \mathsf{Update}(\mathsf{crs}, x_1, w_1, \mathsf{st}_0, \Pi_0)$, otherwise inductively compute $(\mathsf{st}_{K-1}, \Pi_{K-1}) = \mathsf{Prove}(\mathsf{crs}, (x_k, w_k)_{k\in[K-1]})$ and output $(\mathsf{st}_K, \Pi_K) \leftarrow \mathsf{Update}(\mathsf{crs}, x_K, w_K, \mathsf{st}_{K-1}, \Pi_{K-1})$.

Definition 4.2 (ICZK)**.** An incrementally computable zero-knowledge argument $(\mathsf{Setup}, \mathsf{Update}, \mathsf{Verify})$ for $\mathcal{L}$ is required to satisfy the following properties:

Completeness. For all $\lambda, n_x, n_w \in \mathbb{N}$ and $(\overline{x}, \overline{w}) \in \mathcal{R}$ where $|x_k| = n_x$ and $|w_k| = n_w$ we have:

$$\Pr[\mathsf{Verify}(\mathsf{crs}, \overline{x}, \Pi) = 1 : \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, n_x, n_w), \Pi \leftarrow \mathsf{Prove}(\mathsf{crs}, \overline{x}, \overline{w})] = 1.$$

Adaptive Soundness. For any PPT adversary $\mathcal{A}$, there is a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda, n_x \in \mathbb{N}$:

$$\Pr[\mathsf{Verify}(\mathsf{crs}, \overline{x}, \Pi) = 1 \wedge \overline{x} \notin \mathcal{L} : \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, n_x, n_w), (\overline{x}, \Pi) \leftarrow \mathcal{A}(\mathsf{crs}), |x_k| = n_x] \leq \mathsf{negl}(\lambda)$$

Incremental Zero-Knowledge. There exists a stateful PPT simulator $\mathcal{S}$ such that for any PPT adversary $\mathcal{A}$, there is a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda, n_x, n_w \in \mathbb{N}$:

$$\Pr\left[\begin{array}{lll} & & \mathsf{crs}_0 \leftarrow \mathsf{Setup}(1^\lambda, n_x, n_w), \mathsf{crs}_1 \leftarrow \mathcal{S}(1^\lambda, n_x, n_w), b \leftarrow \{0,1\} \\ \mathcal{A}(\mathsf{st}_b, \Pi_b) = b & : & (\overline{x}, \overline{w}) \leftarrow \mathcal{A}(\mathsf{crs}_b), \mathsf{cf} = \mathcal{M}((\overline{x}, \overline{w}); 1^T) \\ & & (\mathsf{st}_0, \Pi_0) \leftarrow \mathsf{Prove}(\mathsf{crs}_0, \overline{x}, \overline{w}), (\mathsf{st}_1, \Pi_1) \leftarrow \mathcal{S}(\mathsf{crs}_1, \mathsf{cf}) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

4.2 Construction

In this section we first define the notation, then present an overview of our construction, followed by the building blocks and the construction itself.

Notations. In this construction we will use the algorithms in predicated zk-IVsC (with RAM access) as subroutines, thus we let $\Psi.\mathsf{A}$ denote calling algorithm A from the predicated zk-IVsC construction as a subroutine in this construction. Let $\mathcal{M}$ be a RAM machine for relation $\mathcal{R}$ with space S, and running time T (on each input $z_i = (x_i, w_i)$). Define $\mathcal{M}'$ to be the same as $\mathcal{M}$ except that it takes

$z_i = (x_i, w_i, y_1, y_2)$ as input but does not use y_1 and y_2 in the computation. Later, we let $y_1 = \mathsf{ps}_i$ and $y_2 = r_i$ be randomnesses for different PKE schemes.

Construction overview. Let $\mathcal{M}$ be a RAM machine for relation $\mathcal{R}$ with space S, and running time T (on each input $z_k = (x_k, w_k)$). Our starting point is to consider the computation of $\mathcal{M}$ as a $\mathsf{zk\text{-}IVsC}$ where the streaming inputs are $z_k = (x_k, w_k)$ and the tuple (hz, Π) is considered as the proof of computation. While this proof is zero-knowledge, one cannot verify that hz is an honest digest of $\overline{z}$. Our idea is to send a hiding encoding of the witnesses and use the input well-formedness proof to verify the consistency of hz with such an encoding. There are two caveats in such an approach: (1) both the encoding of witnesses and encrypted digest of the RAM machine have to be randomized to satisfy any hiding properties, however, our predicated $\mathsf{zk\text{-}IVsC}$ only supports deterministic computation, and the predicates have to be publicly known, and (2) the predicate proof is sound only at a trapdoor location, thus we have to break the global well-formedness of hz to a local well-formedness checks. We get around both of these issues using following blueprint: (1) encrypt the witness and private state (w_k, ps_k) under some randomness r_k to get ct_k, (2) use $\mathsf{zk\text{-}IVsC}$ to prove the computation of a RAM machine $\mathcal{M}'$, and (3) use predicate f_k to prove x_k is the correct statement, $(w_k, \mathsf{ps}_k, r_k)$ is consistent with ct_k, and $\mathsf{hz}_{k-1} \xrightarrow{\mathsf{ct}_{\mathsf{in}}} \mathsf{hz}_k$ where $\mathsf{ct}_{\mathsf{in}}$ is the encryption of z_k's root under randomness ps_k. Then the final proof consists of $\overline{\mathsf{ct}}_k$, $\overline{\mathsf{hz}}_k$, and $\mathsf{zk\text{-}IVsC}$ computation and predicate proofs.

Building blocks and parameters. Let HT be a hash tree with output root of size λ on any input, CRHF be a collision resistant $\mathsf{Hash} : \{0,1\}^{2\lambda} \to \{0,1\}^{\lambda}$, seBARG be an almost rate-1 seBARG, SEH be an almost rate-1 SEH, NIZK be a NIZK, PKE be a *randomness-dependent message* public-key encryption (RDM-PKE), and rrPKE be a rate-1 leakage resilient public-key encryption with randomness recovery. In addition to predicated $\mathsf{zk\text{-}IVsC}$ sizes, we let n_{ps} be the size of the randomness ps for the PKE above, n_r be the size of the randomness for the rrPKE above, and $n = (n_x + n_w + n_{\mathsf{ps}} + n_r)$. Additionally, we let n_f be the max size of the function f_k from Fig. 8.

Predicate f_k

Hardwired: $\mathsf{hk}_{\mathsf{ht}}, \mathsf{pk}, \mathsf{pk}', (x_k, \mathsf{ct}_k, \mathsf{hz}_{k-1}, \mathsf{hz}_k)$
Input: $z = (x, w, \mathsf{ps}, r)$
Output: 1 if the following hold:

The RAM input is consistent with NIZK statement. Namely, $x = x_k$.

The RAM input is consistent with encrypted witness. $\mathsf{PKE.Enc}(\mathsf{pk}, (w, \mathsf{ps}); r) = \mathsf{ct}_k$.

The RAM input is consistent with $\mathsf{hz}_{k-1} \to \mathsf{hz}_k$ transition Namely, $(\mathsf{rt}, \mathsf{tree}) = \mathsf{HT.Hash}(\mathsf{hk}_{\mathsf{ht}}, x, w, \mathsf{ps}, r)$, $\mathsf{ct}_{\mathsf{in}} = \mathsf{PKE.Enc}(\mathsf{pk}', \mathsf{rt}; \mathsf{ps})$ and $\mathsf{nxt\text{-}hz}(\mathsf{hk}_{\mathsf{ch}}, \mathsf{hz}_{k-1}, \mathsf{ct}_{\mathsf{in}}) = \mathsf{hz}_k$ using Fig. 5.

Figure 8: Description of predicate f_k in the ICZK construction.

Construction 4.3. In this construction, we always parse $\Pi = (\overline{\mathsf{hz}}, \Pi_\psi, \Sigma_\psi, \overline{\mathsf{ct}})$ and $\mathsf{crs} = (\mathsf{crs}_\psi, \mathsf{pk})$, where we have $\mathsf{crs}_\psi = (\mathsf{pk}', \mathsf{hk}_{\mathsf{ht}}, \mathsf{hk}_{\mathsf{ch}}, \mathsf{crs}_{\mathsf{zk}}, \mathsf{crs}_{\mathsf{zk},f}, (\mathsf{crs}_{\mathsf{bp},\ell}, \mathsf{hk}_{\mathsf{cf},\ell})_{\ell\in[\lambda]}, (\mathsf{crs}_{\mathsf{bp},f,\ell})_{\ell\in[\log T,\lambda]}, \mathsf{pcf}_{1,0})$.

$\mathsf{Setup}(1^\lambda, n_x, n_w) \to \mathsf{crs}$**.** This algorithm samples crs_ψ similar to predicated $\mathsf{zk\text{-}IVsC}$ (except that now PKE is RDM-PKE), and $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{rrPKE.Gen}(1^\lambda)$, and outputs $\mathsf{crs} = (\mathsf{crs}_\psi, \mathsf{pk})$.

$\mathsf{Update}(\mathsf{crs}, x_k, w_k, \mathsf{st}_{k-1}, \Pi_{k-1}) \to (\mathsf{st}_k, \Pi_k)$**.** This algorithm does the following:

1. Parse $\mathsf{crs} = (\mathsf{crs}_\psi, \mathsf{pk})$ and $\Pi_{k-1} = (\overline{\mathsf{hz}}_{k-1}, \Pi_{\psi,k-1}, \Sigma_{\psi,k-1}, \overline{\mathsf{ct}}_{k-1})$.
2. *Encrypt the witness and private state:* sample randomness ps_k, r_k, and compute $\mathsf{ct}_k = \mathsf{rrPKE.Enc}(\mathsf{pk}, (w_k, \mathsf{ps}_k); r_k)$.
3. *Update the computation proof for running $\mathcal{M}'$ on $z_k = (x_k, w_k, \mathsf{ps}_k, r_k)$:* $(\mathsf{st}_k, \mathsf{hz}_k, \Pi_{\psi,k}) \leftarrow \Psi.\underline{\mathsf{Update}}(\mathsf{crs}_\psi, z_k = (x_k, w_k, \mathsf{ps}_k, r_k), \mathsf{st}_{k-1}, \mathsf{hz}_{k-1}, \Pi_{\psi,k-1}; \mathsf{ps}_k)$.
4. *Get the predicates:* find $\mathsf{hk}_{\mathsf{ht}}, \mathsf{pk}' \in \mathsf{crs}_\psi$; then construct $\overline{f}_k$ using Fig. 8.
5. *Update the predicate well-formedness proof w.r.t. $f_k(z_k) = 1$:* compute $(\mathsf{hz}_k, \Sigma_{\psi,k}) \leftarrow \Psi.\mathsf{PredicateUpdate}(\mathsf{crs}_\psi, f_k, z_k, \mathsf{ps}_k, \overline{f}_{k-1}, \mathsf{hz}_{k-1}, \Sigma_{\psi,k-1})$.
6. Output $(\mathsf{st}_k, \Pi_k = (\overline{\mathsf{hz}}_k, \Pi_{\psi,k}, \Sigma_{\psi,k}, \overline{\mathsf{ct}}_k))$.

$\mathsf{Verify}(\mathsf{crs}, \overline{x}, \Pi) \to 0/1$**.** This algorithm parses $\mathsf{crs} = (\mathsf{crs}_\psi, \mathsf{pk})$, and $\Pi = (\overline{\mathsf{hz}}, \Pi_\psi, \Sigma_\psi, \overline{\mathsf{ct}})$. It additionally parses crs_ψ to find $\mathsf{hk}_{\mathsf{ht}}$ and pk', and then construct $\overline{f}$ using Fig. 8. Finally it checks $\Psi.\mathsf{Verify}(\mathsf{crs}, (\mathsf{hz}_K, 1), \Pi_\psi) = 1$ and $\Psi.\mathsf{PredicateVerify}(\mathsf{crs}, \overline{f}, \mathsf{hz}_K, \Sigma_\psi) = 1$. It outputs 1 if both checks pass, and 0 otherwise.

4.3 Analysis

Theorem 4.4. Assuming that seBARG is an almost rate-1 seBARG (Definitions 2.6 and 2.8), NIZK is a secure NIZK protocol (Definition 2.15), PKE is a secure RDM-PKE system (Definitions 2.17 and 2.24), SEH is an almost rate-1 SEH (Definitions 2.2 and 2.4), HT is a hash tree (Definition 2.11), CRHF is a 2-to-1 collision resistant hash function (Definition 2.10), and rrPKE is a secure rate-1 leakage resilient PKE with randomness recovery (Definitions 2.17 to 2.19 and 2.22), Construction 4.3 is an ICZK (Definition 4.2) with proof size $K|w_i| + K \cdot \mathsf{poly}(\lambda, \log|x_1|, \log|w_1|, \log K) + \mathsf{poly}(\lambda, \log|x_1|, |w_1|, \log K)$.

Corollary 4.5. Assuming almost rate-1 seBARGs and lossy trapdoor functions, Construction 4.3 is an ICZK.

Corollary 4.6. Assuming either LWE, or DLIN, or sub-exponential DDH, Construction 4.3 is an ICZK.

Next, we provide an overview of soundness and zero-knowledge of our construction and refer the reader to Appendix C for a complete analysis of our construction.

Soundness overview. To rely on the soundness of zk-IVsC we need to find $(\overline{z}, \overline{\mathsf{ps}})$ s.t. hz is the honest digest of $(\overline{z}, \overline{\mathsf{ps}})$. First, note that $\overline{x}$ is publicly known. Then, we decrypt all the ciphertexts $\overline{\mathsf{ct}}$ to find $(\overline{w}, \overline{\mathsf{ps}})$; moreover, recall that $\overline{\mathsf{ct}}$ is generated using PKE with *randomness recovery*, thus, we can recover $\overline{r}$ as well. Hence, we can reconstruct $z_k = (x_k, w_k, \mathsf{ps}_k, r_k)$. Next, we need to prove that hz is consistent with $\overline{z}$ as reconstructed above.

Suppose this doesn't hold; then there is an index k such that $\mathsf{nxt\text{-}hz}(\mathsf{hz}_{k-1}, \mathsf{ct}_{\mathsf{in}}) \neq \mathsf{hz}_k$ while $\mathsf{hz}_{k-1} = \mathsf{Digest}(\overline{z}_{k-1}, \overline{\mathsf{ps}}_{k-1})$. Next we guess index k, and prove that the above event cannot happen if our guess is correct. Namely, if the predicate proof is extractable on k, then there is an input z^* that satisfies the predicate. By the predicate check, $\mathsf{nxt\text{-}hz}(\mathsf{hz}_{k-1}, \mathsf{ct}^*_{\mathsf{in}}) \neq \mathsf{hz}_k$, where $\mathsf{ct}^*_{\mathsf{in}}$ is computed from z^*. Now note that the computation of $\mathsf{ct}^*_{\mathsf{in}}$ (resp. $\mathsf{ct}_{\mathsf{in}_k}$) from z^* (resp. z_k) is deterministic, and its correctness is checked by the predicate f_k. Therefore, all that is left to prove is that $z_k = z^*$. For this, we crucially rely on the *perfect correctness* of the PKE that is used to compute $\overline{\mathsf{ct}}$. Namely,

since we check that each predicate input is consistent with ct_k, and the choice of such input is unique by the perfect correctness of PKE, it holds that z^* is the same as z_k.

Incremental zero-knowledge overview. We first recall the main circularity challenges in proving zero-knowledge of our construction. First, note that the ciphertext ct_k is the encryption of (w_k, ps_k) and its security relies on the randomness of r_k, however, r_k is used as part of the input z_k everywhere in the $\mathsf{zk\text{-}IVsC}$ proofs. Moreover, the zero-knowledge of $\mathsf{zk\text{-}IVsC}$ holds if ps_k is random and independent. However, not only there is leakage of ps_k in ct_k, but also the message encrypted under the randomness ps_k (which is the hash of z_k), depends on ps_k. Our main observation is that we can break down these circularities step by step and bound the leakage at every step. Next, we show how to carefully formulate the hybrids and use the right building blocks to get around such issues.

$\mathcal{H}_0$: This is the original experiment.

$\mathcal{H}_1$: *Use* NIZK *simulator to compute* $\mathsf{crs}_{\mathsf{zk}}$*, and* π_{zk} *in* Π_ψ*.* $\mathcal{H}_0 \to \mathcal{H}_1$ relies on zero-knowledge property of the underlying NIZK. Note that after this step the only information about an intermediate configurations $\mathsf{cf}_{k,t}$ is in $\mathsf{epcf}_{k,t}$, and the only information about the inputs of $\mathcal{M}$ is in $(\overline{\mathsf{hz}}, \overline{\mathsf{ct}}, \Sigma_\psi)$ and $(\mathsf{ct}_{\mathsf{in},k})_{k\in[K]}$ (that is being used in the statements of Π_ψ and Σ_ψ).

$\mathcal{H}_2$: *Use* NIZK *simulator to compute* $\mathsf{crs}_{\mathsf{zk},f}$*, and* $\pi_{\mathsf{zk},f}$ *in* Σ_ψ*.* $\mathcal{H}_1 \to \mathcal{H}_2$ also relies on the zero-knowledge property of the underlying NIZK. Note that after this step there is no information about inputs $(\overline{w}, \overline{\mathsf{ps}}, \overline{r})$ in the proof except the ones in $(\mathsf{hz}_k, \mathsf{ct}_k, \mathsf{ct}_{\mathsf{in},k})_{k\in[K]}$ where $\overline{\mathsf{hz}}$ is constructed only using $(\mathsf{ct}_{\mathsf{in},k})_{k\in[K]}$. Thus, the only information about z_k is in ct_k and $\mathsf{ct}_{\mathsf{in},k}$.

$\mathcal{H}_3$: *For every* $\mathsf{cf}_{k,t}$ *(except* $\mathsf{cf}_{1,0}$ *and* $\mathsf{cf}_{K,T}$*), compute* $\mathsf{epcf}_{k,t} \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}, 0^{|\mathsf{pcf}|})$*.* $\mathcal{H}_2 \to \mathcal{H}_3$ relies on the PKE security. Note that after this step there is no information about the intermediate configurations in the proof.

$\mathcal{H}_4$: *For every* ct_k*, independently sample* r_k *and let* $\mathsf{ct}_k = \mathsf{rrPKE.Enc}(\mathsf{pk}, 0^{n_w+|\mathsf{ps}|}; r_k)$*.* $\mathcal{H}_3 \to \mathcal{H}_4$ relies on the security with leakage of the underlying rrPKE as the only information about randomness $\overline{r}$ remained in the proof is in $(\mathsf{ct}_{\mathsf{in},k})_{k\in[K]}$ (that is an encryption of a short digest of the input z_k.). Note that after this step there's no information about $\overline{w}$ and $\overline{\mathsf{ps}}$ in the $\overline{\mathsf{ct}}$.

$\mathcal{H}_5$: *For every* z_k*, let* $\mathsf{ct}_{\mathsf{in},k} \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}, 0^\lambda)$*.* $\mathcal{H}_4 \to \mathcal{H}_5$ relies on the RDM security of the underlying PKE as the only information about randomness $\overline{\mathsf{ps}}$ remained in the proof is in $(\mathsf{ct}_{\mathsf{in},k})_{k\in[K]}$ (that is the encryption of a short digest of the inputs). Note that after this step there's no information about $\overline{w}$ in the proof and the proof can be simulated using only $(\overline{x}, \mathsf{cf})$.

5 From IVsC to Deterministic Distributed Computation

We present this section as PCD for deterministic computation, however, we note that our constructions are not true PCD and we modify the network by adding new communication edges.

Distributed Computation. A distributed computation is a tuple $\mathsf{S} = (G, \mathsf{data}, \mathsf{linp}, \mathbb{C})$ where G denotes the graph of computation, data denotes the data that is transferred over the edges, linp denotes the local input of any user/vertex, and $\mathbb{C}$ denotes a compliance function for the inputs and output of any node. More formally we define a distributed computation as follows:

- $G = (V, E)$.

- $\mathsf{D} = \text{Diameter}(G)$.
- $V = \{v_i^d\}$ where $d \in \mathsf{D}$ and $i \in [2^{d-1}]$.
- $E = \{(v_j^{d+1}, v_i^d)\}_{j \in [2(i-1)+1, 2\cdot i], d \in [\mathsf{D}-1]}$.
- $\mathsf{data}_{\mathsf{inp}}(v_i^d) = (\mathsf{data}(v, v_i^d))_v$ s.t. $(v, v_i^d) \in E$.
- $\mathsf{data}_{\mathsf{out}}(v_i^d) = \mathsf{data}(v_i^d, v)$ where $(v_i^d, v) \in E$.
- $\mathbb{C}(\mathsf{data}_{\mathsf{inp}}(v), \mathsf{linp}(v), \mathsf{data}_{\mathsf{out}}(v)) = 1$ for all $v \in V$.

Proof Carrying Data (PCD) Systems. PCD systems are usually defined for non-deterministic computations where the security is w.r.t. a knowledge extractor. However, IVsC is defined for deterministic computation. Therefore, for applications of IVsC to PCD we define PCD for deterministic computations where we assume that the compliance functions $\mathbb{C}$ for each set of node inputs $(\mathsf{data}_{\mathsf{inp}}(v), \mathsf{linp}(v))$ outputs 1 only for a unique node output $\mathsf{data}_{\mathsf{out}}(v)$. Moreover, this output can be computed using a RAM machine, i.e., there exists a RAM machine $\mathcal{R}$ s.t. $\mathcal{R}(\mathsf{data}_{\mathsf{inp}}(v), \mathsf{linp}(v)) = \mathsf{data}_{\mathsf{out}}(v)$. For the rest of this section by PCD we refer to PCD for deterministic computation.

Definition 5.1 (PCD for Deterministic Computation)**.** A PCD system $\Pi = (\mathsf{Gen}, \mathsf{Prove}, \mathsf{Verify})$ for a deterministic compliance function $\mathbb{C}$ consists of the following algorithms:

- $\mathsf{Gen}(1^\lambda) \to \mathsf{crs}$
- $\mathsf{Prove}(\mathsf{crs}, \mathsf{linp}(v), \mathsf{data}_{\mathsf{inp}}(v), \pi_{\mathsf{inp}}(v)) \to (\mathsf{data}_{\mathsf{out}}(v), \pi_{\mathsf{out}}(v))$.
- $\mathsf{Verify}(\mathsf{crs}, G, (\mathsf{linp}(v))_{v \in V}, \mathsf{data}_{\mathsf{out}}(v^*), \pi_{\mathsf{out}}(v^*)) \to \{0, 1\}$.

Where v^* is the sink of the graph. Note that the Prove algorithm is defined for a single vertex, and we define the $\mathsf{GlobalProve}(\mathsf{crs}, G, (\mathsf{linp}(v))_{v \in V}) \to (\mathsf{data}_{\mathsf{out}}(v^*), \pi_{\mathsf{out}}(v^*))$ inductively by writing down a topological order of the graph and running the Prove algorithm for each of the vertices in order.

Completeness. For any $\lambda \in \mathbb{N}$ it holds that:

$$\Pr\left[\mathsf{Verify}(\mathsf{crs}, X) = 1 \quad : \quad \begin{array}{c} (\mathsf{data}_{\mathsf{out}}(v^*), \pi_{\mathsf{out}}(v^*)) \leftarrow \\ \mathsf{GlobalProve}(\mathsf{crs}, G, (\mathsf{linp}(v))_{v \in V}) \end{array} \right] = 1$$

where $\mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda)$, and $X = (G, (\mathsf{linp}(v))_{v \in V}, \mathsf{data}_{\mathsf{out}}(v^*), \pi_{\mathsf{out}}(v^*))$.

Soundness. For any PPT stateful admissible adversary $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that for every $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l} \mathsf{Verify}(\mathsf{crs}, G, (\mathsf{linp}(v))_{v \in V}, \mathsf{out}, \pi) = 1, \\ \wedge\ \mathsf{out} \neq \mathsf{data}_{\mathsf{out}}(v^*) \end{array} \quad : \quad \begin{array}{l} (G) \leftarrow \mathcal{A}(1^\lambda), \mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda), \\ ((\mathsf{linp}(v))_{v \in V}, \mathsf{out}, \pi) \leftarrow \mathcal{A}(\mathsf{crs}) \end{array} \right] \leq \mathsf{negl}(\lambda).$$

Remark 5.2 (From IVsC to Linear PCD)**.** Note that our construction of IVsC (i.e., consider Construction 3.4 without zero-knowledge proofs and properties) implies a PCD for a deterministic computation on a path graph. More specifically, we can use $(\mathsf{Gen}, \mathsf{Update}, \mathsf{Verify})$ of an IVsC as

$(\mathsf{Gen}, \mathsf{Prove}, \mathsf{Verify})$ in a PCD for path graphs. Note that we can consider a RAM machine $\mathcal{R}$ with fixed input size n, max configuration size S, and running time T that computes the compliant function $\mathbb{C}$. Then we can instantiate an IVsC using these parameters and let the PCD algorithms be the same as the IVsC algorithms. Note that the way we analyze our construction it is only secure against adversaries that output $1^{N \cdot T}$, therefore our PCD constructions from IVsC are also secure only if the adversary outputs the running time of the computation in unary. Additionally, note that our construction achieves a stronger security notion w.r.t. a verifier that only takes a digest of the local inputs. This implies the notion of PCD for path graphs and also enables us to strengthen this notion correspondingly.

In the following, we discuss the set of distributed computations for which we can generate a PCD using our results on IVsC.

5.1 Binary Tree to Path Distributed Computation

Here we show how to transform a depth-D binary tree distributed computation into a Path computation with a succinct amount of data (a D overhead on the original data size) carried over the edges. Hence, given a PCD scheme for a path distributed computation, we can generate a PCD scheme for a depth-D (with an acceptable D, say $\mathsf{poly}(\lambda)$ at most) binary tree distributed computation.

Construction 5.3. Given $\mathsf{S} = (G, \mathsf{data}, \mathsf{linp}, \mathbb{C})$ construct $\mathsf{S}' = (G', \mathsf{data}', \mathsf{linp}, \mathbb{C}')$ as follows:

1. $G' = (V, E')$.
2. $E' = E_1 \cup E_2 \cup E_3$ where
   - $E_1 = \{(v^{\mathsf{D}}_{2i-1}, v^{\mathsf{D}}_{2i})\}_{i \in [2^{d-2}]}$.
   - $E_2 = \{(v^{d+1}_{2i}, v^{d}_{i})\}_{d \in [\mathsf{D}-1], i \in [2^{d-1}]}$.
   - $E_3 = \{(v^{d}_{2i-1}, v^{\mathsf{D}}_{(2i-1)(2^{\mathsf{D}-d})+1})\}_{d \in [\mathsf{D}-1], i \in [2^{d-2}]}$.
3. $\mathsf{linp}'(v^d_i) = \mathsf{linp}(v^d_i)$.
4. Let $\mathsf{data}'_{\mathsf{out}}(v) = (\mathsf{data}^1_{\mathsf{out}}(v), \ldots, \mathsf{data}^{\mathsf{D}}_{\mathsf{out}}(v))$ and $\mathsf{data}'_{\mathsf{inp}}(v) = (\mathsf{data}^1_{\mathsf{inp}}(v), \ldots, \mathsf{data}^{\mathsf{D}}_{\mathsf{inp}}(v))$ be as follows:
   - Let $\mathsf{data}'_{\mathsf{inp}}(v^{\mathsf{D}}_1) = \epsilon$, and for $(d, i) \neq (\mathsf{D}, 1)$, $(v^{d'}_{i'}, v^d_i) \in E'$, let $\mathsf{data}'_{\mathsf{inp}}(v^d_i) = \mathsf{data}'_{\mathsf{out}}(v^{d'}_{i'})$.
   - For any edge $(v^{\mathsf{D}}_{2i-1}, v^{\mathsf{D}}_{2i}) \in E_1$, let $\mathsf{data}'_{\mathsf{out}}(v^{\mathsf{D}}_{2i-1}) = \mathsf{data}'_{\mathsf{inp}}(v^{\mathsf{D}}_{2i-1})$ and append $\mathsf{data}_{\mathsf{out}}(v^{\mathsf{D}}_{2i-1})$ to $\mathsf{data}^{\mathsf{D}}_{\mathsf{out}}(v^{\mathsf{D}}_{2i-1})$ where $\mathbb{C}(\epsilon, \mathsf{linp}(v^{\mathsf{D}}_{2i-1}), \mathsf{data}_{\mathsf{out}}(v^{\mathsf{D}}_{2i-1})) = 1$.
   - For any edge $(v^{d+1}_{2i}, v^d_i) \in E_2$, let $\mathsf{data}'_{\mathsf{out}}(v^{d+1}_{2i}) = \mathsf{data}'_{\mathsf{inp}}(v^{d+1}_{2i})$, $\mathsf{data}^{d+2}_{\mathsf{out}}(v^{d+1}_{2i}) = \epsilon$, and append $\mathsf{data}_{\mathsf{out}}(v^{d+1}_{2i})$ to $\mathsf{data}^{d+1}_{\mathsf{out}}(v^{d+1}_{2i})$ where $\mathbb{C}(\mathsf{data}^{d+2}_{\mathsf{inp}}(v^{d+1}_{2i}), \mathsf{linp}(v^{d+1}_{2i}), \mathsf{data}_{\mathsf{out}}(v^{d+1}_{2i})) = 1$.
   - For any edge $(v^d_{2i-1}, v^{\mathsf{D}}_{(2i-1)(2^{\mathsf{D}-d})+1}) \in E_3$, let $\mathsf{data}'_{\mathsf{out}}(v^d_{2i-1}) = \epsilon$ and append $\mathsf{data}_{\mathsf{out}}(v^d_{2i-1})$ to $\mathsf{data}^d_{\mathsf{out}}(v^d_{2i-1})$ where $\mathbb{C}(\mathsf{data}^{d+1}_{\mathsf{inp}}(v^d_{2i-1}), \mathsf{linp}(v^d_{2i-1}), \mathsf{data}_{\mathsf{out}}(v^d_{2i-1})) = 1$.
5. $\mathbb{C}'(\mathsf{data}'_{\mathsf{inp}}(v), \mathsf{linp}(v), \mathsf{data}'_{\mathsf{out}}(v)) = 1$ for all $v \in V$.

Correctness. From the construction it is clear that for any $v \in V$, $\mathsf{data}_{\mathsf{inp}}(v) \subseteq \mathsf{data}'_{\mathsf{inp}}(v)$, therefore, user v has all the data it needs to perform its computation. Additionally, it is clear from the construction of S' that every v has exactly one input and output data, so it is a path computation.

Succinctness. Let $\max_{(v,v')\in E} |\mathsf{data}(v,v')| = S_{\mathsf{PCD}}$, then for any user/vertex $v \in V$, we have: $|\mathsf{data}_{\mathsf{inp}}| = \mathsf{D} \cdot S_{\mathsf{PCD}}$ since for any $d \in \mathsf{D}$ (except one at most), $\mathsf{data}^d_{\mathsf{inp}}(v)$ includes one data at most.

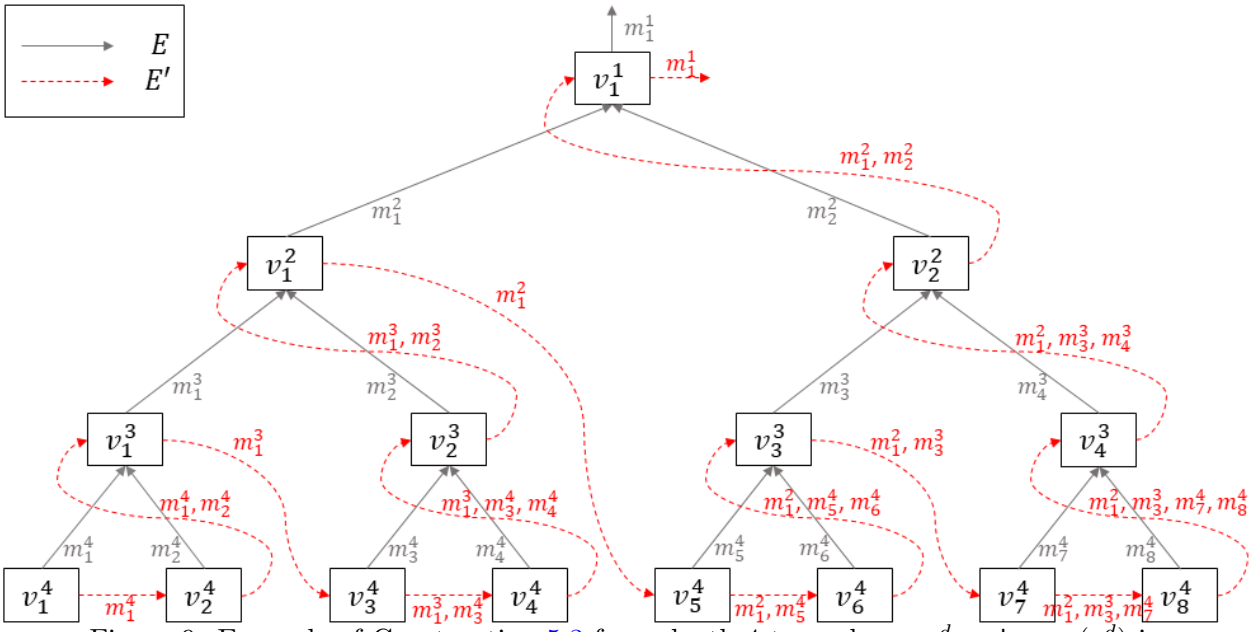

Figure 9: Example of Construction 5.3 for a depth 4 tree where $m^d_i = \mathsf{data}_{\mathsf{out}}(v^d_i)$ in the original distributed computation. Here E' and $\mathsf{data}_{\mathsf{out}}(v^d_i)$ are determined by red.

5.2 Class of Succinctly Transformable Distributed Computations to a Path

Here we generalize the approach in the previous subsection and discuss a bigger class of distributed computations that can be transformed into a path distributed computation. First note that the main challenge in such a transformation is to keep a succinct set of all the data that is needed to do the rest of the computation. More specifically, in this approach, we have to find a topological order of a directed graph such that at any point in the order, the amount of data that needs to be kept to do the rest of the computation is succinct. We can transform any graph G for which such a topological order exists, to a path of computation with a succinct amount of data carried over the edges.

More formally, let $(v_1, \ldots, v_N)$ be a topological order of a graph G. Denote $\mathbb{L}$ to be a labeling function such that if the computation for v_i is done then $\mathbb{L}(v_i) = 1$, and otherwise $\mathbb{L}(v_i) = 0$. Define a node v_i to be open, $\mathbb{O}(v_i) = 1$, if $\mathbb{L}(v_i) = 1$ and $\mathbb{L}(v_j) = 0$ where $(v_i, v_j) \in E$. Suppose we start the computation from v_1 and stop right after some arbitrary point v_i. Define the set of open nodes as follows $\mathcal{O}_i = \{v_j \mid \mathbb{O}(v_j) = 1\}$. Now note that at any point v_i, we only need to keep $\mathsf{data}_{\mathsf{out}}$ for the open nodes. This is because if a node v_k is not open either (1) $\mathbb{L}(v_k) = 0$ hence $\mathsf{data}_{\mathsf{out}}(v_k)$ hasn't

been computed yet, or (2) $\mathbb{L}(v_k) = 1$ and $\mathbb{L}(v_{k'}) = 1$ where $(v_k, v_{k'}) \in E$, therefore, $\mathsf{data}_{\mathsf{out}}(v_k)$ is already used as part of $\mathsf{data}_{\mathsf{inp}}(v_{k'})$ and it will not be used elsewhere, so we don't need to keep it. Putting this all together means that if for a graph G, there exists a topological order $(v_1, \ldots, v_N)$ such that $|\mathcal{O}_i| = \mathsf{poly}(\lambda)$ for any $i \in [N]$, then there exists a transformation from $\mathsf{S} = (G, \ldots)$ to a linear distributed computation with a succinct amount of data carried over the edges.

Zero-Knowledge for Distributed Computation. We call a graph *good* if it is succinctly transformable to a path. Defining zero-knowledge for these graphs is a challenging task let alone constructing a zero-knowledge PCD for these graphs. Here we informally discuss a high-level idea to achieve zero-knowledge and then analyze the kind of zero-knowledge that it potentially achieves. After transforming a graph into a path, we generate a pair of public and secret keys of an FHE encryption system for the sink node and publish the public key. Then every user encrypts its local input under that public key and the computation is defined as a homomorphic computation over the inputs. This approach achieves zero-knowledge for the inputs for every user against all the other users except the sink user. Note that the same approach can be taken to achieve privacy in a path graph as well but our construction has two advantages over this. First, it is based on simpler assumptions than FHE. Second, the sink user learns nothing about the other users' inputs other than the last configuration of the machine.

Now if the sink user of a good graph is not predetermined, e.g., the graph is still evolving and more users are joining, then we can use the same idea with a leveled set of public and secret keys, where every user has a pair of secret and public keys, and the public keys are published. The first user starts by encrypting its message under its parent's public key and performs the homomorphic computation and sends the result to the next user. Every user upon receiving an input has two choices, either (1) the computation is done under some other user's public key, in which case the user will continue the computation under the same public key, or (2) the computation is done under that user's public key, in which case it uses its secret key to decrypt the computation, then encrypts the computation using its parent's public key and performs the rest of the computation homomorphically under that public key. This approach achieves zero-knowledge for inputs for user v against any user v' if there is no path from v to v' in the graph.

We leave the formalization and constructions of zero-knowledge PCD for general good graphs (and without FHE) as an interesting open problem.

6 Benefits and Applications

The streaming scenario. In many modern applications, the data for computation might not be available *all at once*, or it could be in an ongoing process of generation. Additionally, datasets are often too large and it is difficult to store the entire data simultaneously, or with a single data processor. E.g., consider a cloud service provider running an ML process on a massive dataset collected by a trusted agency such as the National Institute of Health (NIH). The data could be a large corpus of *sensitive* patient data from many different sources (e.g., healthcare providers, research facilities, etc), where the data is signed by each contributing source. *The question is how can we verify that the clour provider trained the model correctly?*

NIH might ask the cloud provider to use IVC, but this would require the training data to be *fixed at the beginning*. Therefore, if new data arrives (e.g., new routine blood sample results), then the cloud provider has to restart the IVC computation. Also, if the ML algorithm is itself a

streaming algorithm (e.g., reinforcement learning), then the cloud cannot take advantage of this. Furthermore, the proof might leak information about the sensitive data, thus cannot be shared with third-party clients (e.g., an insurance provider, or research lab) who are not authorized to look at the entire data but only the parameters of the trained ML model.

One could consider many more such applications (e.g., online video-processing of traffic-videos/user-videos by National Highway Institute/Youtube, etc). We believe that IVsC protocols will be a very valuable tool for all such applications. A cloud provider can create incremental proofs over streaming (sensitive) data. Moreover, as long as there is a publicly checkable predicate associated with the data (e.g., every medical record is signed by a trusted entity), then the proof can be verified without the entire dataset, but just the additional predicated proof. Thus, it will solve: (a) the streaming problem, and (b) data privacy problem.

Step-by-step zero-knowledge. Similarly, the notion of step-by-step zero-knowledge could be a great addition to practical deployments of proof systems. For instance, consider any sensitive computation performed by one user/device, or even multiple. A popular threat in many deployed cryptographic systems is of total system corruption. Unfortunately, standard zero-knowledge completely fails in such settings. However, step-by-step zero-knowledge offers a new approach for capturing such practical threats by defining a *best-possible* zero-knowledge property. E.g., consider any application where a user wants to prove that it correctly computed a PRF on a public input x. That is, it wants to prove $y = F_K(x)$, where K is the user's secret. Such proofs of PRF evaluations routinely occur in many blockchain applications (e.g., for leader election in proof-of-stake) or anonymous credentials/ blind signatures (for generating blinded credentials/signatures). Using a step-by-step ZK protocol, we can prove that the user's PRF key K will never be fully compromised even an attacker gets a hold of their system during the proof generation process. Basically, if one use GGM-style PRFs [GGM86], then after each computation step, the user can delete the PRG seed after it correctly expands it (depending upon the next input bit). This merely sketches how step-by-step ZK could go beyond conventional zero-knowledge formulations to provide better protections, and we believe further exploration would be quite meaningful for applications.

Acknowledgments. We would like to thank Elahe Sadeghi and Saikumar Yadugiri for their valuable suggestions and assistance in refining the content of this paper. We also thank anonymous reviewers for valuable feedback. Their inputs greatly improved the clarity and presentation of the ideas discussed.

References

[ADKL19] Prabhanjan Ananth, Apoorvaa Deshpande, Yael Tauman Kalai, and Anna Lysyanskaya. Fully homomorphic nizk and niwi proofs. In *Theory of Cryptography Conference*, pages 356–385. Springer, 2019.

[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 (SP)*, pages 315–334. IEEE, 2018.

[BC23] Benedikt Bünz and Binyi Chen. Protostar: Generic efficient accumulation/folding for special sound protocols. *Cryptology ePrint Archive*, 2023.

[BC24] Dan Boneh and Binyi Chen. Latticefold: A lattice-based folding scheme and its applications to succinct proof systems. *Cryptology ePrint Archive*, 2024.

[BCCT13] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for snarks and proof-carrying data. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 111–120, 2013.

[BCL+21] Benedikt Bünz, Alessandro Chiesa, William Lin, Pratyush Mishra, and Nicholas Spooner. Proof-carrying data without succinct arguments. In *Advances in Cryptology–CRYPTO 2021: 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16–20, 2021, Proceedings, Part I 41*, pages 681–710. Springer, 2021.

[BCMS20] Benedikt Bünz, Alessandro Chiesa, Pratyush Mishra, and Nicholas Spooner. Proof-carrying data from accumulation schemes. *Cryptology ePrint Archive*, 2020.

[BCPT13] Eleanor Birrell, Kai-Min Chung, Rafael Pass, and Sidharth Telang. Randomness-dependent message security. In *Theory of Cryptography: 10th Theory of Cryptography Conference, TCC 2013, Tokyo, Japan, March 3-6, 2013. Proceedings*, pages 700–720. Springer, 2013.

[BCTV14] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Scalable zero knowledge via cycles of elliptic curves. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 276–294. Springer, Heidelberg, August 2014.

[BDFG21] Dan Boneh, Justin Drake, Ben Fisch, and Ariel Gabizon. Halo infinite: Proof-carrying data from additive polynomial commitments. In *Advances in Cryptology–CRYPTO 2021: 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16–20, 2021, Proceedings, Part I 41*, pages 649–680. Springer, 2021.

[BGH19] Sean Bowe, Jack Grigg, and Daira Hopwood. Recursive proof composition without a trusted setup. *Cryptology ePrint Archive*, 2019.

[BKP+23] Nir Bitansky, Chethan Kamath, Omer Paneth, Ron Rothblum, and Prashant Nalini Vasudevan. Batch proofs are statistically hiding. *Cryptology ePrint Archive*, 2023.

[BMNW24] Benedikt Bünz, Pratyush Mishra, Wilson Nguyen, and William Wang. Accumulation without homomorphism. *Cryptology ePrint Archive*, 2024.

[BMRS20] Joseph Bonneau, Izaak Meckler, Vanishree Rao, and Evan Shapiro. Coda: Decentralized cryptocurrency at scale. *Cryptology ePrint Archive*, 2020.

[BR94] Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In Douglas R. Stinson, editor, *CRYPTO'93*, volume 773 of *LNCS*, pages 232–249. Springer, Heidelberg, August 1994.

[BSCTV17] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Scalable zero knowledge via cycles of elliptic curves. *Algorithmica*, 79:1102–1160, 2017.

[But22] Vitalik Buterin. The different types of zk-evms. https://vitalik.eth.limo/general/2022/08/04/zkevm.html, 2022.

[BWW23] Eli Bradley, Brent Waters, and David J Wu. Batch arguments to nizks from one-way functions. *Cryptology ePrint Archive*, 2023.

[CCDW20] Weikeng Chen, Alessandro Chiesa, Emma Dauterman, and Nicholas P Ward. Reducing participation costs via incremental verification for ledger systems. *Cryptology ePrint Archive*, 2020.

[CCG+23] Megan Chen, Alessandro Chiesa, Tom Gur, Jack O'Connor, and Nicholas Spooner. Proof-carrying data from arithmetized random oracles. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 379–404. Springer, 2023.

[CCS22] Megan Chen, Alessandro Chiesa, and Nicholas Spooner. On succinct non-interactive arguments in relativized worlds. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 336–366. Springer, 2022.

[CG] Jiaqi Cheng and Rishab Goyal. Personal communication.

[CGSY24] Alessandro Chiesa, Ziyi Guan, Shahar Samocha, and Eylon Yogev. Security bounds for proof-carrying data from straightline extractors. In *Theory of Cryptography Conference*, pages 464–496. Springer, 2024.

[CJJ21] Arka Rai Choudhuri, Abhihsek Jain, and Zhengzhong Jin. Snargs for $\mathcal{P}$ from LWE. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 68–79. IEEE, 2021.

[COS20] Alessandro Chiesa, Dev Ojha, and Nicholas Spooner. Fractal: Post-quantum and transparent recursive proofs from holography. In *Advances in Cryptology–EUROCRYPT 2020: 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10–14, 2020, Proceedings, Part I 39*, pages 769–793. Springer, 2020.

[CT10] Alessandro Chiesa and Eran Tromer. Proof-carrying data and hearsay arguments from signature cards. In *ICS*, volume 10, pages 310–331, 2010.

[CTV13] Stephen Chong, Eran Tromer, and Jeffrey A Vaughan. Enforcing language semantics using proof-carrying data. *Cryptology ePrint Archive*, 2013.

[CTV15] Alessandro Chiesa, Eran Tromer, and Madars Virza. Cluster computing in zero knowledge. In *Advances in Cryptology-EUROCRYPT 2015: 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II 34*, pages 371–403. Springer, 2015.

[CW23] Jeffrey Champion and David J Wu. Non-interactive zero-knowledge from non-interactive batch arguments. *Cryptology ePrint Archive*, 2023.

[CY21] Alessandro Chiesa and Eylon Yogev. Subquadratic snargs in the random oracle model. In *Advances in Cryptology–CRYPTO 2021: 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16–20, 2021, Proceedings, Part I 41*, pages 711–741. Springer, 2021.

[DGKV22] Lalita Devadas, Rishab Goyal, Yael Kalai, and Vinod Vaikuntanathan. Rate-1 non-interactive arguments for batch-np and applications. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1057–1068. IEEE, 2022.

[FGK+10] David Mandell Freeman, Oded Goldreich, Eike Kiltz, Alon Rosen, and Gil Segev. More constructions of lossy and correlation-secure trapdoor functions. In *Public Key Cryptography–PKC 2010: 13th International Conference on Practice and Theory in Public Key Cryptography, Paris, France, May 26-28, 2010. Proceedings 13*, pages 279–295. Springer, 2010.

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *Journal of the ACM (JACM)*, 33(4):792–807, 1986.

[GHMR18] Sanjam Garg, Mohammad Hajiabadi, Mohammad Mahmoody, and Ahmadreza Rahimi. Registration-based encryption: removing private-key generator from ibe. In *Theory of Cryptography: 16th International Conference, TCC 2018, Panaji, India, November 11–14, 2018, Proceedings, Part I 16*, pages 689–718. Springer, 2018.

[GKP+13] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. How to run turing machines on encrypted data. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 536–553. Springer, Heidelberg, August 2013.

[GV20] Rishab Goyal and Satyanarayana Vusirikala. Verifiable registration-based encryption. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part I*, volume 12170 of *LNCS*, pages 621–651. Springer, Heidelberg, August 2020.

[GW11] Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 99–108, 2011.

[HN23] Mathias Hall-Andersen and Jesper Buus Nielsen. On valiant's conjecture: impossibility of incrementally verifiable computation from random oracles. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 438–469. Springer, 2023.

[HW15] Pavel Hubacek and Daniel Wichs. On the communication complexity of secure function evaluation with long output. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science*, pages 163–172, 2015.

[KB23] Assimakis Kattis and Joseph Bonneau. Proof of necessary work: succinct state verification with fairness guarantees. In *International Conference on Financial Cryptography and Data Security*, pages 18–35. Springer, 2023.

[KLVW23] Yael Kalai, Alex Lombardi, Vinod Vaikuntanathan, and Daniel Wichs. Boosting batch arguments and ram delegation. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, pages 1545–1552, 2023.

[KRR14] Yael Tauman Kalai, Ran Raz, and Ron D Rothblum. How to delegate computations: the power of no-signaling proofs. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pages 485–494, 2014.

[KS22] Abhiram Kothapalli and Srinath Setty. Supernova: Proving universal machine executions without universal circuits. *Cryptology ePrint Archive*, 2022.

[KST22] Abhiram Kothapalli, Srinath Setty, and Ioanna Tzialla. Nova: Recursive zero-knowledge arguments from folding schemes. In *Annual International Cryptology Conference*, pages 359–388. Springer, 2022.

[Mer87] Ralph C Merkle. A digital signature based on a conventional encryption function. In *Conference on the theory and application of cryptographic techniques*, pages 369–378. Springer, 1987.

[Min17] O(1) labs. mina cryptocurrency. https://minaprotocol.com/, 2017.

[Nao03] Moni Naor. On cryptographic assumptions and challenges. In *Annual International Cryptology Conference*, pages 96–109. Springer, 2003.

[NT16] Assa Naveh and Eran Tromer. Photoproof: Cryptographic image authentication for any set of permissible transformations. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 255–271. IEEE, 2016.

[OPWW15] Tatsuaki Okamoto, Krzysztof Pietrzak, Brent Waters, and Daniel Wichs. New realizations of somewhere statistically binding hashing and positional accumulators. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015, Part I*, volume 9452 of *LNCS*, pages 121–145. Springer, Heidelberg, November / December 2015.

[PP22] Omer Paneth and Rafael Pass. Incrementally verifiable computation via rate-1 batch arguments. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1045–1056. IEEE, 2022.

[PW08] Chris Peikert and Brent Waters. Lossy trapdoor functions and their applications. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 187–196, 2008.

[Val08] Paul Valiant. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In *Theory of Cryptography: Fifth Theory of Cryptography Conference, TCC 2008, New York, USA, March 19-21, 2008. Proceedings 5*, pages 1–18. Springer, 2008.

[XZC+22] Tiancheng Xie, Jiaheng Zhang, Zerui Cheng, Fan Zhang, Yupeng Zhang, Yongzheng Jia, Dan Boneh, and Dawn Song. zkbridge: Trustless cross-chain bridges made practical. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 3003–3017, 2022.

[ZGSX23] Tianyu Zheng, Shang Gao, Yubo Song, and Bin Xiao. Leaking arbitrarily many secrets: Any-out-of-many proofs and applications to ringct protocols. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 2533–2550. IEEE, 2023.

A zk-IVsC Complete Analysis.

Proof of Theorem 3.5. We prove completeness, efficiency, strong soundness, and zero-knowledge properties of our construction.

Completeness. The completeness follows by the construction and the completeness of the underlying seBARG, NIZK, PKE, H, HT and SEH.

Efficiency. Recall that we are applying almost rate-1 SEH, and almost rate-1 seBARG. Moreover, note that the size of a ciphertext only grows polynomially with the message size and the size of a NIZK proof only grows polynomially with the size witness size and the statement size. Hence the size of each proof π_i is computed as follows (where c and c' are constants):

- $|\pi_{\mathsf{out}}| = \mathsf{poly}(\lambda, \log S, n_{\mathsf{out}})$, and $|\mathsf{epcf}|, |\mathsf{pcf}|, |\pi_0| = \mathsf{poly}(\lambda, \log S, \log n)$.
- $|\pi_1| = 2n_{\mathsf{cf},0} + (1 + c/\lambda)|\pi_0| + \mathsf{poly}(\lambda) \leq (1 + c/\lambda) \cdot \mathsf{poly}(\lambda, \log S, \log n)$
- For $\ell \leq \log T - 1$, $|\pi_\ell| = (1 + c'/\lambda)n_{\mathsf{cf},\ell-1} + 2|\rho| + (1 + c/\lambda)|\pi_{\ell-1}| + \mathsf{poly}(\lambda) \leq (1 + (c + c')/\lambda)^\ell \cdot \mathsf{poly}(\lambda, \log S, \log n, \ell)$.
- For $\log(T \cdot K) \geq \ell \geq \log T$, $|\pi_\ell| = (1 + c'/\lambda)n_{\mathsf{cf},\ell-1} + 2|\rho| + 2\lambda + (1 + c/\lambda)|\pi_{\ell-1}| + \mathsf{poly}(\lambda) \leq (1 + (c + c')/\lambda)^\ell \cdot \mathsf{poly}(\lambda, \log S, \log n, \ell) \leq e^{c+c'} \cdot \mathsf{poly}(\lambda, \log S, \log n, \ell)$.

Therefore it holds that $|\Pi| = \mathsf{poly}(\lambda, \log S, \log n, \log T, \log K)$. Note that here we consider RAM machines with polynomially bounded outputs. If the output size n_{out} grows larger (say, grows with S), we slightly modify the construction so that the proof size does not grow with n_{out} (see Section 3.4).

By the same argument since verifier only checks seBARG proofs and SEH path openings validity, and since the verification circuit size for BARGs at the base level (which grows with the NIZK verification size which in turn grows with the size of $\mathsf{is\text{-}nxt\text{-}cnfg}_{\mathcal{M}}$ and with encryption) is $\mathsf{poly}(\lambda, \log S, \log n, n_{\mathsf{out}})$, the verifier's running time grows with $\mathsf{poly}(\lambda, \log S, \log n, \log T, \log K)$.

The setup running time grows with the batch argument, the NIZK and SEH setup running time, and the pseudo-configuration computation time. The first three grow at most with $\mathsf{poly}(\lambda, \log S, \log n, \ell)$ for $\ell \leq \lambda$, thus the total running time of the crs and hk generations is $\mathsf{poly}(\lambda, \log S, \log n)$. For pseudo-configuration computation, the size of the work tape can be large, but since it is initialized to all zeros, the pseudo-configuration computation can be done efficiently by taking advantage of the symmetric structure of the computation when run on all zeros input. Thus, we only need an order of tree-depth steps to compute the root, and hence the pseudo-configuration computation time is $\mathsf{poly}(\lambda, \log S)$ making the total running time of the setup $\mathsf{poly}(\lambda, \log S, \log n)$. With the same argument, the crs size is $\mathsf{poly}(\lambda, \log S, \log n)$.

Finally, assuming RAM access to the memory (in the underlined parts in the construction) that includes the entire tree of the input and configurations, one can generate reading and writing openings by only reading the path from root to leaf of the tree. Now since each element on the path is

$\mathsf{poly}(\lambda)$-size and the path length is logarithmic in n and S, the total running time of a single reading or writing generation is $\mathsf{poly}(\lambda, \log S, \log n)$. Thus, the running time of a single step of update only grows with $\mathsf{poly}(\lambda, \log S, \log n, \log T, \log k, n_{\mathsf{out}})$. Hence, the running time of the update grows with T single steps plus the hash tree computation of the beginning and input digests. Therefore, the total running time of Update grows with $\mathsf{poly}(\lambda, n)+\mathsf{poly}(\lambda, S)+T\cdot\mathsf{poly}(\lambda, \log S, \log n, \log T, \log k, n_{\mathsf{out}})$.

Strong Soundness. Define the original experiment $\exp_{\mathsf{orig}}$ and the event Bad as follows:

$$\exp_{\mathsf{orig}} \coloneqq \left[\begin{array}{l} (n, S, T) \leftarrow \mathcal{A}(1^\lambda), \\ \mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda, n, S, T), \\ (\mathsf{hz}, (\mathsf{out}^{(b)}, \Pi^{(b)})_{b\in\{0,1\}}, 1^{K\cdot T}) \leftarrow \mathcal{A}(\mathsf{crs}) \end{array}\right],$$

$$\mathsf{Bad} \coloneqq \left[\begin{array}{l} \mathsf{Verify}(\mathsf{crs}, (\mathsf{hz}, \mathsf{out}^{(0)}), \Pi^{(0)}) = 1 \wedge \\ \mathsf{Verify}(\mathsf{crs}, (\mathsf{hz}, \mathsf{out}^{(1)}), \Pi^{(1)}) = 1 \wedge \\ \mathsf{out}^{(0)} \neq \mathsf{out}^{(1)} \end{array}\right].$$

Our goal is to prove that for any stateful PPT adversary $\mathcal{A}$ and any $\lambda \in \mathbb{N}$,

$$\Pr\left[\ \mathsf{Bad} \ : \ \exp_{\mathsf{orig}}\ \right] \leq \mathsf{negl}(\lambda).$$

Notation. For $t = T$, we define $(k, t, t+1) \coloneqq (k+1, 0, 1)$ (e.g., $(\mathsf{cf}_{k,t}, \mathsf{cf}_{k,t+1}) \coloneqq (\mathsf{cf}_{k+1,0}, \mathsf{cf}_{k+1,1})$). Let $\mathsf{pcf}^{(b)}_{k,t}$ (resp. $\mathsf{epcf}^{(b)}_{k,t}$) be the pseudo-configuration (resp. encrypted pseudo-configuration) of t^{th} configuration on the k^{th} input in the proof $\Pi^{(b)}$ for $b \in \{0, 1\}$. Furthermore, let $\mathsf{Gen}_{\alpha,\beta}$ be a generation algorithm with respect to α, β, where the NIZK extractor (instead of NIZK setup) is used to generate $\mathsf{crs}_{\mathsf{zk}}$, i.e. $(\mathsf{crs}_{\mathsf{zk}}, \mathsf{td}_{\mathsf{zk}}) \leftarrow \mathsf{NIZK}.\mathcal{E}(1^\lambda, 1^{n_{\mathsf{zk},x}}, 1^{n_{\mathsf{zk},w}})$, and let $\mathsf{Gen}_{\alpha,\beta}$ in addition to the crs output $\mathsf{td} = (\mathsf{td}_{\mathsf{bp}}, \mathsf{td}_{\mathsf{cf}}, \mathsf{td}_{\mathsf{zk}}, \mathsf{sk})$ (where $\mathsf{td}_{\mathsf{bp}} = (\mathsf{td}_{\mathsf{bp},\ell})_{\ell\in[\lambda]}$ and $\mathsf{td}_{\mathsf{cf}} = (\mathsf{td}_{\mathsf{cf},\ell})_{\ell\in[\lambda]}$). Define $\mathsf{depth}(\cdot)$ for a hash value v (resp. a proof π^*) be ℓ if $\mathsf{hk}_{\mathsf{cf},\ell}$ (resp. $\mathsf{crs}_{\mathsf{bp},\ell}$) is used to generate that value. For any hash value v of depth 0 let $\mathsf{v} = (\mathsf{epcf}_1, \mathsf{epcf}_2)$, then we denote by $\mathsf{v}[b] = \mathsf{epcf}_b$. We say α or β corresponds to $(k, t-1, t)$ if it indicates the unique path to $\mathsf{v}^* = (\mathsf{epcf}_{k,t-1}, \mathsf{epcf}_{k,t})$ from a root of depth λ (a complete tree). Additionally, define the following extraction algorithms:

- $\mathsf{Extract}_{\mathsf{cf}}(\mathsf{td}, \Pi^{(b)}) \to (\mathsf{epcf}^{(b)}_{k,t-1}, \mathsf{epcf}^{(b)}_{k,t})$. It finds $(k, t-1, t)$ corresponding to α (w.l.o.g. $\mathsf{td}_{\mathsf{cf},\ell}$ includes $\alpha[\ell]$ for all $\ell \in [\lambda]$). Then locates $\mathsf{v}^{(b)}_d \in \Pi^{(b)}$ (where $d = \mathsf{depth}(v_d)$) such that it includes $(\mathsf{pcf}_{k,t-1}, \mathsf{pcf}_{k,t})$. Then recursively uses $\mathsf{v}^{(b)}_{\ell-1} = \mathsf{SEH}.\mathsf{Extract}(\mathsf{td}, \mathsf{v}^{(b)}_\ell)$ for $\ell \in [d]$ to find $(\mathsf{epcf}^{(b)}_{k,t-1}, \mathsf{epcf}^{(b)}_{k,t})$. Since this algorithm only uses $\mathsf{v}^{(b)} \in \pi^{(b)}_\ell \in \Pi^{(b)}$, we can w.l.o.g. let $\mathsf{Extract}_{\mathsf{cf}}(\mathsf{td}, \Pi^{(b)}) \equiv \mathsf{Extract}_{\mathsf{cf}}(\mathsf{td}, \pi^{(b)}_\ell) \equiv \mathsf{Extract}_{\mathsf{cf}}(\mathsf{td}, \mathsf{v}^{(b)})$.

- $\mathsf{Extract}_{\mathsf{bp}}(\mathsf{td}, \Pi^{(b)}) \to w^{(b)}$. It finds $(k, t-1, t)$ corresponding to β, then locates $\pi^{(b)}_\ell = (\mathsf{v}^{(b)}, \rho^{(b)}_1, \rho^{(b)}_2, \mathsf{h}^{(b)}, \hat{\pi}^{(b)}) \in \Pi^{(b)}$ such that it includes a proof for $(\mathsf{pcf}_{k,t-1}, \mathsf{pcf}_{k,t})$. Then it computes $w^{(b)} \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_{\mathsf{bp},\ell}, (\mathsf{h}^{(b)}, \mathsf{v}^{(b)}, j)_{j\in[2]}, \hat{\pi}^{(b)})$. Since this algorithm only uses $\hat{\pi}^{(b)} \in \pi^{(b)}_\ell \in \Pi^{(b)}$, we can w.l.o.g. let $\mathsf{Extract}_{\mathsf{bp}}(\mathsf{td}, \Pi^{(b)}) \equiv \mathsf{Extract}_{\mathsf{bp}}(\mathsf{td}, \pi^{(b)}_\ell) \equiv \mathsf{Extract}_{\mathsf{bp}}(\mathsf{td}, \hat{\pi}^{(b)})$.

- $\mathsf{Extract}_r(\mathsf{td}, \Pi^{(b)}) \to (\mathsf{epcf}^{(b)}_r)$. It finds $(k, t, t+1)$ corresponding to β. Let the path to $(k, t, t+1)$ be $(1, \cdots, 1, 2, \beta[j+1], \cdots, \beta[\lambda])$ for some $j \in [\lambda]$ (number of 1's at the beginning can be 0). In words this algorithm extracts using batch proofs until finding a proof at level j and

its witness, then it uses path opening from *right* to find an encrypted pseudo-configuration $\mathsf{epcf}_r^{(b)}$ corresponding to $\mathsf{cf}_{k,t}$.

Formally, $\mathsf{Extract}_r$ finds a proof $\pi_\ell^{(b)} = (\mathsf{v}^{(b)}, \rho_1^{(b)}, \rho_2^{(b)}, \mathsf{h}^{(b)}, \hat{\pi}^{(b)}) \in \Pi^{(b)}$ such that it contains a poof for $(\mathsf{pcf}_{k,t}, \mathsf{pcf}_{k,t+1})$. Now find $\mathsf{epcf}_r^{(b)}$ as follows:

- If (k,t) is the leftmost config in the tree with root $\mathsf{v}^{(b)}$ (e.g., $\ell < j$), then let $\rho_1^{(b)} = ((\mathsf{v}_{0,1}^{(b)}, \mathsf{v}_{0,2}^{(b)}), \ldots, (\mathsf{v}_{\ell-1,1}^{(b)}, \mathsf{v}_{\ell-1,2}^{(b)}))$, $\mathsf{v}_{0,1}^{(b)} = (\mathsf{epcf}_r^{(b)}, \mathsf{epcf}')$.
- Otherwise, it holds that $\ell \geq j$, then recursively use $\mathsf{seBARG.Extract}(\mathsf{td}_{\mathsf{bp},d}, (x_{d,1}^{(b)}, x_{d,2}^{(b)})) = w_d^{(b)}$ for $d \in [j,\ell]$ where $x_{d,i}^{(b)} = (\tilde{\mathsf{h}}_d^{(b)}, \tilde{\mathsf{v}}_d^{(b)}, i)$ for $i \in [2]$ constructed as follows, (1) $(\tilde{\mathsf{h}}_\ell^{(b)}, \tilde{\mathsf{v}}_\ell^{(b)}) = (\mathsf{v}^{(b)}, \mathsf{h}^{(b)})$, (2) find $(\tilde{\mathsf{v}}_{d-1,1}^{(b)}, \tilde{\mathsf{v}}_{d-1,2}^{(b)}) \in w_d^{(b)}$ and let $\tilde{\mathsf{v}}_{d-1}^{(b)} = \tilde{\mathsf{v}}_{d-1,\beta[d]}^{(b)}$, and (3) if $d < \log T$ let $\tilde{\mathsf{h}}_{d-1}^{(b)} = \tilde{\mathsf{h}}_d^{(b)}$, otherwise Find $(\tilde{\mathsf{h}}_{d-1,1}^{(b)}, \tilde{\mathsf{h}}_{d-1,2}^{(b)}) \in w_d^{(b)}$ and let $\tilde{\mathsf{h}}_{d-1}^{(b)} = \tilde{\mathsf{h}}_{d-1,\beta[d]}^{(b)}$. Finally find $(\tilde{\mathsf{v}}_{j-1,2}^{(b)}, \tilde{\rho}_{j-1,2}^{(b)}) \in w_j^{(b)}$ and do the following, (1) if $j = 1$ then let $\tilde{\mathsf{v}}_{j-1,2}^{(b)} = (\mathsf{epcf}_r^{(b)}, \mathsf{epcf}')$, (2) otherwise let $\rho_{j-1,2}^{(b)} = ((\mathsf{v}_{0,1}^{(b)}, \mathsf{v}_{0,2}^{(b)}), \ldots, (\mathsf{v}_{j-2,1}^{(b)}, \mathsf{v}_{j-2,2}^{(b)}))$, and $\mathsf{v}_{0,1}^{(b)} = (\mathsf{epcf}_r^{(b)}, \mathsf{epcf}')$.

- $\mathsf{Extract}_l(\mathsf{td}, \Pi^{(b)}) \to (\mathsf{epcf}_l^{(b)})$. This is similar to $\mathsf{Extract}_r$ except that in the end it uses path opening from *left* to find an encrypted pseudo-configuration $\mathsf{epcf}_l^{(b)}$. Namely, it finds $(k,t,t{+}1)$ corresponding to β. Let the path to $(k,t,t+1)$ be $(1,\cdots,1,2,\beta[j+1],\cdots,\beta[\lambda])$ for some $j \in [\lambda]$ (number of 1's at the beginning can be 0).

  Formally, $\mathsf{Extract}_l$ finds a proof $\pi_\ell^{(b)} = (\mathsf{v}^{(b)}, \rho_1^{(b)}, \rho_2^{(b)}, \mathsf{h}^{(b)}, \hat{\pi}^{(b)}) \in \Pi^{(b)}$ such that it contains a poof for $(\mathsf{pcf}_{k,t-1}, \mathsf{pcf}_{k,t})$. Now find $\mathsf{epcf}_l^{(b)}$ as follows:

  - If (k,t) is the rightmost config in the tree with root $\mathsf{v}^{(b)}$ (e.g., $\ell < j$), then let $\rho_2^{(b)} = ((\mathsf{v}_{0,1}^{(b)}, \mathsf{v}_{0,2}^{(b)}), \ldots, (\mathsf{v}_{\ell-1,1}^{(b)}, \mathsf{v}_{\ell-1,2}^{(b)}))$, $\mathsf{v}_{0,2}^{(b)} = (\mathsf{epcf}', \mathsf{epcf}_l^{(b)})$.
  - Otherwise similar to $\mathsf{Extract}_r$ find $w_j^{(b)}$. Finally find $(\tilde{\mathsf{v}}_{j-1,1}^{(b)}, \tilde{\rho}_{j-1,1}^{(b)}) \in w_j^{(b)}$ and do the following, (1) if $j = 1$ then let $\tilde{\mathsf{v}}_{j-1,1}^{(b)} = (\mathsf{epcf}', \mathsf{epcf}_l^{(b)})$, (2) otherwise let $\rho_{j-1,1}^{(b)} = ((\mathsf{v}_{0,1}^{(b)}, \mathsf{v}_{0,2}^{(b)}), \ldots, (\mathsf{v}_{j-2,1}^{(b)}, \mathsf{v}_{j-2,2}^{(b)}))$, $\mathsf{v}_{0,2}^{(b)} = (\mathsf{epcf}', \mathsf{epcf}_l^{(b)})$.

For any stateful PPT adversary $\mathcal{A}$ and any $\lambda \in \mathbb{N}$ define the experiment $\exp_{\alpha,\beta}$ (where α

corresponds to $(k, t-1, t)$) as follows:

$$\exp_{\alpha,\beta} \coloneqq \left[\begin{array}{l} n, S, T \leftarrow \mathcal{A}(1^\lambda), \\ (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Gen}_{\alpha,\beta}(1^\lambda, n, S, T), \\ (\mathsf{hz}, (\mathsf{out}^{(0)}, \Pi^{(0)}), (\mathsf{out}^{(1)}, \Pi^{(1)}), 1^{N \cdot T}) \leftarrow \mathcal{A}(\mathsf{crs}), \\ (\mathsf{epcf}^{(b)}_{k,t-1}, \mathsf{epcf}^{(b)}_{k,t}) \leftarrow \mathsf{Extract}_{\mathsf{cf}}(\mathsf{td}, \Pi^{(b)}) \text{ for } b \in \{0,1\}, \\ \mathsf{pcf}^{(b)}_{k,t-b'} = \mathsf{PKE.Dec}(\mathsf{sk}, \mathsf{epcf}^{(b)}_{k,t-b'}) \text{ for } b, b' \in \{0,1\}, \\ w^{(b)} \leftarrow \mathsf{Extract}_{\mathsf{bp}}(\mathsf{td}, \Pi^{(b)}) \text{ for } b \in \{0,1\}, \\ \mathsf{epcf}^{(b)}_{r} \leftarrow \mathsf{Extract}_{r}(\mathsf{td}, \Pi^{(b)}) \text{ for } b \in \{0,1\}, \\ \mathsf{pcf}^{(b)}_{r} = \mathsf{PKE.Dec}(\mathsf{sk}, \mathsf{epcf}^{(b)}_{r}) \text{ for } b \in \{0,1\}, \\ \mathsf{epcf}^{(b)}_{l} \leftarrow \mathsf{Extract}_{l}(\mathsf{td}, \Pi^{(b)}) \text{ for } b \in \{0,1\}, \\ \mathsf{pcf}^{(b)}_{l} = \mathsf{PKE.Dec}(\mathsf{sk}, \mathsf{epcf}^{(b)}_{l}) \text{ for } b \in \{0,1\} \end{array}\right]$$

Note that if $\alpha = \beta = 1^\lambda$, then $\exp_{\alpha,\beta}$ is the same as the strong soundness experiment except that the setup algorithm uses NIZK extractor to generate $\mathsf{crs}_{\mathsf{zk}}$ and the experiment performs some additional computations on $\mathcal{A}$'s output.

We define the following hybrids.

$\mathcal{H}_0$: This is the original experiment.

$\mathcal{H}_{k,t,1}$: This is $\exp_{\alpha,\beta}$ where α and β correspond to $(k, t-1, t)$.

$\mathcal{H}_{k,t,2}$: This is $\exp_{\alpha,\beta}$ where α corresponds to $(k, t-1, t)$, and β corresponds to $(k, t, t+1)$.

$\mathcal{H}_{k,t,3}$: This is $\exp_{\alpha,\beta}$ where α and β correspond to $(k, t, t+1)$. Note that this hybrid is equal to $\mathcal{H}_{k,t+1,1}$ (or $\mathcal{H}_{k+1,1,1}$ if $t = T$).

If proofs $\Pi^{(0)}$ and $\Pi^{(1)}$ verify, we prove the followings hold except with a negligible probability:

- (Claim A.1) For any adversary $\mathcal{A}$ the probability that $\mathsf{out}^{(0)} \neq \mathsf{out}^{(1)}$ in $\mathcal{H}_0$ and $\mathcal{H}_{1,1,1}$ is negligibly close (henceforth, our goal is to prove $\Pr[\mathsf{Bad} : \exp_{\alpha,\beta}] \leq \mathsf{negl}(\lambda)$).
- (Claim A.2) In hybrid $\mathcal{H}_{1,1,1}$ $\mathsf{pcf}^{(0)}_{1,0} = \mathsf{pcf}^{(1)}_{1,0}$.
- (Claim A.3) In hybrid $\mathcal{H}_{k,t,1}$ if $\mathsf{pcf}^{(0)}_{k,t-1} = \mathsf{pcf}^{(1)}_{k,t-1}$ then $\mathsf{pcf}^{(0)}_{k,t} = \mathsf{pcf}^{(1)}_{k,t}$.
- (Claim A.4) $\mathcal{H}_{k,t,1}$ and $\mathcal{H}_{k,t,2}$ are indistinguishable.
- (Claim A.5) $\mathcal{H}_{k,t,2}$ and $\mathcal{H}_{k,t,3}$ are indistinguishable.
- (Claim A.6) In hybrid $\mathcal{H}_{N,T,1}$ if $\mathsf{pcf}^{(0)}_{N,T} = \mathsf{pcf}^{(1)}_{N,T}$ then the adversary cannot generate valid proofs $\Pi^{(0)}$ and $\Pi^{(1)}$ for two different final outputs $\mathsf{out}^{(0)}$ and $\mathsf{out}^{(1)}$.

Claim A.1. If NIZK extractor satisfies crs indisstinguishability and $\alpha = \beta = 1^\lambda$, then for any stateful PPT adversary $\mathcal{A}$ it holds that:

$$|\Pr[\mathsf{Bad} : \exp_{\alpha,\beta}] - \Pr[\mathsf{Bad} : \exp_{\mathsf{orig}}]| \leq \mathsf{negl}(\lambda).$$

Proof. The proof directly follows from the crs indistinguishability of the NIZK extractor. □

Claim A.2. If SEH is somewhere binding w.r.t. path opening, PKE is correct, and $\alpha = \beta = 1^\lambda$, then for any stateful PPT adversary $\mathcal{A}$ it holds that:

$$\Pr[\mathsf{Bad} \ \wedge \ \mathsf{pcf}^{(0)}_{1,0} \neq \mathsf{pcf}^{(1)}_{1,0} \ : \ \exp_{\alpha,\beta}] \leq \mathsf{negl}(\lambda).$$

Proof. We will prove that for $b \in \{0,1\}$, $\mathsf{pcf}^{(b)}_{1,0} = \mathsf{pcf}_{1,0}$ (as given in the crs). First let $\ell \in [\lambda]$ be the biggest index such that $\pi^{(b)}_\ell \neq \epsilon$, then parse $\pi^{(b)}_\ell = (\mathsf{v}^{(b)}, \rho^{(b)}_1, \rho^{(b)}_2, \mathsf{h}^{(b)}, \hat{\pi}^{(b)})$, $\rho^{(b)}_1 = ((\mathsf{v}^{(b)}_{0,1}, \mathsf{v}^{(b)}_{0,2}), \ldots, (\mathsf{v}^{(b)}_{\ell-1,1}, \mathsf{v}^{(b)}_{\ell-1,2}))$, and $\mathsf{v}^{(b)}_{0,1} = (\mathsf{epcf}^{(b)}_{1,0}, \mathsf{epcf}^{(b)}_{1,1})$. Note that since $\alpha = \beta = 1^\lambda$, then $\mathsf{Extract}_{\mathsf{cf}}(\mathsf{td}, \Pi^{(b)}) = (\mathsf{epcf}^{(b)}_{1,0}, \mathsf{epcf}^{(b)}_{1,1})$ by somewhere binding w.r.t. path opening of SEH. Also, note that by the verifier's check $\mathsf{epcf}^{(b)}_{1,0} = \mathsf{PKE.Enc}(\mathsf{pk}, \mathsf{pcf}_{1,0}; 0^{|r|})$ for $b \in \{0,1\}$. Then by PKE correctness it holds that $\mathsf{pcf}^{(b)}_{1,0} = \mathsf{pcf}_{1,0}$, and hence we have $\mathsf{pcf}^{(0)}_{1,0} = \mathsf{pcf}^{(1)}_{1,0}$. □

Claim A.3. If seBARG is somewhere extractable, NIZK has knowledge extractor, PKE is correct, SEH is somewhere binding, HT has reading and writing soundness, and CRHF is collision resistant, and α and β correspond to $(k, t-1, t)$, and for any stateful PPT adversary $\mathcal{A}$ it holds that:

$$\Pr[\mathsf{Bad} \ \wedge \ \mathsf{pcf}^{(0)}_{k,t-1} \neq \mathsf{pcf}^{(1)}_{k,t-1} \ : \ \exp_{\alpha,\beta}] \leq \mathsf{negl}(\lambda),$$

then for any such adversary, it holds that:

$$\Pr[\mathsf{Bad} \ \wedge \ \mathsf{pcf}^{(0)}_{k,t} \neq \mathsf{pcf}^{(1)}_{k,t} \ : \ \exp_{\alpha,\beta}] \leq \mathsf{negl}(\lambda).$$

Proof. To prove the claim, by total law of the probablity we only need to show that for any stateful PPT adversary $\mathcal{A}$ it holds that:

$$\delta_0 = \Pr[\mathsf{Bad} \ \wedge \ \mathsf{pcf}^{(0)}_{k,t-1} = \mathsf{pcf}^{(1)}_{k,t-1} \ \wedge \ \mathsf{pcf}^{(0)}_{k,t} \neq \mathsf{pcf}^{(1)}_{k,t} \ : \ \exp_{\alpha,\beta}] \leq \mathsf{negl}(\lambda).$$

We prove this by induction on the depth ℓ of the proof $\pi^{(b)}_\ell$ that contains $(\mathsf{pcf}_{k,t-1}, \mathsf{pcf}_{k,t})$ (the depth is the same for both proofs as it only depends on the time T and number of inputs N). Note that this is a two-step induction, first on the depth $\ell \leq \log T - 1$, and then on the depth $\ell \geq \log T$. Also note that if $\ell < \log T - 1$ then there is no $\mathsf{h}^{(b)}$ in the proof $\pi^{(b)}_\ell$ that contains $(\mathsf{pcf}_{k,t-1}, \mathsf{pcf}_{k,t})$, thus in this case we let $\mathsf{h}^{(b)} = \mathsf{h}_{\log T - 1}$ where $\mathsf{h}_{\log T-1} \in \mathsf{hz}$.

Let $(\mathsf{h}^{(b)}, \mathsf{v}^{(b)}, \hat{\pi}^{(b)}) \in \pi^{(b)}_\ell$ and define Ver_ℓ be the following event:

$$\mathsf{Ver}_\ell := \left[\begin{array}{l} \mathsf{seBARG.Verify}(\mathsf{crs}_{\mathsf{bp},\ell}, (\mathsf{h}^{(0)}, \mathsf{v}^{(0)}, j)_{j\in[2]}, \hat{\pi}^{(0)}) = 1 \ \wedge \\ \mathsf{seBARG.Verify}(\mathsf{crs}_{\mathsf{bp},\ell}, (\mathsf{h}^{(1)}, \mathsf{v}^{(1)}, j)_{j\in[2]}, \hat{\pi}^{(1)}) = 1 \end{array} \right]$$

and

$$\delta_{1,\ell} = \Pr[\mathsf{Ver}_\ell \ \wedge \ \mathsf{pcf}^{(0)}_{k,t-1} = \mathsf{pcf}^{(1)}_{k,t-1} \ \wedge \ \mathsf{pcf}^{(0)}_{k,t} \neq \mathsf{pcf}^{(1)}_{k,t} \ : \ \exp_{\alpha,\beta}].$$

Since Ver_ℓ is implied by Bad, it holds that $\delta_{1,\ell} \geq \delta_0$, thus it suffices to prove that $\delta_{1,\ell} \leq \mathsf{negl}(\lambda)$.

In the following induction let $\mathsf{hz} = (\mathsf{h}_{\log T-1}, \ldots, \mathsf{h}_\lambda)$ and note that $\mathsf{h}^{(b)} = \mathsf{h}_{\log T-1}$ by the verifier's check.

Induction Base. Now suppose $\pi_1^{(b)} = (\mathsf{v}^{(b)}, \rho_1^{(b)}, \rho_2^{(b)}, \mathsf{h}^{(b)}, \hat{\pi}^{(b)}) \in \Pi^{(b)}$ is the proof that contains $(\mathsf{pcf}_{k,t-1}^{(b)}, \mathsf{pcf}_{k,t}^{(b)})$ (i.e. $\ell = 1$). This means $\mathsf{Extract}_{\mathsf{cf}}(\mathsf{td}, \Pi^{(b)}) = \mathsf{SEH.Extract}(\mathsf{td}_{\mathsf{cf},1}, \mathsf{v}^{(b)})$ and $w^{(b)} = \mathsf{Extract}_{\mathsf{bp}}(\mathsf{td}, \Pi^{(b)}) = \mathsf{seBARG.Extract}(\mathsf{td}_{\mathsf{bp},1}, (\mathsf{h}^{(b)}, \mathsf{v}^{(b)}, j)_{j\in[2]}, \hat{\pi}^{(b)}) = (\tilde{\mathsf{v}}_1^{(b)}, \tilde{\mathsf{v}}_2^{(b)}, \pi_{\mathsf{zk}}^{(b)})$, is a witness for $x^{(b)} = (\mathsf{h}^{(b)}, \mathsf{v}^{(b)}, \beta[1])$. Let $\tilde{\mathsf{v}}_{\beta[1]}^{(b)} = (\tilde{\mathsf{epcf}}_{k,t-1}^{(b)}, \tilde{\mathsf{epcf}}_{k,t}^{(b)})$.

Now let $\exp_{\alpha,\beta}$ additionally compute $\mathsf{NIZK}.\mathcal{E}(\mathsf{td}_{\mathsf{zk}}, x_{\mathsf{zk}}^{(b)} = (\mathsf{h}^{(b)}, \tilde{\mathsf{epcf}}_{k,t-1}^{(b)}, \tilde{\mathsf{epcf}}_{k,t}^{(b)}), \pi_{\mathsf{zk}}^{(b)}) = w_{\mathsf{zk}}^{(b)}$ and parse $w_{\mathsf{zk}}^{(b)} = (\mathsf{rt}_{\mathsf{in}}^{(b)}, r_{\mathsf{in}}^{(b)}, \tilde{\mathsf{pcf}}_{k,t-1}^{(b)}, r_{k,t-1}^{(b)}, \tilde{\mathsf{pcf}}_{k,t}^{(b)}, r_{k,t}^{(b)}, \mathsf{rbit}^{(b)}, \mathsf{rop}^{(b)}, \mathsf{wop}^{(b)})$.

Recall that $\mathcal{L}_{\mathsf{bp},1}$ from Fig. 4 is the corresponding language for $(x^{(b)}, w^{(b)})$ and $\mathcal{L}_{\mathsf{zk}}$ from Fig. 3 is the corresponding language for $(x_{\mathsf{zk}}^{(b)}, w_{\mathsf{zk}}^{(b)})$.

Now consider the following probabilities:

$$\delta_2 = \Pr\left[\begin{array}{ll} & (x^{(0)}, w^{(0)}) \in \mathcal{L}_{\mathsf{bp},1} \ \wedge \\ & \mathsf{seBARG.Verify}(\mathsf{crs}_{\mathsf{bp},1}, (\mathsf{h}^{(1)}, \mathsf{v}^{(1)}, j)_{j\in[2]}, \hat{\pi}^{(1)}) = 1 \ \wedge \\ & \mathsf{pcf}_{k,t-1}^{(0)} = \mathsf{pcf}_{k,t-1}^{(1)} \ \wedge \ \mathsf{pcf}_{k,t}^{(0)} \neq \mathsf{pcf}_{k,t}^{(1)} \end{array} : \exp_{\alpha,\beta}\right].$$

$$\delta_3 = \Pr\left[\begin{array}{ll} & (x^{(b)}, w^{(b)}) \in \mathcal{L}_{\mathsf{bp},1} \text{ for } b \in \{0,1\} \ \wedge \\ & \mathsf{pcf}_{k,t-1}^{(0)} = \mathsf{pcf}_{k,t-1}^{(1)} \ \wedge \ \mathsf{pcf}_{k,t}^{(0)} \neq \mathsf{pcf}_{k,t}^{(1)} \end{array} : \exp_{\alpha,\beta}\right].$$

$$\delta_4 = \Pr\left[\begin{array}{ll} & (\tilde{\mathsf{epcf}}_{k,t-1}^{(0)}, \tilde{\mathsf{epcf}}_{k,t}^{(0)}) = (\mathsf{epcf}_{k,t-1}^{(0)}, \mathsf{epcf}_{k,t}^{(0)}) \ \wedge \\ & \mathsf{NIZK.Verify}(\mathsf{crs}_{\mathsf{zk}}, x_{\mathsf{zk}}^{(0)}, \pi_{\mathsf{zk}}^{(0)}) = 1 \\ & (x^{(1)}, w^{(1)}) \in \mathcal{L}_{\mathsf{bp},1} \ \wedge \\ & \mathsf{pcf}_{k,t-1}^{(0)} = \mathsf{pcf}_{k,t-1}^{(1)} \ \wedge \ \mathsf{pcf}_{k,t}^{(0)} \neq \mathsf{pcf}_{k,t}^{(1)} \end{array} : \exp_{\alpha,\beta}\right].$$

$$\delta_5 = \Pr\left[\begin{array}{ll} & (\tilde{\mathsf{epcf}}_{k,t-1}^{(b)}, \tilde{\mathsf{epcf}}_{k,t}^{(b)}) = (\mathsf{epcf}_{k,t-1}^{(b)}, \mathsf{epcf}_{k,t}^{(b)}) \ \text{ for } b \in \{0,1\} \ \wedge \\ & \mathsf{NIZK.Verify}(\mathsf{crs}_{\mathsf{zk}}, x_{\mathsf{zk}}^{(b)}, \pi_{\mathsf{zk}}^{(b)}) = 1 \ \text{ for } b \in \{0,1\} \ \wedge \\ & \mathsf{pcf}_{k,t-1}^{(0)} = \mathsf{pcf}_{k,t-1}^{(1)} \ \wedge \ \mathsf{pcf}_{k,t}^{(0)} \neq \mathsf{pcf}_{k,t}^{(1)} \end{array} : \exp_{\alpha,\beta}\right].$$

$$\delta_6 = \Pr\left[\begin{array}{ll} & (\tilde{\mathsf{epcf}}_{k,t-1}^{(b)}, \tilde{\mathsf{epcf}}_{k,t}^{(b)}) = (\mathsf{epcf}_{k,t-1}^{(b)}, \mathsf{epcf}_{k,t}^{(b)}) \ \text{ for } b \in \{0,1\} \ \wedge \\ & (x_{\mathsf{zk}}^{(0)}, w_{\mathsf{zk}}^{(0)}) \in \mathcal{L}_{\mathsf{zk}} \ \wedge \\ & \mathsf{NIZK.Verify}(\mathsf{crs}_{\mathsf{zk}}, x_{\mathsf{zk}}^{(1)}, \pi_{\mathsf{zk}}^{(1)}) = 1 \ \wedge \\ & \mathsf{pcf}_{k,t-1}^{(0)} = \mathsf{pcf}_{k,t-1}^{(1)} \ \wedge \ \mathsf{pcf}_{k,t}^{(0)} \neq \mathsf{pcf}_{k,t}^{(1)} \end{array} : \exp_{\alpha,\beta}\right].$$

$$\delta_7 = \Pr\left[\begin{array}{ll} & (\tilde{\mathsf{epcf}}_{k,t-1}^{(b)}, \tilde{\mathsf{epcf}}_{k,t}^{(b)}) = (\mathsf{epcf}_{k,t-1}^{(b)}, \mathsf{epcf}_{k,t}^{(b)}) \ \text{ for } b \in \{0,1\} \ \wedge \\ & (x_{\mathsf{zk}}^{(b)}, w_{\mathsf{zk}}^{(b)}) \in \mathcal{L}_{\mathsf{zk}} \ \text{ for } b \in \{0,1\} \ \wedge \\ & \mathsf{pcf}_{k,t-1}^{(0)} = \mathsf{pcf}_{k,t-1}^{(1)} \ \wedge \ \mathsf{pcf}_{k,t}^{(0)} \neq \mathsf{pcf}_{k,t}^{(1)} \end{array} : \exp_{\alpha,\beta}\right].$$

$$\delta_8 = \Pr\left[\begin{array}{ll} & (\tilde{\mathsf{pcf}}_{k,t-1}^{(b)}, \tilde{\mathsf{pcf}}_{k,t}^{(b)}) = (\mathsf{pcf}_{k,t-1}^{(b)}, \mathsf{pcf}_{k,t}^{(b)}) \ \text{ for } b \in \{0,1\} \ \wedge \mathsf{rt}_{\mathsf{in}}^{(0)} = \mathsf{rt}_{\mathsf{in}}^{(1)} \ \wedge \\ & \mathsf{is\text{-}nxt\text{-}cnfg}_{\mathcal{M}}(\mathsf{hk}_{\mathsf{ht}}, \mathsf{rt}_{\mathsf{in}}^{(b)}, \tilde{\mathsf{pcf}}_{k,t-1}^{(b)}, \tilde{\mathsf{pcf}}_{k,t}^{(b)}, \mathsf{rbit}^{(b)}, \mathsf{rop}^{(b)}, \mathsf{wop}^{(b)}) = 1 \text{ for } b \in \{0,1\} \ \wedge \\ & \mathsf{pcf}_{k,t-1}^{(0)} = \mathsf{pcf}_{k,t-1}^{(1)} \ \wedge \ \mathsf{pcf}_{k,t}^{(0)} \neq \mathsf{pcf}_{k,t}^{(1)} \end{array} : \exp_{\alpha,\beta}\right].$$

$$\delta_9 = \Pr\left[\begin{array}{l} \mathsf{rbit}^{(0)} = \mathsf{rbit}^{(1)} \ \wedge \\ \mathbb{C}_{\mathcal{M}}(\tilde{\mathsf{q}}^{(b)}_{k,t-1}, \mathsf{rbit}^{(b)}) = (\hat{\mathsf{q}}^{(b)}_{k,t}, \hat{\mathsf{rid}}\mathsf{x}^{(b)}_{k,t}, \mathsf{widx}^{(b)}, \mathsf{wbit}^{(b)}) \ \wedge \\ (\tilde{\mathsf{q}}^{(b)}_{k,t}, \tilde{\mathsf{rid}}\mathsf{x}^{(b)}_{k,t}) = (\hat{\mathsf{q}}^{(b)}_{k,t}, \hat{\mathsf{rid}}\mathsf{x}^{(b)}_{k,t}) \ \wedge \\ \mathsf{HT.VfyWt}(\mathsf{hk}_{\mathsf{ht}}, \tilde{\mathsf{rt}}^{(b)}_{k,t-1}, \mathsf{widx}^{(b)}, \mathsf{wbit}^{(b)}, \tilde{\mathsf{rt}}^{(b)}_{k,t}, \mathsf{wop}^{(b)}) = 1 \ \text{ for } b \in \{0,1\} \ \wedge \\ (\tilde{\mathsf{pcf}}^{(b)}_{k,t-1}, \tilde{\mathsf{pcf}}^{(b)}_{k,t}) = (\mathsf{pcf}^{(b)}_{k,t-1}, \mathsf{pcf}^{(b)}_{k,t}) \text{ for } b \in \{0,1\} \ \wedge \\ \mathsf{pcf}^{(0)}_{k,t-1} = \mathsf{pcf}^{(1)}_{k,t-1} \ \wedge \ \mathsf{pcf}^{(0)}_{k,t} \neq \mathsf{pcf}^{(1)}_{k,t} \end{array} : \ \exp_{\alpha,\beta} \right].$$

We let $\delta_1 = \delta_{1,1}$, and $\epsilon_i = \delta_i - \delta_{i+1}$ for $i \in [8]$ and $\epsilon_9 = \delta_9$. Since $\delta_1 = \sum_{i=1}^{6} \epsilon_i$ we only need to show that for $i \in [9]$, ϵ_i is negligible. Now the following Eqs. (3) to (8) holds.

Case 1. For $b \in \{0,1\}$ it holds that:

$$\Pr[\mathsf{seBARG.Verify}(\mathsf{crs}_{\mathsf{bp},1}, (\mathsf{h}^{(b)}, \mathsf{v}^{(b)}, j)_{j\in[2]}, \hat{\pi}^{(b)}) = 1 \ \wedge \ (x^{(b)}, w^{(b)}) \notin \mathcal{L}_{\mathsf{bp},1} : \exp_{\alpha,\beta}] \geq \epsilon_{b+1} \tag{3}$$

Suppose towards the contradiction that for some $b^* \in \{0,1\}$, $\epsilon_{b^*+1}(\cdot)$ is non-negligible. Then we construct $\mathcal{B}_{\mathsf{bp}}$ s.t. $\mathcal{B}_{\mathsf{bp}} = (\alpha, \beta, 1, \mathcal{A}, b^*)$ breaks the seBARG security. Note that the advantage of $\mathcal{B}_{\mathsf{bp}}(\alpha, \beta, 1, \mathcal{A}, b^*)$ is greater than that of $\mathcal{A}$ in Eq. (3) contradicting the seBARG security.

$\mathcal{B}_{\mathsf{bp}}(\alpha, \beta, \ell, \mathcal{A}, b)$ This PPT attacker does the following:

1. Receives (n, S, T) from $\mathcal{A}$ and crs^* from the challenger where $(\mathsf{crs}^*, \mathsf{td}^*) \leftarrow \mathsf{seBARG.Gen}(1^\lambda, 2, n_{\mathsf{bp},\ell}, \beta[\ell])$.
2. Computes $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Gen}_{\alpha,\beta}(1^\lambda, n, S, T)$ and updates crs by replacing $\mathsf{crs}_{\mathsf{bp},\ell}$ with crs^*.
3. Sends crs to $\mathcal{A}$ and receives $(\mathsf{hz}, (\mathsf{out}^{(0)}, \Pi^{(0)}), (\mathsf{out}^{(1)}, \Pi^{(1)}), 1^{N\cdot T})$.
4. Finds $\pi^{(b)}_\ell = (\mathsf{v}^{(b)}, \rho^{(b)}_1, \rho^{(b)}_2, \mathsf{h}^{(b)}, \hat{\pi}^{(b)}) \in \Pi^{(b)}$, and output $((\mathsf{h}^{(b)}, \mathsf{v}^{(b)}, j)_{j\in[2]}, \hat{\pi}^{(b)})$.

Case 2. For $b \in \{0,1\}$ it holds that:

$$\Pr\left[\begin{array}{l} \mathsf{SEH.Hash}(\mathsf{hk}_{\mathsf{cf},1}, (\tilde{\mathsf{v}}^{(b)}_1, \tilde{\mathsf{v}}^{(b)}_2)) = \mathsf{v}^{(b)} \ \wedge \\ (\tilde{\mathsf{epcf}}^{(b)}_{k,t-1}, \tilde{\mathsf{epcf}}^{(b)}_{k,t}) \neq (\mathsf{epcf}^{(b)}_{k,t-1}, \mathsf{epcf}^{(b)}_{k,t}) \end{array} : \ \exp_{\alpha,\beta} \right] \geq \epsilon_{b+3} \tag{4}$$

Suppose towards the contradiction that for some $b^* \in \{0,1\}$, $\epsilon_{b^*+3}(\cdot)$ is non-negligible. Then we construct $\mathcal{B}_{\mathsf{seh}}$ s.t. $\mathcal{B}_{\mathsf{seh}} = (\alpha, \beta, 1, \mathcal{A}, b^*)$ breaks the SEH security. Note that the advantage of $\mathcal{B}_{\mathsf{seh}}(\alpha, \beta, 1, \mathcal{A}, b^*)$ is greater than that of $\mathcal{A}$ in Eq. (4) contradicting the SEH security.

$\mathcal{B}_{\mathsf{seh}}(\alpha, \beta, \ell, \mathcal{A}, b)$ This PPT attacker does the following:

1. Receives (n, S, T) from $\mathcal{A}$ and hk^* from the challenger where $(\mathsf{hk}^*, \mathsf{td}^*) \leftarrow \mathsf{SEH.Gen}(1^\lambda, \Sigma^{\ell-1}_{\mathsf{cf}}, 2, \alpha[\ell])$.
2. Computes $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Gen}_{\alpha,\beta}(1^\lambda, n, S, T)$ and updates crs by replacing $\mathsf{hk}_{\mathsf{cf},\ell}$ with hk^*.

3. Sends crs to $\mathcal{A}$ and receives $(\mathsf{hz}, (\mathsf{out}^{(0)}, \Pi^{(0)}), (\mathsf{out}^{(1)}, \Pi^{(1)}), 1^{N \cdot T})$.
4. Finds $\pi_\ell^{(b)} = (\mathsf{v}^{(b)}, \rho_1^{(b)}, \rho_2^{(b)}, \mathsf{h}^{(b)}, \hat{\pi}^{(b)}) \in \Pi^{(b)}$.
5. Computes $w^{(b)} = \mathsf{seBARG.Extract}(\mathsf{td}_{\mathsf{bp},\ell}, (\mathsf{h}^{(b)}, \mathsf{v}^{(b)}, j)_{j\in[2]}, \hat{\pi}^{(b)})$ and finds $(\tilde{\mathsf{v}}_1^{(b)}, \tilde{\mathsf{v}}_2^{(b)}) \in w^{(b)}$.
6. Outputs $((\tilde{\mathsf{v}}_1^{(b)}, \tilde{\mathsf{v}}_2^{(b)}), \mathsf{v}^{(b)})$

Case 3. For $b \in \{0,1\}$ it holds that:

$$\Pr[\mathsf{NIZK.Verify}(\mathsf{crs}_{\mathsf{zk}}, (\mathsf{h}^{(b)}, \tilde{\mathsf{epcf}}_{k,t-1}^{(b)}, \tilde{\mathsf{epcf}}_{k,t}^{(b)}), \pi_{\mathsf{zk}}^{(b)}) = 1 \ \wedge \ (x_{\mathsf{zk}}^{(b)}, w_{\mathsf{zk}}^{(b)}) \notin \mathcal{L}_{\mathsf{zk}} : \exp_{\alpha,\beta}] \geq \epsilon_{b+5} \quad (5)$$

Suppose towards the contradiction that for some $b^* \in \{0,1\}$, $\epsilon_{b^*+1}(\cdot)$ is non-negligible. Then we construct $\mathcal{B}_{\mathsf{zk}}$ s.t. $\mathcal{B}_{\mathsf{zk}} = (\alpha, \beta, 1, \mathcal{A}, b^*)$ breaks the NIZK knowledge extractor property. Note that the advantage of $\mathcal{B}_{\mathsf{zk}}(\alpha, \beta, 1, \mathcal{A}, b^*)$ is greater than that of $\mathcal{A}$ in Eq. (3) contradicting NIZK security.

$\mathcal{B}_{\mathsf{zk}}(\alpha, \beta, \ell, \mathcal{A}, b)$ This PPT attacker does the following:

1. Receives (n, S, T) from $\mathcal{A}$ and crs^* from the challenger where $(\mathsf{crs}^*, \mathsf{td}^*) \leftarrow \mathsf{NIZK}.\mathcal{E}(1^\lambda, 1^{n_x}, 1^{n_w})$.
2. Computes $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Gen}_{\alpha,\beta}(1^\lambda, n, S, T)$ and updates crs by replacing $\mathsf{crs}_{\mathsf{zk}}$ with crs^*.
3. Sends crs to $\mathcal{A}$ and receives $(\mathsf{hz}, (\mathsf{out}^{(0)}, \Pi^{(0)}), (\mathsf{out}^{(1)}, \Pi^{(1)}), 1^{N \cdot T})$.
4. Finds $\pi_\ell^{(b)} = (\mathsf{v}^{(b)}, \rho_1^{(b)}, \rho_2^{(b)}, \mathsf{h}^{(b)}, \hat{\pi}^{(b)}) \in \Pi^{(b)}$.
5. Compute $w^{(b)} = \mathsf{seBARG.Extract}(\mathsf{td}_{\mathsf{bp},1}, (\mathsf{h}^{(b)}, \mathsf{v}^{(b)}, j)_{j\in[2]}, \hat{\pi}^{(b)})$.
6. Parse $w^{(b)} = (\tilde{\mathsf{v}}_1^{(b)}, \tilde{\mathsf{v}}_2^{(b)}, \pi_{\mathsf{zk}}^{(b)})$, and let $\tilde{\mathsf{v}}_{\beta[1]}^{(b)} = (\tilde{\mathsf{epcf}}_{k,t-1}^{(b)}, \tilde{\mathsf{epcf}}_{k,t}^{(b)})$.
7. Output $((\mathsf{h}^{(b)}, \tilde{\mathsf{epcf}}_{k,t-1}^{(b)}, \tilde{\mathsf{epcf}}_{k,t}^{(b)}), \pi_{\mathsf{zk}}^{(b)})$.

Case 4. We claim $\epsilon_7 = 0$ (6).

Note that if for $b \in \{0,1\}$, $(x_{\mathsf{zk}}^{(b)}, w_{\mathsf{zk}}^{(b)}) \in \mathcal{L}_{\mathsf{zk}}$ (where $x_{\mathsf{zk}}^{(b)} = (\mathsf{h}^{(b)}, \tilde{\mathsf{epcf}}_{k,t-1}^{(b)}, \tilde{\mathsf{epcf}}_{k,t}^{(b)})$ and $w_{\mathsf{zk}}^{(b)} = (\mathsf{rt}_{\mathsf{in}}^{(b)}, r_{\mathsf{in}}^{(b)}, \tilde{\mathsf{pcf}}_{k,t-1}^{(b)}, r_{k,t-1}^{(b)}, \tilde{\mathsf{pcf}}_{k,t}^{(b)}, r_{k,t}^{(b)}, \mathsf{rbit}^{(b)}, \mathsf{rop}^{(b)}, \mathsf{wop}^{(b)})$), then by the definition of $\mathcal{L}_{\mathsf{zk}}$ (see *Fig.* 3) for $b \in \{0,1\}$ it holds that $\mathsf{PKE.Enc}(\mathsf{pk}, \mathsf{rt}_{\mathsf{in}}^{(b)}; r_{\mathsf{in}}^{(b)}) = \mathsf{h}^{(b)}$, $\mathsf{PKE.Enc}(\mathsf{pk}, \tilde{\mathsf{pcf}}_{k,t-1}^{(b)}; r_{k,t-1}^{(b)}) = \tilde{\mathsf{epcf}}_{k,t-1}^{(b)}$, and $\mathsf{PKE.Enc}(\mathsf{pk}, \tilde{\mathsf{pcf}}_{k,t}^{(b)}; r_{k,t}^{(b)}) = \tilde{\mathsf{epcf}}_{k,t}^{(b)}$.

Now first note that, as discussed in the beginning of the claim's proof, it holds that $\mathsf{h}^{(0)} = \mathsf{h}^{(1)} = \mathsf{h}_{\log T - 1}$, thus by the perfect correctness of PKE for $b \in \{0,1\}$ we have $\mathsf{rt}_{\mathsf{in}}^{(0)} = \mathsf{rt}_{\mathsf{in}}^{(1)}$. Moreover, by the definition of $\exp_{\alpha,\beta}$ for $b, b' \in \{0,1\}$ it holds that $\mathsf{pcf}_{k,t-b'}^{(b)} = \mathsf{PKE.Dec}(\mathsf{sk}, \mathsf{epcf}_{k,t-b'}^{(b)})$. Thus if for $b \in \{0,1\}$ it holds that $(\tilde{\mathsf{epcf}}_{k,t-1}^{(b)}, \tilde{\mathsf{epcf}}_{k,t}^{(b)}) = (\mathsf{epcf}_{k,t-1}^{(b)}, \mathsf{epcf}_{k,t}^{(b)})$, then by the perfect correctness of PKE for $b \in \{0,1\}$ we have $(\tilde{\mathsf{pcf}}_{k,t-1}^{(b)}, \tilde{\mathsf{pcf}}_{k,t}^{(b)}) = (\mathsf{pcf}_{k,t-1}^{(b)}, \mathsf{pcf}_{k,t}^{(b)})$. By the same argument, the other direction can be proved, therefore, we conclude that $\delta_7 = \delta_8$ and $\epsilon_7 = 0$.

Case 5. If $\mathsf{ridx}^{(0)}_{k,t-1} \leq n$ then let $(m^*, i^*) = (\mathsf{h}_{\log T-1}, \mathsf{ridx}^{(0)}_{k,t-1})$, otherwise let $(m^*, i^*) = (\mathsf{rt}^{(0)}_{k,t-1}, \mathsf{ridx}^{(0)}_{k,t-1})$. Then it holds that:

$$\Pr\left[\begin{array}{l}\mathsf{rbit}^{(0)} \neq \mathsf{rbit}^{(1)} \wedge \\ \mathsf{HT.VfyRd}(\mathsf{hk}_{\mathsf{MT}}, m^*, i^*, \mathsf{rbit}^{(b)}, \mathsf{rop}^{(b)}) = 1 \ \wedge \quad \text{for } b \in \{0,1\}\end{array} \ : \ \exp_{\alpha,\beta}\right] \geq \epsilon_8 \tag{7}$$

Suppose towards the contradiction that $\epsilon_8(\cdot)$ is non-negligible. Then we construct $\mathcal{B}^{\mathsf{read}}_{\mathsf{ht}}$ s.t. $\mathcal{B}^{\mathsf{read}}_{\mathsf{ht}} = (\alpha, \beta, 1, \mathcal{A})$ breaks the hash tree security. Note that the advantage of $\mathcal{B}^{\mathsf{read}}_{\mathsf{ht}}(\alpha, \beta, 1, \mathcal{A})$ is greater than that of $\mathcal{A}$ in Eq. (7) contradicting the hash tree reading soundness.

$\mathcal{B}^{\mathsf{read}}_{\mathsf{ht}}(\alpha, \beta, \ell, \mathcal{A})$ This PPT attacker does the following:

1. Receives (n, S, T) from $\mathcal{A}$ and hk^* from the challenger where $\mathsf{hk}^* \leftarrow \mathsf{HT.Gen}(1^\lambda)$.
2. Computes $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Gen}_{\alpha,\beta}(1^\lambda, n, S, T)$ and updates crs by replacing $\mathsf{hk}_{\mathsf{ht}}$ with hk^*.
3. Sends crs to $\mathcal{A}$ and receives $(\mathsf{hz}, (\mathsf{out}^{(0)}, \Pi^{(0)}), (\mathsf{out}^{(1)}, \Pi^{(1)}), 1^{N \cdot T})$.
4. For $b \in \{0,1\}$, first finds $\pi^{(b)}_\ell = (\mathsf{v}^{(b)}, \rho^{(b)}_1, \rho^{(b)}_2, \mathsf{h}^{(b)}, \hat{\pi}^{(b)}) \in \Pi^{(b)}$, then computes $w^{(b)} = \mathsf{seBARG.Extract}(\mathsf{td}_{\mathsf{bp},\ell}, (\mathsf{h}^{(b)}, \mathsf{v}^{(b)}, j)_{j\in[2]}, \hat{\pi}^{(b)})$, $w^{(b)}_{\mathsf{zk}} = \mathsf{NIZK}.\mathcal{E}(\mathsf{td}_{\mathsf{zk}}, (\mathsf{h}^{(b)}, \tilde{\mathsf{epcf}}^{(b)}_{k,t-1}, \tilde{\mathsf{epcf}}^{(b)}_{k,t}), \pi^{(b)}_{\mathsf{zk}})$, and finally finds $(\mathsf{rbit}^{(b)}, \mathsf{rop}^{(b)}) \in w^{(b)}_{\mathsf{zk}}$.
5. Outputs $(m^*, i^*, \mathsf{rbit}^{(0)}, \mathsf{rop}^{(0)}, \mathsf{rbit}^{(1)}, \mathsf{rop}^{(1)})$ where $(m^*, i^*) = (\mathsf{h}_{\log T-1}, \mathsf{ridx}^{(0)}_{k,t-1})$ if $\mathsf{ridx}^{(0)}_{k,t-1} \leq n$, and $(m^*, i^*) = (\mathsf{rt}^{(0)}_{k,t-1}, \mathsf{ridx}^{(0)}_{k,t-1})$ otherwise.

Case 6. It holds that:

$$\Pr\left[\begin{array}{l}\mathbb{C}_{\mathcal{M}}(\mathsf{q}^{(0)}_{k,t-1}, \mathsf{rbit}^{(0)}) = (\mathsf{q}^{(b)}_{k,t}, \mathsf{ridx}^{(b)}_{k,t}, \mathsf{wbit}^{(0)}, \mathsf{widx}^{(0)}) \ \wedge \ \mathsf{rt}^{(0)}_{k,t} \neq \mathsf{rt}^{(1)}_{k,t} \wedge \\ \mathsf{HT.VfyWt}(\mathsf{hk}_{\mathsf{MT}}, \mathsf{rt}^{(0)}_{k,t-1}, \mathsf{widx}^{(0)}, \mathsf{wbit}^{(0)}, \mathsf{rt}^{(b)}_{k,t}, \mathsf{wop}^{(b)}) = 1 \text{ for } b \in \{0,1\}\end{array} \ : \ \exp_{\alpha,\beta}\right] \geq \epsilon_9 \tag{8}$$

Suppose towards the contradiction that $\epsilon_9(\cdot)$ is non-negligible. Then we construct $\mathcal{B}^{\mathsf{write}}_{\mathsf{ht}}$ s.t. $\mathcal{B}^{\mathsf{write}}_{\mathsf{ht}} = (\alpha, \beta, 1, \mathcal{A})$ breaks the hash tree security. Note that the advantage of $\mathcal{B}^{\mathsf{read}}_{\mathsf{ht}}(\alpha, \beta, 1, \mathcal{A})$ is greater than that of $\mathcal{A}$ in Eq. (8) contradicting the hash tree writing soundness.

$\mathcal{B}^{\mathsf{write}}_{\mathsf{ht}}(\alpha, \beta, \ell, \mathcal{A})$ This PPT attacker does the following:

1. Receives (n, S, T) from $\mathcal{A}$ and hk^* from the challenger where $\mathsf{hk}^* \leftarrow \mathsf{HT.Gen}(1^\lambda)$.
2. Computes $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Gen}_{\alpha,\beta}(1^\lambda, n, S, T)$ and updates crs by replacing $\mathsf{hk}_{\mathsf{ht}}$ with hk^*.
3. Sends crs to $\mathcal{A}$ and receives $(\mathsf{hz}, (\mathsf{out}^{(0)}, \Pi^{(0)}), (\mathsf{out}^{(1)}, \Pi^{(1)}), 1^{N \cdot T})$.
4. For $b \in \{0,1\}$, first finds $\pi^{(b)}_\ell = (\mathsf{v}^{(b)}, \rho^{(b)}_1, \rho^{(b)}_2, \mathsf{h}^{(b)}, \hat{\pi}^{(b)}) \in \Pi^{(b)}$, then computes $w^{(b)} = \mathsf{seBARG.Extract}(\mathsf{td}_{\mathsf{bp},\ell}, (\mathsf{h}^{(b)}, \mathsf{v}^{(b)}, j)_{j\in[2]}, \hat{\pi}^{(b)})$, $w^{(b)}_{\mathsf{zk}} = \mathsf{NIZK}.\mathcal{E}(\mathsf{td}_{\mathsf{zk}}, (\mathsf{h}^{(b)}, \tilde{\mathsf{epcf}}^{(b)}_{k,t-1}, \tilde{\mathsf{epcf}}^{(b)}_{k,t}), \pi^{(b)}_{\mathsf{zk}})$, and finally finds $(\mathsf{rbit}^{(b)}, \mathsf{wop}^{(b)}) \in w^{(b)}_{\mathsf{zk}}$.
5. Computes $\mathbb{C}_{\mathcal{M}}(\mathsf{q}^{(0)}_{k,t-1}, \mathsf{rbit}^{(0)}) = (\mathsf{q}^{(b)}_{k,t}, \mathsf{ridx}^{(b)}_{k,t}, \mathsf{wbit}^{(0)}, \mathsf{widx}^{(0)})$.
6. Outputs $(\mathsf{rt}^{(0)}_{k,t-1}, \mathsf{widx}^{(0)}, \mathsf{wbit}^{(0)}, \mathsf{rt}^{(0)}_{k,t}, \mathsf{wop}^{(0)}, \mathsf{rt}^{(1)}_{k,t}, \mathsf{wop}^{(1)})$.

Induction Step (1). Suppose the claim holds for any $\ell' < \ell \leq \log T - 1$, namely, $\delta_{1,\ell'} \leq \mathsf{negl}(\lambda)$. Let $\pi_\ell^{(b)} = (\mathsf{v}^{(b)}, \rho_1^{(b)}, \rho_2^{(b)}, \mathsf{h}^{(b)}, \hat{\pi}^{(b)}) \in \Pi^{(b)}$ be the proof that contains $(\mathsf{pcf}_{k,t-1}^{(b)}, \mathsf{pcf}_{k,t}^{(b)})$. Now the followings hold:

- $\mathsf{Extract}_{\mathsf{cf}}(\mathsf{td}, \Pi^{(b)}) = \mathsf{Extract}_{\mathsf{cf}}(\mathsf{td}, \mathsf{v}^{(b)}) = \mathsf{Extract}_{\mathsf{cf}}(\mathsf{td}, \mathsf{v}_{\alpha[\ell]}^{(b)})$ where $\mathsf{v}_{\alpha[\ell]}^{(b)} = \mathsf{SEH.Extract}(\mathsf{td}_{\mathsf{cf},\ell}, \mathsf{v}^{(b)})$. Let $\mathsf{Extract}_{\mathsf{cf}}(\mathsf{td}, \Pi^{(b)})$ additionally output $\mathsf{v}_{\alpha[\ell]}^{(b)}$.
- $w^{(b)} = \mathsf{Extract}_{\mathsf{bp}}(\mathsf{td}, \Pi^{(b)}) = \mathsf{seBARG.Extract}(\mathsf{td}_{\mathsf{bp},\ell}, (\mathsf{h}^{(b)}, \mathsf{v}^{(b)}, j)_{j\in[2]}, \hat{\pi}^{(b)}) = (\tilde{\mathsf{v}}_1^{(b)}, \tilde{\mathsf{v}}_2^{(b)}, \tilde{\rho}_1^{(b)}, \tilde{\rho}_2^{(b)}, \tilde{\pi}^{(b)})$, is a witness for $x^{(b)} = (\mathsf{h}^{(b)}, \mathsf{v}^{(b)}, \beta[\ell])$

Let $\mathcal{L}_{\mathsf{bp},\ell}$ be the corresponding language for $(x^{(b)}, w^{(b)})$ defined in the *Fig.* 4.

Now consider the following probabilities:

$$\delta_2 = \Pr\left[\begin{array}{ll} & (x^{(0)}, w^{(0)}) \in \mathcal{L}_{\mathsf{bp},\ell} \wedge \\ & \mathsf{seBARG.Verify}(\mathsf{crs}_{\mathsf{bp},\ell}, (\mathsf{h}^{(1)}, \mathsf{v}^{(1)}, j)_{j\in[2]}, \hat{\pi}^{(1)}) = 1 \wedge \\ & \mathsf{pcf}_{k,t-1}^{(0)} = \mathsf{pcf}_{k,t-1}^{(1)} \ \wedge \ \mathsf{pcf}_{k,t}^{(0)} \neq \mathsf{pcf}_{k,t}^{(1)} \end{array} : \ \exp_{\alpha,\beta}\right].$$

$$\delta_3 = \Pr\left[\begin{array}{ll} & (x^{(b)}, w^{(b)}) \in \mathcal{L}_{\mathsf{bp},\ell} \text{ for } b \in \{0,1\} \wedge \\ & \mathsf{pcf}_{k,t-1}^{(0)} = \mathsf{pcf}_{k,t-1}^{(1)} \ \wedge \ \mathsf{pcf}_{k,t}^{(0)} \neq \mathsf{pcf}_{k,t}^{(1)} \end{array} : \ \exp_{\alpha,\beta}\right].$$

$$\delta_4 = \Pr\left[\begin{array}{ll} & \tilde{\mathsf{v}}_{\alpha[\ell]}^{(0)} = \mathsf{v}_{\alpha[\ell]}^{(0)} \wedge \\ & \mathsf{seBARG.Verify}(\mathsf{crs}_{\mathsf{bp},\ell-1}, (\mathsf{h}^{(0)}, \tilde{\mathsf{v}}_{\alpha[\ell]}^{(0)}, j)_{j\in[2]}, \tilde{\pi}^{(0)}) = 1 \wedge \\ & (x^{(1)}, w^{(1)}) \in \mathcal{L}_{\mathsf{bp},1} \wedge \\ & \mathsf{pcf}_{k,t-1}^{(0)} = \mathsf{pcf}_{k,t-1}^{(1)} \ \wedge \ \mathsf{pcf}_{k,t}^{(0)} \neq \mathsf{pcf}_{k,t}^{(1)} \end{array} : \ \exp_{\alpha,\beta}\right].$$

$$\delta_5 = \Pr\left[\begin{array}{ll} & \tilde{\mathsf{v}}_{\alpha[\ell]}^{(b)} = \mathsf{v}_{\alpha[\ell]}^{(b)} \text{ for } b \in \{0,1\} \wedge \\ & \mathsf{seBARG.Verify}(\mathsf{crs}_{\mathsf{bp},\ell-1}, (\mathsf{h}^{(b)}, \tilde{\mathsf{v}}_{\alpha[\ell]}^{(b)}, j)_{j\in[2]}, \tilde{\pi}^{(b)}) = 1 \text{ for } b \in \{0,1\} \wedge \\ & \mathsf{pcf}_{k,t-1}^{(0)} = \mathsf{pcf}_{k,t-1}^{(1)} \ \wedge \ \mathsf{pcf}_{k,t}^{(0)} \neq \mathsf{pcf}_{k,t}^{(1)} \end{array} : \ \exp_{\alpha,\beta}\right].$$

We let $\delta_1 = \delta_{1,\ell}$, and $\epsilon_i = \delta_i - \delta_{i+1}$ for $i \in [4]$ and $\epsilon_5 = \delta_5$. Since $\delta_1 = \sum_{i=1}^{5} \epsilon_i$ we only need to show that for $i \in [5]$, ϵ_i is negligible.

Case 1. Similar to the case 1 in the induction base if ϵ_{b+2} is non-negligible then $\mathcal{B}_{\mathsf{bp}}(\alpha, \beta, \ell, \mathcal{A}, b)$ breaks the security of seBARG.

Case 2. Similar to the case 2 in the induction base if ϵ_{b+4} is non-negligible then $\mathcal{B}_{\mathsf{seh}}(\alpha, \beta, \ell, \mathcal{A}, b)$ breaks the security of SEH.

Case 3. Notice that $\epsilon_5 \leq \delta_{1,\ell-1}$, therefore, by the induction hypothesis, $\epsilon_5 \leq \mathsf{negl}(\lambda)$.

Induction Step (2). Suppose the claim holds for any $\ell' < \ell$ (where $\ell \geq \log T$), namely, $\delta_{1,\ell'} \leq \mathsf{negl}(\lambda)$. We follow the same approach as induction step (1) except that $w^{(b)}$ additionally includes $(\tilde{\mathsf{h}}_1^{(b)}, \tilde{\mathsf{h}}_2^{(b)})$. Thus we only need to show that $\tilde{\mathsf{h}}_{\alpha[\ell]}^{(0)} = \tilde{\mathsf{h}}_{\alpha[\ell]}^{(1)}$ to use the induction step. We let δ_2 and δ_3 to be the same as in induction step (1), and define the rest as follows:

$$\delta_4 = \Pr\left[\begin{array}{ll} \tilde{\mathsf{v}}_{\alpha[\ell]}^{(0)} = \mathsf{v}_{\alpha[\ell]}^{(0)} \wedge & \\ \mathsf{CRHF.Hash}(\mathsf{hk}_{\mathsf{ch}}, (\tilde{\mathsf{h}}_1^{(0)}, \tilde{\mathsf{h}}_2^{(0)})) = \mathsf{h}^{(0)} \wedge & \\ \mathsf{seBARG.Verify}(\mathsf{crs}_{\mathsf{bp},\ell-1}, (\tilde{\mathsf{h}}_{\alpha[\ell]}^{(0)}, \tilde{\mathsf{v}}_{\alpha[\ell]}^{(0)}, j)_{j\in[2]}, \tilde{\pi}^{(0)}) = 1 \wedge & : \quad \exp_{\alpha,\beta} \\ (x^{(1)}, w^{(1)}) \in \mathcal{L}_{\mathsf{bp},1} \wedge & \\ \mathsf{pcf}_{k,t-1}^{(0)} = \mathsf{pcf}_{k,t-1}^{(1)} \ \wedge \ \mathsf{pcf}_{k,t}^{(0)} \neq \mathsf{pcf}_{k,t}^{(1)} & \end{array}\right].$$

$$\delta_5 = \Pr\left[\begin{array}{ll} \tilde{\mathsf{v}}_{\alpha[\ell]}^{(b)} = \mathsf{v}_{\alpha[\ell]}^{(b)} \text{ for } b \in \{0,1\} \wedge & \\ \mathsf{CRHF.Hash}(\mathsf{hk}_{\mathsf{ch}}, (\tilde{\mathsf{h}}_1^{(b)}, \tilde{\mathsf{h}}_2^{(b)})) = \mathsf{h}^{(b)} \text{ for } b \in \{0,1\} \wedge & \\ \mathsf{seBARG.Verify}(\mathsf{crs}_{\mathsf{bp},\ell-1}, (\mathsf{h}_{\alpha[\ell]}^{(b)}, \tilde{\mathsf{v}}_{\alpha[\ell]}^{(b)}, j)_{j\in[2]}, \tilde{\pi}^{(b)}) = 1 \text{ for } b \in \{0,1\} \wedge & : \quad \exp_{\alpha,\beta} \\ \mathsf{pcf}_{k,t-1}^{(0)} = \mathsf{pcf}_{k,t-1}^{(1)} \ \wedge \ \mathsf{pcf}_{k,t}^{(0)} \neq \mathsf{pcf}_{k,t}^{(1)} & \end{array}\right].$$

$$\delta_6 = \Pr\left[\begin{array}{ll} \tilde{\mathsf{v}}_{\alpha[\ell]}^{(b)} = \mathsf{v}_{\alpha[\ell]}^{(b)} \text{ for } b \in \{0,1\} \ \wedge \tilde{\mathsf{h}}_{\alpha[\ell]}^{(0)} = \tilde{\mathsf{h}}_{\alpha[\ell]}^{(1)} \wedge & \\ \mathsf{seBARG.Verify}(\mathsf{crs}_{\mathsf{bp},\ell-1}, (\mathsf{h}_{\alpha[\ell]}^{(b)}, \tilde{\mathsf{v}}_{\alpha[\ell]}^{(b)}, j)_{j\in[2]}, \tilde{\pi}^{(b)}) = 1 \text{ for } b \in \{0,1\} \wedge & : \quad \exp_{\alpha,\beta} \\ \mathsf{pcf}_{k,t-1}^{(0)} = \mathsf{pcf}_{k,t-1}^{(1)} \ \wedge \ \mathsf{pcf}_{k,t}^{(0)} \neq \mathsf{pcf}_{k,t}^{(1)} & \end{array}\right].$$

Similar to induction step (1) define δ_1, and ϵ_i for $i \in [6]$.

Case 1 and 2. Similar to the case 1 and 2 in the induction step (1).

Case 3. It holds that:

$$\Pr\left[\begin{array}{ll} \tilde{\mathsf{h}}_{\alpha[\ell]}^{(0)} \neq \tilde{\mathsf{h}}_{\alpha[\ell]}^{(1)} \wedge & \\ \mathsf{CRHF.Hash}(\mathsf{hk}_{\mathsf{ch}}, (\tilde{\mathsf{h}}_1^{(b)}, \tilde{\mathsf{h}}_2^{(b)})) = \mathsf{h}_\ell \text{ for } b \in \{0,1\} & : \quad \exp_{\alpha,\beta} \end{array}\right] \geq \epsilon_5 \tag{9}$$

Suppose towards the contradiction that $\epsilon_5(\cdot)$ is non-negligible. Then we construct $\mathcal{B}_{\mathsf{ch}}$ s.t. $\mathcal{B}_{\mathsf{ch}}(\alpha, \beta, \ell, \mathcal{A})$ breaks the CRHF security. Note that the advantage of $\mathcal{B}_{\mathsf{ch}}(\alpha, \beta, \ell, \mathcal{A})$ is greater than that of $\mathcal{A}$ in Eq. (9) contradicting the CRHF security.

$\mathcal{B}_{\mathsf{ch}}(\alpha, \beta, \ell, \mathcal{A})$ This PPT attacker does the following:

1. Receives (n, S, T) from $\mathcal{A}$ and hk^* from the challenger where $\mathsf{hk}^* \leftarrow \mathsf{CRHF.Gen}(1^\lambda)$.
2. Computes $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Gen}_{\alpha,\beta}(1^\lambda, n, S, T)$ and updates crs by replacing $\mathsf{hk}_{\mathsf{ch}}$ with hk^*.
3. Sends crs to $\mathcal{A}$ and receives $(\mathsf{hz}, (\mathsf{out}^{(0)}, \Pi^{(0)}), (\mathsf{out}^{(1)}, \Pi^{(1)}), 1^{N \cdot T})$.
4. Finds $\pi_\ell^{(b)} = (\mathsf{v}^{(b)}, \rho_1^{(b)}, \rho_2^{(b)}, \mathsf{h}^{(b)}, \hat{\pi}^{(b)}) \in \Pi^{(b)}$.
5. Computes $w^{(b)} = \mathsf{seBARG.Extract}(\mathsf{td}_{\mathsf{bp},\ell}, (\mathsf{h}^{(b)}, \mathsf{v}^{(b)}, j)_{j\in[2]}, \hat{\pi}^{(b)})$ and finds $(\tilde{\mathsf{h}}_1^{(b)}, \tilde{\mathsf{h}}_2^{(b)}) \in w^{(b)}$.
6. Outputs $((\tilde{\mathsf{h}}_1^{(0)}, \tilde{\mathsf{h}}_2^{(0)}), (\tilde{\mathsf{h}}_1^{(1)}, \tilde{\mathsf{h}}_2^{(1)}))$.

Case 4. Notice that $\epsilon_6 \leq \delta_{1,\ell-1}$, therefore, by the induction hypothesis, $\epsilon_6 \leq \mathsf{negl}(\lambda)$.

□

Claim A.4. If seBARG is index hiding, α and β correspond to $(k, t-1, t)$, β' corresponds to $(k, t, t+1)$, and for any stateful PPT adversary $\mathcal{A}$ it holds that:

$$\Pr[\mathsf{Bad} \ \wedge \ \mathsf{pcf}^{(0)}_{k,t} \neq \mathsf{pcf}^{(1)}_{k,t} \ : \ \exp_{\alpha,\beta}] \leq \mathsf{negl}(\lambda),$$

then for any such adversary, it holds that:

$$\Pr[\mathsf{Bad} \ \wedge \ \mathsf{pcf}^{(0)}_{k,t} \neq \mathsf{pcf}^{(1)}_{k,t} \ : \ \exp_{\alpha,\beta'}] \leq \mathsf{negl}(\lambda).$$

Proof. There is some index j s.t. $\beta = (2, \cdots, 2, 1, \beta[j+1], \cdots, \beta[\lambda])$ and $\beta' = (1, \cdots, 1, 2, \beta[j+1], \cdots, \beta[\lambda])$. For $d \in [0, j]$ define β_d as $\beta_d[\ell] = \beta'[\ell]$ for $\ell \in [d]$ and $\beta_d[\ell] = \beta[\ell]$ at every other index. Therefore $\beta_0 = \beta$ (correspond to $(k, t-1, t)$) and $\beta_j = \beta'$ (corresponds to $(k, t, t+1)$). Note that every β_{d-1} and β_d differ only on index d.

If the claim doesn't hold, then there is an adversary $\mathcal{A}$ s.t. for some $d \in [j]$ there is a non-negligible function $\epsilon(\cdot)$ s.t.:

$$\Pr[\mathsf{Bad} \ \wedge \ \mathsf{pcf}^{(0)}_{k,t} \neq \mathsf{pcf}^{(1)}_{k,t} \ : \ \exp_{\alpha,\beta_d}] - \Pr[\mathsf{Bad} \ \wedge \ \mathsf{pcf}^{(0)}_{k,t} \neq \mathsf{pcf}^{(1)}_{k,t} \ : \ \exp_{\alpha,\beta_{d-1}}] \geq \epsilon(\lambda) \tag{10}$$

Now we construct $\mathcal{B}^{\mathsf{ih}}_{\mathsf{bp}}$ s.t. $\mathcal{B}^{\mathsf{ih}}_{\mathsf{seh}}(\alpha, \beta_d, d, \mathcal{A})$ breaks the index-hiding property of seBARG on two indices $e_1 = \beta_d[d]$ and $e_2 = \beta_{d-1}[d]$.

$\mathcal{B}^{\mathsf{ih}}_{\mathsf{bp}}(\alpha, \beta, \ell, \mathcal{A})$ This PPT attacker does the following:

1. Receives (n, S, T) from $\mathcal{A}$ and crs^* from the challenger where $(\mathsf{crs}^*, \mathsf{td}^*) \leftarrow \mathsf{seBARG.Gen}(1^\lambda, 2, n_{\mathsf{bp},\ell-1}, e_b)$.
2. Computes $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Gen}_{\alpha,\beta}(1^\lambda, n, S, T)$ and updates crs by replacing $\mathsf{crs}_{\mathsf{bp},\ell}$ with crs^*.
3. Sends crs to $\mathcal{A}$ and receives $(\mathsf{hz}, (\mathsf{out}^{(0)}, \Pi^{(0)}), (\mathsf{out}^{(1)}, \Pi^{(1)}), 1^{N \cdot T})$.
4. If for any $b \in \{0,1\}$, $\mathsf{Verify}(\mathsf{crs}, (\mathsf{hz}, \mathsf{out}^{(b)}), \Pi^{(b)}) = 0$ or $\mathsf{out}^{(0)} = \mathsf{out}^{(1)}$ then abort.
5. Finds $\pi^{(b)}_\ell = (\mathsf{v}^{(b)}, \rho_1^{(b)}, \rho_2^{(b)}, \mathsf{h}^{(b)}, \hat{\pi}^{(b)}) \in \Pi^{(b)}$.
6. Compute $(\mathsf{epcf}^{(b)}_{k,t-1}, \mathsf{epcf}^{(b)}_{k,t}) \leftarrow \mathsf{Extract}_{\mathsf{cf}}(\mathsf{td}, \Pi^{(b)})$ for $b \in \{0,1\}$.
7. Compute $\mathsf{pcf}^{(b)}_{k,t} = \mathsf{PKE.Dec}(\mathsf{sk}, \mathsf{epcf}^{(b)}_{k,t})$ for $b \in \{0,1\}$.
8. Output 1 if $\mathsf{pcf}^{(0)}_{k,t} \neq \mathsf{pcf}^{(1)}_{k,t}$ and 2 otherwise.

In the reduction above $b = 1$ corresponds to $\exp_{\alpha,\beta_d}$ and $b = 2$ corresponds to $\exp_{\alpha,\beta_{d-1}}$. Therefore, $\mathcal{B}^{\mathsf{ih}}_{\mathsf{bp}}(\alpha, \beta_d, d, \mathcal{A})$'s distinguishing advantage is greater than that of $\mathcal{A}$ in Eq. (10) contradicting the seBARG security.

□

Claim A.5. If seBARG, SEH, HT, and CRHF are secure, α corresponds to $(k, t-1, t)$, α' and β' correspond to $(k, t, t+1)$, and for any stateful PPT adversary $\mathcal{A}$ it holds that:

$$\Pr[\mathsf{Bad} \ \wedge \ \mathsf{pcf}^{(0)}_{k,t} \neq \mathsf{pcf}^{(1)}_{k,t} \ : \ \exp_{\alpha,\beta'}] \leq \mathsf{negl}(\lambda),$$

then for any such adversary, it holds that:

$$\Pr[\mathsf{Bad} \ \wedge \ \mathsf{pcf}^{(0)}_{k,t} \neq \mathsf{pcf}^{(1)}_{k,t} \ : \ \exp_{\alpha',\beta'}] \leq \mathsf{negl}(\lambda).$$

Proof. Consider the following probabilities:

$$\delta_0 = \Pr[\mathsf{Bad} \ \wedge \ \mathsf{pcf}^{(0)}_{k,t} \neq \mathsf{pcf}^{(1)}_{k,t} \ : \ \exp_{\alpha,\beta'}]$$

$$\delta_1 = \Pr[\mathsf{Bad} \ \wedge \ \mathsf{pcf}^{(0)}_{l} \neq \mathsf{pcf}^{(1)}_{l} \ : \ \exp_{\alpha,\beta'}]$$

$$\delta_2 = \Pr[\mathsf{Bad} \ \wedge \ \mathsf{pcf}^{(0)}_{r} \neq \mathsf{pcf}^{(1)}_{r} \ : \ \exp_{\alpha,\beta'}]$$

$$\delta_3 = \Pr[\mathsf{Bad} \ \wedge \ \mathsf{pcf}^{(0)}_{r} \neq \mathsf{pcf}^{(1)}_{r} \ : \ \exp_{\alpha',\beta'}]$$

$$\delta_4 = \Pr[\mathsf{Bad} \ \wedge \ \mathsf{pcf}^{(0)}_{k,t} \neq \mathsf{pcf}^{(1)}_{k,t} \ : \ \exp_{\alpha',\beta'}]$$

We let $\epsilon_i = \delta_i - \delta_{i-1}$ for $i \in [4]$. Since $\delta_4 = \delta_0 + \sum_{i=1}^{4} \epsilon_i$, and by the claim assumption $\delta_0 \leq \mathsf{negl}(\lambda)$, we only need to show that for $i \in [4]$, ϵ_i is negligible.

Case 1. It holds that:

$$\epsilon_1 \leq \Pr[\mathsf{Bad} \ \wedge \ \mathsf{pcf}^{(0)}_{l} \neq \mathsf{pcf}^{(1)}_{l} \ \wedge \ \mathsf{pcf}^{(0)}_{k,t} = \mathsf{pcf}^{(1)}_{k,t} \ : \ \exp_{\alpha,\beta'}]$$

Thus for some $b \in \{0,1\}$ it holds that:

$$\epsilon_1/2 \leq \Pr[\mathsf{Bad} \ \wedge \ \mathsf{pcf}^{(b)}_{k,t} \neq \mathsf{pcf}^{(b)}_{l} \ : \ \exp_{\alpha,\beta'}]$$

Now two cases could happen:

- $\mathsf{pcf}^{(b)}_{l}$ is computed using some $\rho^{(b)}_2 \in \pi^{(b)}_{\ell} \in \Pi^{(b)}$. In this case by the verifier's check it holds that $\mathsf{SEH.VerifyAcc}((\mathsf{hk}_{\mathsf{cf},d})_{d\in[\ell]}, \mathsf{v}^{(b)}, \rho^{(b)}_2, 2) = 1$ where $\mathsf{v}^{(b)} \in \pi^{(b)}_{\ell}$. Now since α corresponds to $(k, t-1, t)$, it is binding on the path $\rho^{(b)}_2$, thus by the SEH security and PKE correctness it holds that $\Pr[\mathsf{Bad} \ \wedge \ \mathsf{pcf}^{(b)}_{k,t} \neq \mathsf{pcf}^{(b)}_{l} \ : \ \exp_{\alpha,\beta'}] \leq \mathsf{negl}(\lambda)$.

- Find $\pi^{(b)}_{\ell} = (\mathsf{v}^{(b)}, \rho^{(b)}_1, \rho^{(b)}_2, \mathsf{h}^{(b)}, \hat{\pi}^{(b)}) \in \Pi^{(b)}$ that contains a proof for $(k, t-1, t)$. Note that $j < \ell$ thus α and β both are binding on the same path from a level ℓ proof until a level j proof. Additionally recall that to find $\mathsf{pcf}^{(b)}_{l}$, seBARG extraction is recursively used to find witness $w^{(b)}_j$. Then $w^{(b)}$ is parsed to get $(\tilde{\mathsf{v}}^{(b)}_{j-1,1}, \tilde{\rho}^{(b)}_{j-1,1}) \in w^{(b)}_j$ and $\mathsf{pcf}^{(b)}_{l}$ is found using them. Let $\mathsf{v}^{(b)}_{d}$ be the hash of pseudo-configurations at level d that is extracted by recursively running SEH extraction starting from $\mathsf{v}^{(b)}$. Now two cases can happen:

– $\tilde{\mathsf{v}}^{(b)}_{j-1} \neq \mathsf{v}^{(b)}_{j-1}$. This contradicts the seBARG and SEH security since both are binding on the same path from ℓ to j.
– $\tilde{\mathsf{v}}^{(b)}_{j-1} = \mathsf{v}^{(b)}_{j-1} \wedge \mathsf{pcf}^{(b)}_{k,t} \neq \mathsf{pcf}^{(b)}_{l}$. This contradicts either the somewhere statistically binding of SEH on the path from $\mathsf{v}^{(b)}_{j-1}$ to $\mathsf{epcf}^{(b)}_{l}$ or the PKE correctness.

Case 2. Similar to case 1, for some $b \in \{0,1\}$ it holds that:

$$\epsilon_2/2 \leq \Pr[\mathsf{Bad} \ \wedge \ \mathsf{pcf}^{(b)}_{r} \neq \mathsf{pcf}^{(b)}_{l} \ : \ \exp_{\alpha,\beta'}]$$

Two cases could happen:

- $\mathsf{pcf}^{(b)}_{l}$ (resp. $\mathsf{pcf}^{(b)}_{r}$) is computed using some $\rho^{(b)}_2 \in \pi^{(b)}_{\ell} \in \Pi^{(b)}$ (resp. $\rho^{(b)}_1 \in \pi^{(b)}_{\ell'} \in \Pi^{(b)}$) where $\ell > \ell'$. In this case by the verifier's check, the leftmost encrypted pseudo-configuration in $\pi^{(b)}_{\ell'}$ is the same as the rightmost encrypted pseudo-configuration in $\pi^{(b)}_{\ell}$, namely, $\mathsf{epcf}^{(b)}_{l} = \mathsf{epcf}^{(b)}_{r}$, thus by PKE correctness we have $\mathsf{pcf}^{(b)}_{l} = \mathsf{pcf}^{(b)}_{r}$.

- Otherwise, recall that that for both $\mathsf{pcf}^{(b)}_{l}$ and $\mathsf{pcf}^{(b)}_{r}$, first seBARG extractor is recursively being using to find $w^{(b)}_j$ for a batch proof at level j. Let $(\tilde{\mathsf{v}}^{(b)}_{j-1,1}, \tilde{\mathsf{v}}^{(b)}_{j-1,2}) \in w^{(b)}_j$, then by the definition of $\mathcal{L}_{\mathsf{bp},j}$, validity of $w^{(b)}_j$ (which is implied by the extraction correctness of batch argument) implies that the leftmost encrypted pseudo-configuration in the tree with root $\tilde{\mathsf{v}}^{(b)}_{j-1,2}$ is equal to the rightmost encrypted pseudo-configuration in $\tilde{\mathsf{v}}^{(b)}_{j-1,1}$, namely, $\mathsf{epcf}^{(b)}_{l} = \mathsf{epcf}^{(b)}_{r}$, thus by PKE correctness we have $\mathsf{pcf}^{(b)}_{l} = \mathsf{pcf}^{(b)}_{r}$.

Therefore in both cases it holds that $\Pr[\mathsf{Bad} \ \wedge \ \mathsf{pcf}^{(b)}_{r} \neq \mathsf{pcf}^{(b)}_{l} \ : \ \exp_{\alpha,\beta'}] \leq \mathsf{negl}(\lambda)$.

Case 3. Similar to the proof of Claim A.4 we prove $\epsilon_3 \leq \mathsf{negl}(\lambda)$. There is some index j s.t. $\alpha = (2, \cdots, 2, 1, \alpha[j+1], \cdots, \alpha[\lambda])$ and $\alpha' = (1, \cdots, 1, 2, \alpha[j+1], \cdots, \alpha[\lambda])$. For $d \in [0, j]$ define α_d as $\alpha_d[\ell] = \alpha'[\ell]$ for $\ell \in [d]$ and $\alpha_d[\ell] = \alpha[\ell]$ at every other index. Therefore $\alpha_0 = \alpha$ (correspond to $(k, t-1, t)$) and $\alpha_j = \alpha'$ (corresponds to $(k, t, t+1)$). Note that every α_{d-1} and α_d differ only on index d. There is an adversary $\mathcal{A}$ s.t. for some $d \in [j]$ it holds that:

$$\Pr[\mathsf{Bad} \ \wedge \ \mathsf{pcf}^{(0)}_{r} \neq \mathsf{pcf}^{(1)}_{r} \ : \ \exp_{\alpha_d,\beta'}] - \Pr[\mathsf{Bad} \ \wedge \ \mathsf{pcf}^{(0)}_{r} \neq \mathsf{pcf}^{(1)}_{r} \ : \ \exp_{\alpha_{d-1},\beta'}] \geq \epsilon_3(\lambda)/\lambda \quad (11)$$

Now we construct $\mathcal{B}^{\mathsf{ih}}_{\mathsf{seh}}$ s.t. $\mathcal{B}^{\mathsf{ih}}_{\mathsf{seh}}(\alpha_d, \beta', d, \mathcal{A})$ breaks the index-hiding property of SEH on two indices $e_1 = \alpha_d[d]$ and $e_2 = \alpha_{d-1}[d]$.

$\mathcal{B}^{\mathsf{ih}}_{\mathsf{seh}}(\alpha, \beta, \ell, \mathcal{A})$ This PPT attacker does the following:

1. Receives (n, S, T) from $\mathcal{A}$ and hk^* from the challenger where $(\mathsf{hk}^*, \mathsf{td}^*) \leftarrow \mathsf{seBARG.Gen}(1^\lambda, 2, \Sigma^{\ell-1}_{\mathsf{cf}}, e_b)$.
2. Computes $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Gen}_{\alpha,\beta}(1^\lambda, n, S, T)$ and updates crs by replacing $\mathsf{hk}_{\mathsf{cf},\ell}$ with hk^*.
3. Sends crs to $\mathcal{A}$ and receives $(\mathsf{hz}, (\mathsf{out}^{(0)}, \Pi^{(0)}), (\mathsf{out}^{(1)}, \Pi^{(1)}), 1^{N \cdot T})$.

4. If for any $b \in \{0,1\}$, $\mathsf{Verify}(\mathsf{crs}, (\mathsf{hz}, \mathsf{out}^{(b)}), \Pi^{(b)}) = 0$ or $\mathsf{out}^{(0)} = \mathsf{out}^{(1)}$ then abort.
5. Finds $\pi_\ell^{(b)} = (\mathsf{v}^{(b)}, \rho_1^{(b)}, \rho_2^{(b)}, \mathsf{h}^{(b)}, \hat{\pi}^{(b)}) \in \Pi^{(b)}$.
6. Compute $\mathsf{epcf}_r^{(b)} \leftarrow \mathsf{Extract}_r(\mathsf{td}, \Pi^{(b)})$ and $\mathsf{pcf}_r^{(b)} = \mathsf{PKE.Dec}(\mathsf{sk}, \mathsf{epcf}_r^{(b)})$ for $b \in \{0,1\}$.
7. Output 1 if $\mathsf{pcf}_r^{(0)} \neq \mathsf{pcf}_r^{(1)}$ and 2 otherwise.

In the reduction above $b = 1$ corresponds to $\exp_{\alpha,\beta_d}$ and $b = 2$ corresponds to $\exp_{\alpha,\beta_{d-1}}$. Therefore, $\mathcal{B}^{\mathsf{ih}}_{\mathsf{seh}}(\alpha_d, \beta', d, \mathcal{A})$'s distinguishing advantage is greater than that of $\mathcal{A}$ in Eq. (11) contradicting the SEH security.

Case 4. Similar to case 1, for some $b \in \{0,1\}$ it holds that:

$$\epsilon_4/2 \leq \Pr[\mathsf{Bad} \ \wedge \ \mathsf{pcf}_{k,t}^{(b)} \neq \mathsf{pcf}_r^{(b)} \ : \ \exp_{\alpha',\beta'}]$$

Note that α' and β' are the same path, thus the proof is similar to case one.

□

Claim A.6. If SEH is somewhere binding w.r.t. path opening, HT has reading soundness, PKE is correct, and α and β correspond to $(K, T-1, T)$, and for any stateful PPT adversary $\mathcal{A}$ it holds that:

$$\Pr[\mathsf{Bad} \ \wedge \ \mathsf{pcf}_{K,T}^{(0)} \neq \mathsf{pcf}_{K,T}^{(1)} \ : \ \exp_{\alpha,\beta}] \leq \mathsf{negl}(\lambda),$$

then for any such adversary, it holds that:

$$\Pr[\mathsf{Bad} \ : \ \exp_{\alpha,\beta}] \leq \mathsf{negl}(\lambda).$$

Proof. It suffices to show that for any PPT adversary $\mathcal{A}$ it holds that:

$$\delta_0 = \Pr[\mathsf{Bad} \ \wedge \ \mathsf{pcf}_{K,T}^{(0)} = \mathsf{pcf}_{K,T}^{(1)} \ : \ \exp_{\alpha,\beta}] \leq \mathsf{negl}(\lambda). \tag{12}$$

Let $\pi_\ell^{(b)} = (\mathsf{v}^{(b)}, \rho_1^{(b)}, \rho_2^{(b)}, \mathsf{h}^{(b)}, \hat{\pi}^{(b)}) \in \Pi^{(b)}$ be the proof that contains $(\mathsf{pcf}_{K,T-1}^{(b)}, \mathsf{pcf}_{K,T}^{(b)})$ and For $b \in \{0,1\}$ parse $\rho_2^{(b)} = ((\mathsf{v}_{0,1}^{(b)}, \mathsf{v}_{0,2}^{(b)}), \ldots, (\mathsf{v}_{\ell-1,1}^{(b)}, \mathsf{v}_{\ell-1,2}^{(b)}))$, and $\mathsf{v}_{0,2}^{(b)} = (\mathsf{epcf}_1^{(b)}, \mathsf{epcf}_2^{(b)})$, and $\pi_{\mathsf{out}}^{(b)} = (\mathsf{pcf}_2^{(b)}, r^{(b)}, \hat{\pi}_{\mathsf{out}}^{(b)})$, and $\mathsf{pcf}_2^{(b)} = (\mathsf{rt}_2^{(b)}, \mathsf{q}_2^{(b)}, \mathsf{ridx}_2^{(b)})$. Now, define the following probabilities:

$$\delta_1 = \Pr\left[\begin{array}{ll} \mathsf{HT.VfyRd}(\mathsf{hk}_{\mathsf{ht}}, \mathsf{rt}_2^{(b)}, [n_{\mathsf{out}}], \mathsf{out}^{(b)}, \hat{\pi}_{\mathsf{out}}^{(b)}) = 1 \text{ for } b \in \{0,1\} \ \wedge & \\ \mathsf{PKE.Enc}(\mathsf{pk}, \mathsf{pcf}_2^{(b)}; r^{(b)}) = \mathsf{epcf}_2^{(b)} \text{ for } b \in \{0,1\} \ \wedge & \\ \mathsf{SEH.VerifyAcc}((\mathsf{hk}_{\mathsf{cf},d})_{d\in[\ell]}, \mathsf{v}^{(b)}, \rho_2^{(b)}, 2) \text{ for } b \in \{0,1\} \ \wedge & : \ \exp_{\alpha,\beta} \\ \mathsf{out}^{(0)} \neq \mathsf{out}^{(1)} \ \wedge \ \mathsf{pcf}_{K,T}^{(0)} = \mathsf{pcf}_{K,T}^{(1)} & \end{array}\right].$$

$$\delta_2 = \Pr\left[\begin{array}{ll} \mathsf{HT.VfyRd}(\mathsf{hk}_{\mathsf{ht}}, \mathsf{rt}_2^{(b)}, [n_{\mathsf{out}}], \mathsf{out}^{(b)}, \hat{\pi}_{\mathsf{out}}^{(b)}) = 1 \text{ for } b \in \{0,1\} \ \wedge & \\ \mathsf{PKE.Enc}(\mathsf{pk}, \mathsf{pcf}_2^{(b)}; r^{(b)}) = \mathsf{epcf}_2^{(b)} \text{ for } b \in \{0,1\} \ \wedge & \\ \mathsf{epcf}_{K,T}^{(0)} = \mathsf{epcf}_2^{(0)} \ \wedge & : \ \exp_{\alpha,\beta} \\ \mathsf{SEH.VerifyAcc}((\mathsf{hk}_{\mathsf{cf},d})_{d\in[\ell]}, \mathsf{v}^{(1)}, \rho_2^{(1)}, 2) \ \wedge & \\ \mathsf{out}^{(0)} \neq \mathsf{out}^{(1)} \ \wedge \ \mathsf{pcf}_{K,T}^{(0)} = \mathsf{pcf}_{K,T}^{(1)} & \end{array}\right].$$

$$\delta_3 = \Pr\left[\begin{array}{ll} \mathsf{HT.VfyRd}(\mathsf{hk_{ht}}, \mathsf{rt}_2^{(b)}, [n_{\mathsf{out}}], \mathsf{out}^{(b)}, \hat{\pi}_{\mathsf{out}}^{(b)}) = 1 \text{ for } b \in \{0,1\} \wedge & \\ \mathsf{PKE.Enc}(\mathsf{pk}, \mathsf{pcf}_2^{(b)}; r^{(b)}) = \mathsf{epcf}_2^{(b)} \text{ for } b \in \{0,1\} \wedge & : \quad \exp_{\alpha,\beta} \\ \mathsf{epcf}_{K,T}^{(b)} = \mathsf{epcf}_2^{(b)} \text{ for } b \in \{0,1\} \wedge & \\ \mathsf{out}^{(0)} \neq \mathsf{out}^{(1)} \ \wedge \ \mathsf{pcf}_{K,T}^{(0)} = \mathsf{pcf}_{K,T}^{(1)} & \end{array}\right].$$

$$\delta_4 = \Pr\left[\begin{array}{ll} \mathsf{HT.VfyRd}(\mathsf{hk_{ht}}, \mathsf{rt}_2^{(b)}, [n_{\mathsf{out}}], \mathsf{out}^{(b)}, \hat{\pi}_{\mathsf{out}}^{(b)}) = 1 \text{ for } b \in \{0,1\} \wedge & \\ \mathsf{pcf}_{K,T}^{(b)} = \mathsf{pcf}_2^{(b)} \text{ for } b \in \{0,1\} \wedge & : \quad \exp_{\alpha,\beta} \\ \mathsf{out}^{(0)} \neq \mathsf{out}^{(1)} \ \wedge \ \mathsf{pcf}_{K,T}^{(0)} = \mathsf{pcf}_{K,T}^{(1)} & \end{array}\right].$$

First note that $\delta_0 \leq \delta_1$ since all the events are implied by the verification of the proofs. Also, $\delta_3 = \delta_4$ by the perfect correctness of PKE. Moreover, the advantage of an adversary $\mathcal{B}$ attacking the reading soundness of HT by outputting $(\mathsf{rt}_2^{(0)}, [n_{\mathsf{out}}], \mathsf{out}^{(0)}, \pi_{\mathsf{out}}^{(0)}, \mathsf{out}^{(1)}, \pi_{\mathsf{out}}^{(1)})$ when $\mathcal{A}$ generates $((\mathsf{out}^{(0)}, \Pi^{(0)}), (\mathsf{out}^{(1)}, \Pi^{(1)}))$ is at least δ_4, thus $\delta_4 \leq \mathsf{negl}(\lambda)$. Therefore, we only need to show that $\delta_{b+1} - \delta_{b+2} \leq \mathsf{negl}(\lambda)$ for $b \in \{0,1\}$. It holds that:

$$\delta_{b+1} - \delta_{b+2} \leq \epsilon_b = \Pr\left[\ \mathsf{SEH.VerifyAcc}((\mathsf{hk}_{\mathsf{cf},d})_{d\in[\ell]}, \mathsf{v}^{(b)}, \rho_2^{(b)}, 2) \ \wedge \ \mathsf{pcf}_{K,T}^{(b)} = \mathsf{pcf}_2^{(b)} \quad : \quad \exp_{\alpha,\beta}\ \right].$$

Now by somewhere binding w.r.t. path opening of SEH it holds that $\epsilon_b \leq \mathsf{negl}(\lambda)$ concluding the proof.

□

□

Zero-Knowledge. We construct the simulator $\mathcal{S}(\mathsf{crs}, \mathsf{cf})$ as follows:

- It generates crs similar to Gen except that it runs the NIZK simulator to generate $\mathsf{crs_{zk}}$.
- To generate the proof it does the following:
  1. Compute $\mathsf{epcf}_{k,t}$ for $k \in [K], t \in [T]$ as follows: (1) let $\mathsf{epcf}_{1,0} = \mathsf{PKE.Enc}(\mathsf{pk}, \mathsf{pcf}_{1,0}; 0^{|r|})$, (2) let $\mathsf{cf} = (\mathsf{W}, \mathrm{q}, \mathsf{ridx})$ and compute $(\mathsf{rt}, \mathsf{tree}) = \mathsf{HT.Hash}(\mathsf{hk_{ht}}, \mathsf{W})$, and $\mathsf{pcf}_{K,T} = (\mathsf{rt}, \mathrm{q}, \mathsf{ridx})$, then sample randomness r and compute $\mathsf{epcf}_{K,T} = \mathsf{PKE.Enc}(\mathsf{pk}, \mathsf{pcf}_{K,T})$, (3) for every other (k,t) independently (under different randomness) compute $\mathsf{epcf}_{k,t} = \mathsf{PKE.Enc}(\mathsf{pk}, 0^{|\mathsf{pcf}|})$.
  2. Let $\mathsf{st} = (\mathsf{cf}, \mathsf{pcf}_{K,T}, r)$.
  3. Generate $\mathsf{ct}_{\mathsf{in},k}$ for $k \in [K]$ by independently encrypting 0^λ.
  4. To generate hz use the same iterative process as in Digest except that in Item 2a instead of computing $\mathsf{ct_{in}}$ use $\mathsf{ct}_{\mathsf{in},k}$ computed above.
  5. To generate the proof use the same iterative process as in Update except that do not compute $(\mathsf{pcf}_{t-1}, \mathsf{pcf}_t, \mathsf{rbit}, \mathsf{rop}, \mathsf{wop})$, and to generate π_{zk} use NIZK simulator. Also, never compute π_{out} except when $(k,t) = (K,T)$ in which case compute π_{out} similar to the prover.

We define the following hybrids.

$\mathcal{H}_0$: This is the original experiment. Namely, the adversary sends (n, S, T), challenger sends $\mathsf{crs}_0 \leftarrow \mathsf{Gen}(1^\lambda, n, S, T)$, adversary sends $\overline{z}$., challenger computes $\mathsf{cf} = \mathcal{M}(\overline{z}; 1^T)$ and sends $(\mathsf{st}_0, \mathsf{hz}_0, \Pi_0) \leftarrow \mathsf{Prove}(\mathsf{crs}_0, \overline{z})$ to the adversary and finally adversary outputs b'.

$\mathcal{H}_1$: Similar to the original experiment except that NIZK simulator is used to generate $\mathsf{crs}_{\mathsf{zk}}$, and π_{zk} at every iteration.

$\mathcal{H}_{2,k,t}$: Similar to $\mathcal{H}_1$ except that for every $\mathsf{cf}_{i,j}$ until $\mathsf{cf}_{k,t}$ except $\mathsf{cf}_{1,0}$ and $\mathsf{cf}_{K,T}$, we let $\mathsf{epcf}_{i,j} = \mathsf{PKE.Enc}(\mathsf{pk}, 0^{|\mathsf{pcf}|})$ (and for the rest $\mathsf{epcf}_{i,j}$ is honestly generated). Note that $\mathcal{H}_{2,1,0} = \mathcal{H}_1$ and $\mathcal{H}_{2,K,T-1} = \mathcal{H}_{2,K,T}$.

$\mathcal{H}_{3,k}$: Similar to $\mathcal{H}_{2,K,T}$ except that for every z_i until z_k, we let $\mathsf{ct}_{\mathsf{in},i} = \mathsf{PKE.Enc}(\mathsf{pk}, 0^\lambda)$ (and for the rest $\mathsf{ct}_{\mathsf{in},i}$ is honestly generated). Note that $\mathcal{H}_{3,0} = \mathcal{H}_{2,K,T}$ and $\mathcal{H}_{3,K}$ is the output of the simulator. Namely in $\mathcal{H}_{3,K}$ the adversary sends (n, S, T), the simulator sends $\mathsf{crs}_1 \leftarrow \mathcal{S}(1^\lambda, n, S, T)$, adversary sends $\overline{z}$, simulator computes $\mathsf{cf} = \mathcal{M}(\overline{z}; 1^T)$ and sends $(\mathsf{st}_1, \mathsf{hz}_1, \Pi_1) \leftarrow \mathcal{S}(\mathsf{crs}_0, \mathsf{cf})$ to the adversary and finally adversary outputs b'.

For any adversary let $\mathcal{H}^*(\mathcal{A})$ be the output of the adversary in hybrid $\mathcal{H}^*$.

Claim A.7. If NIZK is zero-knowledge then for any stateful PPT adversary $\mathcal{A}$ it holds that:

$$|\Pr[\mathcal{H}_1(\mathcal{A}) = 1] - \Pr[\mathcal{H}_0(\mathcal{A}) = 1]| \leq \mathsf{negl}(\lambda).$$

Proof. Towards the contradictions suppose there exists an adversary $\mathcal{A}$ and a non-negligible function $\epsilon(\cdot)$ s.t. $|\Pr[\mathcal{H}_1(\mathcal{A}) = 1] - \Pr[\mathcal{H}_0(\mathcal{A}) = 1]| \geq \epsilon(\lambda)$. We construct adversary $\mathcal{B}_{\mathsf{zk}}$ that breaks the zero-knowledge property of the underlying NIZK protocol as follows:

1. Receives (n, S, T) from $\mathcal{A}$, and crs^* from NIZK zero-knowledge challenger.
2. Computes $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Gen}(1^\lambda, n, S, T)$ and updates crs by replacing $\mathsf{crs}_{\mathsf{zk}}$ with crs^*.
3. Sends crs to $\mathcal{A}$, receives $\overline{z}$.
4. Computes $(\mathsf{st}, \mathsf{hz}, \Pi)$ similar to the prover except that at every single step, instead of computing π_{zk}, queries the challenger on (x, w) and receive π_{zk}.
5. Outputs whatever $\mathcal{A}$ outputs.

If the NIZK challenger generates crs and proof using Gen and Prove, then $\mathcal{B}_{\mathsf{zk}}$ simulates $\mathcal{H}_0$ for $\mathcal{A}$ and if the challenger uses NIZK simulator, then $\mathcal{B}_{\mathsf{zk}}$ simulates $\mathcal{H}_1$ for $\mathcal{A}$. Thus $\mathcal{B}_{\mathsf{zk}}$ has a non-negligible advantage $\epsilon(\lambda)$ in breaking the zero-knowledge property of NIZK. □

In the following claim we let $k \in [K]$, $t \in [T]$, and $(k, t) \neq (K, T)$.

Claim A.8. If PKE is secure then for any stateful PPT adversary $\mathcal{A}$ it holds that:

$$|\Pr[\mathcal{H}_{2,k,t}(\mathcal{A}) = 1] - \Pr[\mathcal{H}_{2,k,t-1}(\mathcal{A}) = 1]| \leq \mathsf{negl}(\lambda).$$

Proof. Towards the contradictions suppose there exists an adversary $\mathcal{A}$ and a non-negligible function $\epsilon(\cdot)$ s.t. $|\Pr[\mathcal{H}_{2,k,t}(\mathcal{A})=1]-\Pr[\mathcal{H}_{2,k,t-1}(\mathcal{A})=1]| \geq \epsilon(\lambda)$. We construct adversary $\mathcal{B}_{\mathsf{pke}}$ that breaks the security of the underlying PKE protocol as follows:

1. Receives (n, S, T) from $\mathcal{A}$, and pk^* from PKE challenger.
2. Computes $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Gen}(1^\lambda, n, S, T)$ and updates crs by replacing pk with pk^*.
3. Sends crs to $\mathcal{A}$, receives $\overline{z}$.
4. Computes $(\mathsf{st}, \mathsf{hz}, \Pi)$ similar to $\mathcal{H}_{2,k,t-1}$ except that instead of computing $\mathsf{epcf}_{k,t}$, queries the challenger on $\mathsf{pcf}_{k,t}$ and receive $\mathsf{epcf}_{k,t}$.
5. Outputs whatever $\mathcal{A}$ outputs.

If the PKE challenger generates the ciphertext honestly, then $\mathcal{B}_{\mathsf{pke}}$ simulates $\mathcal{H}_{2,k,t-1}$ for $\mathcal{A}$ and if it responds with a ciphertext of all zeros, then $\mathcal{B}_{\mathsf{pke}}$ simulates $\mathcal{H}_{2,k,t}$ for $\mathcal{A}$. Thus $\mathcal{B}_{\mathsf{pke}}$ has a non-negligible advantage $\epsilon(\lambda)$ in breaking the security of PKE. □

In the following claim we let $k \in [K]$.

Claim A.9. If PKE is secure then for any stateful PPT adversary $\mathcal{A}$ it holds that:

$$|\Pr[\mathcal{H}_{3,k}(\mathcal{A})=1]-\Pr[\mathcal{H}_{3,k-1}(\mathcal{A})=1]| \leq \mathsf{negl}(\lambda).$$

Proof. Similar to the proof of Claim A.8. □

Therefore we conclude the proof of zero-knowledge property by a hybrid argument.

B Predicated zk-IVsC Complete Analysis.

Proof of Theorem 3.11. The completeness, and strong soundness proofs are similar to $\mathsf{zk\text{-}IVsC}$. We only need to prove efficiency, predicate proof soundness, and zero-knowledge properties of our construction.

Efficiency. The crs size and Setup running time in addition to the other elements (which is similar to IVsC) grows with $\mathsf{crs}_{\mathsf{zk},f}$ size and generation time which grow with $\mathsf{poly}(\lambda, n_{\mathsf{zk},f,x}, n_{\mathsf{zk},f,w}) = \mathsf{poly}(\lambda, n, n_f)$, thus in total grows with $\mathsf{poly}(\lambda, \log S, n, n_f)$.

Now note the size of $\pi_{f,\log T-1}$ and its generation/verification time grows with NIZK statement and witness and the the circuit that decides the language, thus all three grow with $\mathsf{poly}(\lambda, n, n_f)$. Additionally, similar to the IVsC analysis the proof size and its generation/verification (given the tree decomposition of $\overline{f}_k$) grows with $\mathsf{poly}(\lambda, n, n_f, \log K)$. Finally tree decomposition of $\overline{f}_k$ grows at most with $\mathsf{poly}(K)$.

Predicate Proof Soundness. First note that the crs indistinguishability of Gen and TGen folows direcrtly from the crs indistinguishablity of the seBARG, and crs indistinguishability of NIZK extractor. Next we construct algorithms TGen, $\mathsf{HashCheck}$ and the extractor $\mathcal{E}$.

The $\mathsf{TGen}(1^\lambda, n, S, T, n_f, k^*) \to \mathsf{crs}$ algorithm is the same as Gen except that γ corresponds to k^*, and NIZK extractor $\mathcal{E}$ is used to generate $\mathsf{crs}_{\mathsf{zk},f}$. It computes $\mathsf{td} = ((\mathsf{crs}_{\mathsf{bp},f,\ell})_{\ell\in[\log T,\lambda]}, \mathsf{td}_{\mathsf{zk},f}, k^*)$.

The $\mathsf{HashCheck}(\mathsf{crs}, \mathsf{hz}, z^*, k^*, \mathsf{op}) \to 0/1$ algorithm does the following:

1. Let $\mathsf{h}_j \in \mathsf{hz}$ be a hash that includes $k^{*\text{th}}$ leaf and $(b_{j-1}, \ldots, b_{\log T-1})$ (where $b_i \in \{0,1\}$) be the path from h_j to the $k^{*\text{th}}$ leaf. Then parse opening as $\mathsf{op} = (r, y_{\log T-1,1}, y_{\log T-1,2}, \ldots, y_{j-1,1}, y_{j-1,2})$.
2. For $i \in [\log T, j-1]$ check $\mathsf{CRHF.Hash}(\mathsf{hk}_{\mathsf{ch}}, (y_{i-1,1}, y_{i-1,2})) = y_{i,b_i}$.
3. If $j \geq \log T$ check $\mathsf{CRHF.Hash}(\mathsf{hk}_{\mathsf{ch}}, (y_{j-1,1}, y_{j-1,2})) = \mathsf{h}_j$ otherwise let $y_{\log T-1,b_{\log T-1}} = \mathsf{h}_j$.
4. Compute $(\mathsf{rt}, \mathsf{tree}) = \mathsf{HT.Hash}(\mathsf{hk}_{\mathsf{ht}}, z^*)$ and check $y_{\log T-1,b_{\log T-1}} = \mathsf{PKE.Enc}(\mathsf{pk}, \mathsf{rt}; r)$.
5. Output 1 if all the checks pass and 0 otherwise.

The extractor $\mathcal{E}$ does the following:

1. Find a proof $\pi_{f,\ell}$ s.t. it contains a proof for z_{k^*} and let $\pi^*_{f,\ell} = \pi_{f,\ell}$.
2. Let $\mathsf{hz}_{k-1} = \{\mathsf{h}_{\log T-1}, \cdots, \mathsf{h}_\lambda\}$ and $\mathsf{h}^*_{\ell,\gamma[\ell+1]} = \mathsf{h}_\ell$.
3. Compute $\overline{f}_{\mathsf{dc},\ell-\log T+1}$ using $\mathsf{TreeDecompose}(\overline{f})$ and let $\overline{f}^*_{\ell,\gamma[\ell+1]} = \overline{f}_{\mathsf{dc},\ell-\log T+1}$.
4. Recursively extract from the proof $\pi_{f,\ell}$. Namely for $i \in [\log T, \ell]$ Compute $(\mathsf{h}^*_{i-1,1}, \mathsf{h}^*_{i-1,2}, \pi^*_{f,i-1}) = \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_{\mathsf{bp},f,i}, (\overline{f}^*_{i,\gamma[i+1]} = (\overline{f}^*_{i-1,1}, \overline{f}^*_{i-1,2}), \mathsf{h}^*_{i,\gamma[i+1]}, j)_{j\in[2]}, \pi^*_{f,i})$.
5. Compute $(z^*, \mathsf{h}^*, r^*) = \mathsf{NIZK}.\mathcal{E}(\mathsf{td}_{\mathsf{zk},f}, (\overline{f}^*_{\log T-1,\gamma[\log T]}, \mathsf{h}^*_{\log T-1,\gamma[\log T]}), \pi^*_{f,\log T-1})$.
6. Compute $\mathsf{op} = (r^*, \mathsf{h}^*_{\log T-1,1}, \mathsf{h}^*_{\log T-1,2}, \ldots, \mathsf{h}^*_{\ell-1,1}, \mathsf{h}^*_{\ell-1,2})$ and output (z^*, op).

Additionally let $\mathsf{Extract}$ be an algorithm that performs only one extraction (either BARG or NIZK depending on ℓ) from proof $\pi_{f,\ell}$.

For any stateful PPT adversary $\mathcal{A}$ and any $\lambda \in \mathbb{N}$ define the experiment $\exp_{\alpha,\beta}$ (where α corresponds to $(k, t-1, t)$) as follows:

$$\exp := \left[\begin{array}{l} n, S, T, n_f, k^* \leftarrow \mathcal{A}_1(1^\lambda), \\ (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{TGen}(1^\lambda, n, S, T, n_f, k^*), \\ (\overline{f}, \mathsf{hz}, \Sigma) \leftarrow \mathcal{A}_2(\mathsf{crs}), \\ (\overline{f}_{\mathsf{dc},0}, \ldots, \overline{f}_{\mathsf{dc},\lambda}) = \mathsf{TreeDecompose}(\overline{f}) \\ (z^*, \mathsf{op}) \leftarrow \mathcal{E}(\mathsf{td}, \overline{f}, \mathsf{hz}, \Sigma) \\ w \leftarrow \mathsf{Extract}(\mathsf{td}, \overline{f}, \mathsf{hz}, \Sigma) \end{array}\right]$$

Moreover, let Bad be the following event:

$$\mathsf{Bad} := \left[\begin{array}{l} \mathsf{PredicateVerify}(\mathsf{crs}, \overline{f}, \mathsf{hz}, \Sigma) = 1 \ \wedge \\ (f_{k^*}(z^*) = 0 \ \vee \ \mathsf{HashCheck}(\mathsf{crs}, \mathsf{hz}, z^*, k^*, \mathsf{op}) = 0) \end{array}\right].$$

To prove the claim, we need to show that for any stateful PPT adversary $\mathcal{A}$ it holds that:

$$\delta_0 = \Pr[\mathsf{Bad} \ : \ \mathrm{exp}] \leq \mathsf{negl}(\lambda).$$

We prove this by induction on the depth ℓ of the proof $\pi_{f,\ell}$ that contains a proof for z_{k^*}. First note that by the construction of the $\mathcal{E}$ it holds that for any $\mathsf{h}_\ell \in \mathsf{hz}$ that includes k^*th leaf, $(\gamma_\ell, \ldots, \gamma_{\log T})$ is a path from h_ℓ to k^*th leaf. Thus we just need to check the CRHF hash correctness on the path to k^*th leaf and the encryption + hash correctness on the base level. Given $\mathsf{op} = (r^*, \mathsf{h}^*_{\log T-1,1}, \mathsf{h}^*_{\log T-1,2}, \ldots, \mathsf{h}^*_{\ell-1,1}, \mathsf{h}^*_{\ell-1,2})$ Let $\mathsf{op}_j = (r^*, \mathsf{h}^*_{\log T-1,1}, \mathsf{h}^*_{\log T-1,2}, \ldots, \mathsf{h}^*_{j-1,1}, \mathsf{h}^*_{j-1,2})$ and $\mathsf{HashCheck}'(\mathsf{crs}, \mathsf{h}_j, z^*, \mathsf{op}_j)$ partially run $\mathsf{HashCheck}$ as follows: 1) check the CRHF hash correctness in the op_j on path $(\gamma_j, \ldots, \gamma_{\log T})$ from root h_j, 2) check the base level encryption + hash correctness.

Define Ver_ℓ be the following event:

- If $\ell \geq \log T$ then

$$\mathsf{Ver}_\ell \coloneqq \left[\begin{array}{l} \mathsf{seBARG.Verify}(\mathsf{crs}_{\mathsf{bp},f,\ell}, (\overline{f}^*_{\ell,\gamma[\ell+1]}, \mathsf{h}_\ell, j)_{j\in[2]}, \pi_{f,\ell}) = 1 \ \wedge \\ (f_{k^*}(z^*) = 0 \ \vee \ \mathsf{HashCheck}'(\mathsf{crs}, \mathsf{h}_\ell, z^*, \mathsf{op}_\ell) = 0 \end{array} \right].$$

- Otherwise

$$\mathsf{Ver}_\ell \coloneqq \left[\begin{array}{l} \mathsf{NIZK.Verify}(\mathsf{crs}_{\mathsf{zk},f}, (\overline{f}^*_{\log T-1,\gamma[\log T]}, \mathsf{h}_\ell), \pi_{f,\ell}) = 1 \ \wedge \\ (f_{k^*}(z^*) = 0 \ \vee \mathsf{HashCheck}'(\mathsf{crs}, \mathsf{h}_\ell, z^*, \mathsf{op}_\ell) = 0) \end{array} \right].$$

and define $\delta_{1,\ell} = \Pr[\mathsf{Ver}_\ell \ : \ \mathrm{exp}]$. Note that $\delta_{1,\ell} \geq \delta_0$ for all ℓ, thus it suffices to prove that $\delta_{1,\ell} \leq \mathsf{negl}(\lambda)$.

Induction Base. Suppose $\ell = \log T - 1$, thus it holds that $\mathcal{E}(\mathsf{td}, \overline{f}, \mathsf{hz}, \Sigma) = (z^*, \mathsf{op} = r^*)$ and $w = \mathsf{Extract}(\mathsf{td}, \overline{f}, \mathsf{hz}, \Sigma) = \mathsf{NIZK}.\mathcal{E}(\mathsf{td}_{\mathsf{zk},f}, x = (\overline{f}^*_{\log T-1,\gamma[\log T]}, \mathsf{h}_\ell), \pi_{f,\ell}) = (z^*, \mathsf{h}^*, r^*)$. Now we define the following probability:

$$\delta_2 = \Pr\left[\ (x,w) \in \mathcal{L}_{\mathsf{zk},f} \ \wedge \ (f_{k^*}(z^*) = 0 \ \vee \ \mathsf{HashCheck}'(\mathsf{crs}, \mathsf{h}_\ell, z^*, \mathsf{op}_\ell) = 0) \ \ : \ \mathrm{exp} \right].$$

Note that:

$$\delta_{1,\ell} - \delta_2 \leq \Pr[\mathsf{NIZK.Verify}(\mathsf{crs}_{\mathsf{zk},f}, x, \pi_{f,\ell}) = 1 \ \wedge \ (x,w) \notin \mathcal{L}_{\mathsf{zk},f} \ : \ \mathrm{exp}] \leq \mathsf{negl}(\lambda)$$

where the last inequality holds by the argument of knowledge property of the NIZK. Additionally note that $\delta_2 = 0$ as $(x,w) \in \mathcal{L}_{\mathsf{zk},f}$ implies (a) $f_{k^*}(z^*) = 0$, and (b) $(\mathsf{h}^*, \mathsf{tree}^*) = \mathsf{HT.Hash}(\mathsf{hk}_{\mathsf{ht}}, z^*)$ and $\mathsf{PKE.Enc}(\mathsf{pk}, \mathsf{h}^*; r^*) = \mathsf{h}_\ell$, thus $\mathsf{HashCheck}'(\mathsf{crs}, \mathsf{h}_\ell, z^*, \mathsf{op}_\ell) = 1$. So it holds that $\delta_{1,\log T-1} \leq \mathsf{negl}(\lambda)$.

Induction Step. Suppose $\delta_{1,i} \leq \mathsf{negl}(\lambda)$ for all $i < \ell$, and we want to prove that $\delta_{1,\ell} \leq \mathsf{negl}(\lambda)$. Then it holds that $w = \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_{\mathsf{bp},f,\ell}, (\overline{f}_{\ell,\gamma[\ell+1]} = (\overline{f}^*_1, \overline{f}^*_2), \mathsf{h}_\ell, j)_{j\in[2]}, \pi_{f,\ell}) = (\mathsf{h}^*_1, \mathsf{h}^*_2, \pi^*_f)$.

$\vee \ \mathsf{HashCheck}'(\mathsf{crs}, \mathsf{h}^*_{\ell-1,\gamma[\ell]}, z^*, \mathsf{op}_{\ell-1}) = 0$

Now for $b = \gamma[\ell]$ we define the following probability:

$$\delta_2 = \Pr[(x = (\overline{f}^*_b, \mathsf{h}^*_b, b), w) \in \mathcal{L}_{\mathsf{bp},f,\ell} \ \wedge \ (f_{k^*}(z^*) = 0 \ \vee \ \mathsf{HashCheck}'(\mathsf{crs}, \mathsf{h}_\ell, z^*, \mathsf{op}_\ell) = 0) \ : \ \mathrm{exp}].$$

Note that:

$$\delta_{1,\ell}-\delta_2 \leq \Pr[\mathsf{seBARG.Verify}(\mathsf{crs}_{\mathsf{bp},f,\ell}, ((\overline{f}^*_1, \overline{f}^*_2), \mathsf{h}_\ell, j)_{j\in[2]}, \pi_{f,\ell}) = 1 \wedge (x,w) \notin \mathcal{L}_{\mathsf{bp},f,\ell} \,:\, \mathrm{exp}] \leq \mathsf{negl}(\lambda)$$

where the last inequality holds by the knowledge extractor property of the seBARG. Additionally $(x,w) \in \mathcal{L}_{\mathsf{bp},f,\ell}$ implies that 1) $\mathsf{CRHF.Hash}(\mathsf{hk}_{\mathsf{ch}}, (\mathsf{h}^*_1, \mathsf{h}^*_2)) = \mathsf{h}_\ell$, 2) if $\ell = \log T$ then $\mathsf{NIZK.Verify}(\mathsf{crs}_{\mathsf{zk},f}, (\overline{f}^*_b, \mathsf{h}^*_b), \pi^*_f) = 1$, and if $\ell > \log T$ then $\mathsf{seBARG.Verify}(\mathsf{crs}_{\mathsf{bp},f,\ell-1}, (\overline{f}^*_b, \mathsf{h}^*_b, j)_{j\in[2]}, \pi^*_f) = 1$.

Now since $\mathsf{HashCheck}'(\mathsf{crs}, \mathsf{h}_\ell, z^*, \mathsf{op}_\ell) = 0$ is equivalent with

$$\mathsf{CRHF.Hash}(\mathsf{hk}_{\mathsf{ch}}, (\mathsf{h}^*_1, \mathsf{h}^*_2)) \neq \mathsf{h}_\ell \vee \mathsf{HashCheck}'(\mathsf{crs}, \mathsf{h}^*_b, z^*, \mathsf{op}_{\ell-1}) = 0,$$

it holds that $\delta_2 \leq \delta_{1,\ell-1} \leq \mathsf{negl}(\lambda)$ where the last inequality holds by the induction assumption. Thus it holds that $\delta_{1,\ell} \leq \mathsf{negl}(\lambda)$ concluding the proof.

Zero-Knowledge. We construct the simulator $\mathcal{S}(\mathsf{crs}, \mathsf{cf})$ as follows:

- It generates crs similar to Gen except that it runs the NIZK simulators to generate $\mathsf{crs}_{\mathsf{zk}}$ and $\mathsf{crs}_{\mathsf{zk},f}$.
- To generate the proof it does the following:
  1. Compute $\mathsf{cf}_{k,t}$, st, $\mathsf{ct}_{\mathsf{in},k}$, $\overline{\mathsf{hz}}$, and Π similar to the zero-knowledge simulator for $\mathsf{zk\text{-}IVsC}$ except that for $\overline{\mathsf{hz}}$ output all the intermediate hz values (i.e., hz_k for $k \in [K]$) instead of the final hz.
  2. Compute Σ similar to the predicate prover except that use NIZK simulator to compute $\pi_{\mathsf{zk},f}$ where we use the previously computed $\mathsf{ct}_{\mathsf{in},k}$ in the statement.

We define the following hybrids (note that the hybrids are similar to the proof of zero-knowledge property of $\mathsf{zk\text{-}IVsC}$ except that we additionally have a $\mathcal{H}_2$ to simulate the predicate proofs).

$\mathcal{H}_0$: This is the original experiment. Namely, the adversary sends (n, S, T, n_f), challenger sends $\mathsf{crs}_0 \leftarrow \mathsf{Gen}(1^\lambda, n, S, T, n_f)$, adversary sends $(\overline{z}, \overline{f})$ s.t. $f_k(z_k) = 1$. Then Challenger computes $\mathsf{cf} = \mathcal{M}(\overline{z}; 1^T)$, $(\mathsf{st}_0, \overline{\mathsf{hz}}_0, \overline{\mathsf{ps}}, \Pi_0) \leftarrow \mathsf{Prove}(\mathsf{crs}_0, \overline{z})$, $(\mathsf{hz}_{0,K}, \Sigma_0) \leftarrow \mathsf{PredicateProve}(\mathsf{crs}, \overline{f}, \overline{z}, \overline{\mathsf{ps}})$ and sends $(\mathsf{st}_0, \overline{\mathsf{hz}}_0, \Pi_0, \Sigma_0)$ to the adversary and finally adversary outputs b'.

$\mathcal{H}_1$: Similar to the original experiment except that NIZK simulator is used to generate $\mathsf{crs}_{\mathsf{zk}}$, and π_{zk} at every iteration of computing Π.

$\mathcal{H}_2$: Similar to $\mathcal{H}_1$ except that NIZK simulator is used to generate $\mathsf{crs}_{\mathsf{zk},f}$, and $\pi_{\mathsf{zk},f}$ at every iteration of computing Σ.

$\mathcal{H}_{3,k,t}$: Similar to $\mathcal{H}_2$ except that for every $\mathsf{cf}_{i,j}$ until $\mathsf{cf}_{k,t}$ except $\mathsf{cf}_{1,0}$ and $\mathsf{cf}_{K,T}$, we let $\mathsf{epcf}_{i,j} = \mathsf{PKE.Enc}(\mathsf{pk}, 0^{|\mathsf{pcf}|})$ (and for the rest $\mathsf{epcf}_{i,j}$ is honestly generated). Note that $\mathcal{H}_{3,1,0} = \mathcal{H}_2$ and $\mathcal{H}_{3,K,T-1} = \mathcal{H}_{3,K,T}$.

$\mathcal{H}_{4,k}$: Similar to $\mathcal{H}_{3,K,T}$ except that for every z_i until z_k, we let $\mathsf{ct}_{\mathsf{in},i} = \mathsf{PKE.Enc}(\mathsf{pk}, 0^\lambda)$ (and for the rest $\mathsf{ct}_{\mathsf{in},i}$ is honestly generated). Note that $\mathcal{H}_{4,0} = \mathcal{H}_{3,K,T}$ and $\mathcal{H}_{4,K}$ is the output of the simulator. Namely in $\mathcal{H}_{4,K}$ the adversary sends (n, S, T, n_f), the simulator sends $\mathsf{crs}_1 \leftarrow \mathcal{S}(1^\lambda, n, S, T, n_f)$, adversary sends $\overline{z}$ $(\overline{z}, \overline{f})$ s.t. $f_k(z_k) = 1$. The simulator computes $\mathsf{cf} = \mathcal{M}(\overline{z}; 1^T)$ and sends $(\mathsf{st}_1, \overline{\mathsf{hz}}_1, \Pi_1, \Sigma_1) \leftarrow \mathcal{S}(\mathsf{crs}_0, \mathsf{cf})$ to the adversary and finally adversary outputs b'.

For any adversary let $\mathcal{H}^*(\mathcal{A})$ be the output of the adversary in hybrid $\mathcal{H}^*$. The proof closely follows the proof of zero-knowledge property in zk-IVsC except that we additionally have to prove the indistinguishability of hybrids $\mathcal{H}_1$ and $\mathcal{H}_2$. Thus we have the following claim:

Claim B.1. If NIZK is zero-knowledge then for any stateful PPT adversary $\mathcal{A}$ it holds that:

$$|\Pr[\mathcal{H}_2(\mathcal{A})=1]-\Pr[\mathcal{H}_1(\mathcal{A})=1]| \leq \mathsf{negl}(\lambda).$$

Proof. Similar to the proof of Claim A.7. □

□

C ICZK Complete Analysis.

Proof of Theorem 4.4. Here we prove completeness, adaptive soundness and incremental zero-knowledge of our construction.

Completeness. The completeness follows by the construction and the completeness of the underlying Ψ, and PKE.

Efficiency. Note that the size of hz_k values and $\Pi_{\psi,k}$ are $\mathsf{poly}(\lambda, \log|x_1|, \log|w_1|, \log K)$, and the size of $\Sigma_{\psi,k}$ is $\mathsf{poly}(\lambda, \log|x_1|, |w_1|, \log K)$. Additionally since rrPKE is rate-1, the size of each ct_k is $|w_i| + \mathsf{poly}(\lambda)$. Therefore, the total proof size is $K|w_i| + K \cdot \mathsf{poly}(\lambda, \log|x_1|, \log|w_1|, \log K) + \mathsf{poly}(\lambda, \log|x_1|, |w_1|, \log K)$.

Adaptive Soundness. Let $\mathsf{Exp}_{\mathsf{ICZK}}$ be the following experiment:

$$\mathsf{Exp}_{\mathsf{ICZK}} \coloneqq \left[\begin{array}{l} (\mathsf{crs},\mathsf{sk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^{n_x}, 1^{n_w}), \\ (\overline{x}, \Pi = (\overline{\mathsf{hz}}, \Pi_\psi, \Sigma_\psi, \overline{\mathsf{ct}})) \leftarrow \mathcal{A}(\mathsf{crs}) \\ |x_k| = n_x, ((w_k, \mathsf{ps}_k; r_k) \leftarrow \mathsf{PKE.Dec}(\mathsf{sk}, \mathsf{ct}_k))_{k\in[K]} \\ (z_k = x_k, w_k, \mathsf{ps}_k, r_k)_{k\in[K]} \end{array}\right]$$

and Bad be the following event:

$$\mathsf{Bad} \coloneqq \left[\begin{array}{l} \overline{x} \notin \mathcal{L} \ \wedge \ \Psi.\mathsf{Verify}(\mathsf{crs}, (\mathsf{hz}_K, 1), \Pi_\psi) = 1 \ \wedge \\ \Psi.\mathsf{PredicateVerify}(\mathsf{crs}, \overline{f}, \mathsf{hz}_K, \Sigma_\psi) = 1 \end{array}\right].$$

Note that to prove the adaptive soundness of Construction 4.3 we only need to prove that $\delta_1 = \Pr[\mathsf{Bad} \ : \ \mathsf{Exp}_{\mathsf{ICZK}}] \leq \mathsf{negl}(\lambda)$. Now define the following probabilities:

$$\delta_2 = \Pr\left[\begin{array}{ll} \overline{x} \notin \mathcal{L} \ \wedge \ \Psi.\mathsf{Verify}(\mathsf{crs}, (\mathsf{hz}_K, 1), \Pi_\psi) = 1 \ \wedge & : \ \mathsf{Exp}_{\mathsf{ICZK}} \\ \mathsf{hz}_K = \Psi.\mathsf{Digest}(\mathsf{crs}, \overline{z}, \overline{\mathsf{ps}}) & \end{array}\right].$$

$$\delta_3 = \Pr\left[\ \overline{x} \notin \mathcal{L} \ \wedge \ \Psi.\mathsf{Verify}(\mathsf{crs}, (\mathsf{hz}_K, 1), \Pi_\psi) = 1 \quad : \quad \mathsf{Exp}_{\mathsf{ICZK}}, \mathsf{hz}_K = \Psi.\mathsf{Digest}(\mathsf{crs}, \overline{z}, \overline{\mathsf{ps}}) \ \right].$$

Now note that $\overline{x} \notin \mathcal{L}$ implies $\mathcal{M}(\overline{x}, \overline{w}) = 0$ that implies $\mathcal{M}'(\overline{z}) = 0$ thus by the soundness property that we prove in predicated zk-IVsC it holds that $\delta_3 \leq \mathsf{negl}(\lambda)$. Additionally note that

$\delta_2 \leq \delta_3$. Hence, it suffices to show that $\delta_1 - \delta_2 \leq \mathsf{negl}(\lambda)$, namely we need to show that $\delta^* \leq \mathsf{negl}(\lambda)$ where:

$$\delta^* = \Pr\left[\begin{array}{ll} \Psi.\mathsf{PredicateVerify}(\mathsf{crs}, \overline{f}, \mathsf{hz}_K, \Sigma_\psi) = 1 \wedge & : \quad \mathsf{Exp}_{\mathsf{ICZK}} \\ \mathsf{hz}_K \neq \Psi.\mathsf{Digest}(\mathsf{crs}, \overline{z}, \overline{f}) & \end{array}\right].$$

Suppose for some non-negligible function $\epsilon(\cdot)$ it holds that $\delta^* \geq \epsilon(\lambda)$, then for some $k^* \in [K]$ and some non-negligible function $\epsilon'(\cdot)$ it holds that:

$$\Pr\left[\begin{array}{ll} \Psi.\mathsf{PredicateVerify}(\mathsf{crs}, \overline{f}, \mathsf{hz}_K, \Sigma_\psi) = 1 \wedge & \\ \mathsf{hz}_{k^*} \neq \Psi.\mathsf{Digest}(\mathsf{crs}, \overline{z}_{k^*}, \overline{\mathsf{ps}}_{k^*}) \wedge & : \quad \mathsf{Exp}_{\mathsf{ICZK}} \\ \mathsf{hz}_{k^*-1} = \Psi.\mathsf{Digest}(\mathsf{crs}, \overline{z}_{k^*-1}, \overline{\mathsf{ps}}_{k^*-1}) & \end{array}\right] \geq \epsilon'(\lambda). \tag{13}$$

Now we construct an adversary $\mathcal{B}$ that given $\mathcal{A}$ breaks the predicate soundness property that we proved in predicated zk-IVsC. $\mathcal{B}$ receives n_x and n_w from $\mathcal{A}$, samples a random index $k' \in [K]$, sends (n, S, T, n_f, k') to the challenger, receives crs_ψ from the challenger, samples $(\mathsf{pk}, \mathsf{sk})$, send $\mathsf{crs} = (\mathsf{crs}_\psi, \mathsf{pk})$ to $\mathcal{A}$, receives $(\overline{x}, \Pi)$ from $\mathcal{A}$, computes $(\overline{w}, \overline{\mathsf{ps}}, \overline{r})$ by decrypting $\overline{\mathsf{ct}}$ using sk, lets $z_k = (x_k, w_k, \mathsf{ps}_k, r_k)$ for $k \in [K]$, finds pk' and $\mathsf{hk}_{\mathsf{ht}}$ in crs_ψ, computes $\overline{f}$, and sends $(\overline{f}, \mathsf{hz}_K, \Sigma_\psi)$ to the challenger. Note that this is the same as $\mathsf{Exp}_{\mathsf{ICZK}}$ except that crs_ψ is generated using $\Psi.\mathsf{TGen}$ on index k'. Let this experiment be $\mathsf{Exp}_{\mathsf{ICZK},k'}$ Note that $\mathcal{B}$'s advantage can be computed as follows:

$$\delta_{\mathcal{B}} = \sum_{k'=1}^{K} 1/K \cdot \Pr\left[\begin{array}{ll} \Psi.\mathsf{PredicateVerify}(\mathsf{crs}, \overline{f}, \mathsf{hz}_K, \Sigma_\psi) = 1 \wedge & : \quad \mathsf{Exp}_{\mathsf{ICZK},k'}, \\ f_{k'}(z^*) = 0 & \quad z^* \leftarrow \Psi.\mathcal{E}(\mathsf{td}, \overline{f}, \mathsf{hz}_K, \Sigma_\psi) \end{array}\right],$$

and it holds that,

$$\delta_{\mathcal{B}} \geq 1/K \cdot \Pr\left[\begin{array}{ll} \Psi.\mathsf{PredicateVerify}(\mathsf{crs}, \overline{f}, \mathsf{hz}_K, \Sigma_\psi) = 1 \wedge & : \quad \mathsf{Exp}_{\mathsf{ICZK},k^*}, \\ f_{k^*}(z^*) = 0 & \quad z^* \leftarrow \Psi.\mathcal{E}(\mathsf{td}, \overline{f}, \mathsf{hz}_K, \Sigma_\psi) \end{array}\right]. \tag{14}$$

Now let $z^* = (x^*, w^*, \mathsf{ps}^*, r^*)$, and consider the following cases:

- If $x^* \neq x_{k^*}$ then by the the description of f_{k^*} it holds that $f_{k^*}(z^*) = 0$.
- If $\mathsf{PKE.Enc}(\mathsf{pk}, (w^*, \mathsf{ps}^*); r^*) \neq \mathsf{ct}_k$ then by the the description of f_{k^*} it holds that $f_{k^*}(z^*) = 0$ (otherwise by perfect correctness of PKE it holds that $(w^*, \mathsf{ps}^*, r^*) = (w_{k^*}, \mathsf{ps}_{k^*}, r_{k^*})$).
- If $z^* = z_{k^*}$, then $f_{k^*}(z^*) = 0$ is equivalent with $f_{k^*}(z_{k^*}) = 0$.

Let $(\mathsf{rt}_{k^*}, \mathsf{tree}_{k^*}) = \mathsf{HT.Hash}(\mathsf{hk}_{\mathsf{ht}}, (x_{k^*}, w_{k^*}, \mathsf{ps}_{k^*}, r_{k^*}))$ and $\hat{\mathsf{ct}}_{k^*} = \mathsf{PKE.Enc}(\mathsf{pk}', \mathsf{rt}_{k^*}; \mathsf{ps}_{k^*})$. Thus by Eq. (14) and the description of f_{k^*} it holds that:

$$\delta_{\mathcal{B}} \geq \Pr\left[\begin{array}{ll} \Psi.\mathsf{PredicateVerify}(\mathsf{crs}, \overline{f}, \mathsf{hz}_K, \Pi_\psi) = 1 \wedge & : \quad \mathsf{Exp}_{\mathsf{ICZK},k^*} \\ \mathsf{nxt\text{-}hz}(\mathsf{hz}_{k-1}, \hat{\mathsf{ct}}_{k^*}) \neq \mathsf{hz}_k & \end{array}\right] \geq \epsilon(\lambda)/K - \mathsf{negl}(\lambda).$$

which means

$$\delta_{\mathcal{B}} \geq \Pr\left[\begin{array}{ll} \Psi.\mathsf{PredicateVerify}(\mathsf{crs}, \overline{f}, \mathsf{hz}_K, \Pi_\psi) = 1 \wedge & \\ \mathsf{hz}_{k^*} \neq \Psi.\mathsf{Digest}(\mathsf{crs}, \overline{z}_{k^*}, \overline{\mathsf{ps}}_{k^*}) \wedge & : \quad \mathsf{Exp}_{\mathsf{ICZK},k^*} \\ \mathsf{hz}_{k^*-1} = \Psi.\mathsf{Digest}(\mathsf{crs}, \overline{z}_{k^*-1}, \overline{\mathsf{ps}}_{k^*-1}) & \end{array}\right] \geq \epsilon(\lambda)/K - \mathsf{negl}(\lambda).$$

there by Eq. (13) it holds that $\delta_{\mathcal{B}} \geq \epsilon'(\lambda)$ which contradicts the predicate soundness property that we proved for predicated zk-IVsC concluding the soundness of Construction 4.3.

Incremental zero-Knowledge. We construct the simulator $\mathcal{S}(\mathsf{crs}, \mathsf{cf})$ as follows:

- It generates crs similar to the zero-knowledge simulator in the predicated zk-IVsC.
- To generate the proof it does the following:
  1. Compute Π_ψ, Σ_ψ, and $\overline{\mathsf{hz}}$ similar to the zero-knowledge simulator in the predicated zk-IVsC.
  2. Compute $\overline{\mathsf{ct}}$ by independently encrypting $0^{n_w+|\mathsf{ps}|}$ for every $k \in [K]$.

We define the following hybrids:

$\mathcal{H}_0$: This is the original experiment. Namely, the challenger on input (n_x, n_w), sends $\mathsf{crs}_0 \leftarrow \mathsf{Gen}(1^\lambda, n_x, n_w)$ to $\mathcal{A}$, $\mathcal{A}$ sends $(\overline{x}, \overline{w})$ s.t. $\mathcal{R}(\overline{x}, \overline{w}) = 1$ and $\mathcal{M}(\overline{x}, \overline{w}) = \mathsf{cf}$. Then the challenger sends $(\mathsf{st}_0, \Pi_0) = \mathsf{Prove}(\mathsf{crs}, \overline{x}, \overline{w})$ to $\mathcal{A}$, and finally $\mathcal{A}$ outputs b'.

$\mathcal{H}_1$: Similar to $\mathcal{H}_0$ except that NIZK simulator is used to generate $\mathsf{crs}_{\mathsf{zk}}$, and π_{zk} at every iteration of computing Π_ψ.

This step only uses the zero-knowledge property of the underlying NIZK. Note that after this step the only information about the intermediate configurations is in $\mathsf{epcf}_{k,t}$ (for $k \in [K]$ and $t \in [T]$) that is the encryption of the root of the pseudo-configuration. Additionally, the only information about the inputs of the machine is in $(\overline{\mathsf{hz}}, \overline{\mathsf{ct}}, \Sigma_\psi)$ and $(\mathsf{ct}_{\mathsf{in},k})_{k\in[K]}$ (that is being used in the statements of Π_ψ).

$\mathcal{H}_2$: Similar to $\mathcal{H}_1$ except that NIZK simulator is used to generate $\mathsf{crs}_{\mathsf{zk},f}$, and $\pi_{\mathsf{zk},f}$ at every iteration of computing Σ_ψ.

This step only uses the zero-knowledge property of the underlying NIZK. Note that after this step there is no information about inputs $(\overline{w}, \overline{\mathsf{ps}}, \overline{r})$ in the proof except the ones in $(\overline{\mathsf{hz}}, \overline{\mathsf{ct}})$ and $(\mathsf{ct}_{\mathsf{in},k})_{k\in[K]}$ (that is being used in the statements of Π_ψ and Σ_ψ). Additionally note that the only information about the input z_k other than the ones in ct_k, is a short (of size λ bits) digest of the input z_k that is encrypted in $\mathsf{ct}_{\mathsf{in},k}$ (since $\overline{\mathsf{hz}}$ is constructed only using $(\mathsf{ct}_{\mathsf{in},k})_{k\in[K]}$).

$\mathcal{H}_{3,k,t}$: Similar to $\mathcal{H}_2$ except that for every $\mathsf{cf}_{i,j}$ until $\mathsf{cf}_{k,t}$ except $\mathsf{cf}_{1,0}$ and $\mathsf{cf}_{K,T}$, and in the generation of Π_ψ we let $\mathsf{epcf}_{i,j} \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}, 0^{|\mathsf{pcf}|})$ (and for the rest $\mathsf{epcf}_{i,j}$ is honestly generated). Note that $\mathcal{H}_{3,1,0} = \mathcal{H}_2$ and $\mathcal{H}_{3,K,T-1} = \mathcal{H}_{3,K,T}$.

This step only uses the security of the underlying PKE. Note that after this step there is no information about the intermediate configurations in the proof.

$\mathcal{H}_{4,k}$: Similar to $\mathcal{H}_{3,K,T}$ except that for every ct_i until ct_k, we independently sample r_i and let $\mathsf{ct}_i = \mathsf{rrPKE.Enc}(\mathsf{pk}, 0^{n_w+|\mathsf{ps}|}; r_i)$ (and for the rest ct_i is honestly generated). Note that $\mathcal{H}_{4,0} = \mathcal{H}_{3,K,T}$.

This step only uses the security with leakage of the underlying rrPKE as the only information about randomness $\overline{r}$ remained in the proof is in $(\mathsf{ct}_{\mathsf{in},k})_{k\in[K]}$ (that is the encryption of a short (of size λ) bits of the inputs). Note that after this step there's no information about $\overline{w}$ and $\overline{\mathsf{ps}}$ in the $\overline{\mathsf{ct}}$. The only information left in the proof is in $(\mathsf{ct}_{\mathsf{in},k})_{k\in[K]}$.

$\mathcal{H}_{5,k}$: Similar to $\mathcal{H}_{4,K}$ except that for every z_i until z_k, we let $\mathsf{ct}_{\mathsf{in},i} \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}, 0^\lambda)$ (and for the rest $\mathsf{ct}_{\mathsf{in},i}$ is honestly generated). Note that $\mathcal{H}_{5,0} = \mathcal{H}_{4,K}$ and $\mathcal{H}_{5,K}$ is the output of the simulator. Namely in $\mathcal{H}_{5,K}$ the simulator on input (n_x, n_w), sends $\mathsf{crs}_1 \leftarrow \mathcal{S}(1^\lambda, n_x, n_w)$ to $\mathcal{A}$, $\mathcal{A}$ sends $(\overline{x}, \overline{w})$ s.t. $\mathcal{R}(\overline{x}, \overline{w}) = 1$ and $\mathcal{M}(\overline{x}, \overline{w}) = \mathsf{cf}$. Then the simulator sends $(\mathsf{st}_1, \Pi_1) = \mathcal{S}(\mathsf{crs}, \overline{x}, \mathsf{cf})$ to $\mathcal{A}$, and finally $\mathcal{A}$ outputs b'.

This step only uses the randomness-dependent message security of the underlying PKE as the only information about randomness $\overline{\mathsf{ps}}$ remained in the proof is in the message of $(\mathsf{ct}_{\mathsf{in},k})_{k\in[K]}$ (that is the encryption of a short (of size λ) bits of the inputs). Note that after this step there's no information about $\overline{w}$ in the proof and the proof can be simulated using only $(\overline{x}, \mathsf{cf})$.

For any adversary let $\mathcal{H}^*(\mathcal{A})$ be the output of the adversary in hybrid $\mathcal{H}^*$. We have the following claims:

Claim C.1. If NIZK is zero-knowledge then for any stateful PPT adversary $\mathcal{A}$ it holds that:

$$|\Pr[\mathcal{H}_1(\mathcal{A}) = 1] - \Pr[\mathcal{H}_0(\mathcal{A}) = 1]| \leq \mathsf{negl}(\lambda).$$

Proof. hi Towards the contradictions suppose there exists an adversary $\mathcal{A}$ and a non-negligible function $\epsilon(\cdot)$ s.t. $|\Pr[\mathcal{H}_1(\mathcal{A}) = 1] - \Pr[\mathcal{H}_0(\mathcal{A}) = 1]| \geq \epsilon(\lambda)$. We construct adversary $\mathcal{B}_{\mathsf{zk}}$ that breaks the zero-knowledge property of the underlying NIZK protocol as follows:

1. Receives $(1^{n_x}, 1^{n_w})$ as input, and crs^* from NIZK zero-knowledge challenger.
2. Computes $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Gen}(1^\lambda, 1^{n_x}, 1^{n_w})$ and updates crs by replacing $\mathsf{crs}_{\mathsf{zk}} \in \mathsf{crs}_\psi$ with crs^*.
3. Sends crs to $\mathcal{A}$, receives $\overline{x}, \overline{w}$.
4. Computes (st, Π) similar to the prover except that at every step of updating hz_{k-1} to $\mathsf{hz}_k - 1$ Π_ψ (every call of $\mathsf{UpdateSingleStep}$ in subroutine $\Psi.\mathsf{Update}$), instead of computing π_{zk}, queries the challenger on the correct computation of corresponding (x, w) and receive π_{zk}.
5. Outputs whatever $\mathcal{A}$ outputs.

If the NIZK challenger generates crs and proof using Gen and Prove, then $\mathcal{B}_{\mathsf{zk}}$ simulates $\mathcal{H}_0$ for $\mathcal{A}$ and if the challenger uses NIZK simulator, then $\mathcal{B}_{\mathsf{zk}}$ simulates $\mathcal{H}_1$ for $\mathcal{A}$. Thus $\mathcal{B}_{\mathsf{zk}}$ has a non-negligible advantage $\epsilon(\lambda)$ in breaking the zero-knowledge property of NIZK. □

Claim C.2. If NIZK is zero-knowledge then for any stateful PPT adversary $\mathcal{A}$ it holds that:

$$|\Pr[\mathcal{H}_2(\mathcal{A}) = 1] - \Pr[\mathcal{H}_1(\mathcal{A}) = 1]| \leq \mathsf{negl}(\lambda).$$

Proof. Towards the contradictions suppose there exists an adversary $\mathcal{A}$ and a non-negligible function $\epsilon(\cdot)$ s.t. $|\Pr[\mathcal{H}_2(\mathcal{A}) = 1] - \Pr[\mathcal{H}_1(\mathcal{A}) = 1]| \geq \epsilon(\lambda)$. We construct adversary $\mathcal{B}_{\mathsf{zk}}$ that breaks the zero-knowledge property of the underlying NIZK protocol as follows:

1. Receives $(1^{n_x}, 1^{n_w})$ as input, and crs^* from NIZK zero-knowledge challenger.
2. Computes $(\mathsf{crs}, \mathsf{td})$ similar to $\mathcal{H}_1$ and updates crs by replacing $\mathsf{crs}_{\mathsf{zk},f} \in \mathsf{crs}_\psi$ with crs^*.
3. Sends crs to $\mathcal{A}$, receives $\overline{x}, \overline{w}$.
4. Computes (st, Π) similar to the $\mathcal{H}_1$ except that at every update of computing Σ_ψ (every call of subroutine $\Psi.\mathsf{PredicateUpdate}$), instead of computing $\pi_{\mathsf{zk},f}$, queries the challenger on the correct computation of the corresponding (x, w) and receive $\pi_{\mathsf{zk},f}$.
5. Outputs whatever $\mathcal{A}$ outputs.

If the NIZK challenger generates crs and proof using Gen and Prove, then $\mathcal{B}_{\mathsf{zk}}$ simulates $\mathcal{H}_1$ for $\mathcal{A}$ and if the challenger uses NIZK simulator, then $\mathcal{B}_{\mathsf{zk}}$ simulates $\mathcal{H}_2$ for $\mathcal{A}$. Thus $\mathcal{B}_{\mathsf{zk}}$ has a non-negligible advantage $\epsilon(\lambda)$ in breaking the zero-knowledge property of NIZK. □

In the following claim we let $k \in [K]$, $t \in [T]$, and $(k, t) \neq (K, T)$.

Claim C.3. If PKE is secure then for any stateful PPT adversary $\mathcal{A}$ it holds that:

$$|\Pr[\mathcal{H}_{3,k,t}(\mathcal{A}) = 1] - \Pr[\mathcal{H}_{3,k,t-1}(\mathcal{A}) = 1]| \leq \mathsf{negl}(\lambda).$$

Proof. Towards the contradictions suppose there exists an adversary $\mathcal{A}$ and a non-negligible function $\epsilon(\cdot)$ s.t. $|\Pr[\mathcal{H}_{3,k,t}(\mathcal{A}) = 1] - \Pr[\mathcal{H}_{3,k,t-1}(\mathcal{A}) = 1]| \geq \epsilon(\lambda)$. We construct adversary $\mathcal{B}_{\mathsf{pke}}$ that breaks the security of the underlying PKE protocol as follows:

1. Receives $(1^{n_x}, 1^{n_w})$ as input, and pk^* from PKE challenger.
2. Computes $(\mathsf{crs}, \mathsf{td})$ similar to $\mathcal{H}_2$ and updates crs by replacing $\mathsf{pk} \in \mathsf{crs}_\psi$ with pk^*.
3. Sends crs to $\mathcal{A}$, receives $(\overline{x}, \overline{w})$.
4. Computes (st, Π) similar to $\mathcal{H}_2$ except that when running single step of computing Π_ψ (calling $\mathsf{UpdateSingleStep}$ in subroutine $\Psi.\mathsf{Update}$) on (k, t), instead of computing $\mathsf{epcf}_{k,t}$, queries the challenger on $\mathsf{pcf}_{k,t}$ and receive $\mathsf{epcf}_{k,t}$.
5. Outputs whatever $\mathcal{A}$ outputs.

If the PKE challenger generates the ciphertext honestly, then $\mathcal{B}_{\mathsf{pke}}$ simulates $\mathcal{H}_{3,k,t-1}$ for $\mathcal{A}$ and if it responds with a ciphertext of all zeros, then $\mathcal{B}_{\mathsf{pke}}$ simulates $\mathcal{H}_{3,k,t}$ for $\mathcal{A}$. Thus $\mathcal{B}_{\mathsf{pke}}$ has a non-negligible advantage $\epsilon(\lambda)$ in breaking the security of PKE. □

In the following claim we let $k \in [K]$.

Claim C.4. If rrPKE is secure then for any stateful PPT adversary $\mathcal{A}$ it holds that:

$$|\Pr[\mathcal{H}_{4,k}(\mathcal{A}) = 1] - \Pr[\mathcal{H}_{4,k-1}(\mathcal{A}) = 1]| \leq \mathsf{negl}(\lambda).$$

Proof. Towards the contradictions suppose there exists an adversary $\mathcal{A}$ and a non-negligible function $\epsilon(\cdot)$ s.t. $|\Pr[\mathcal{H}_{4,k}(\mathcal{A})=1]-\Pr[\mathcal{H}_{4,k-1}(\mathcal{A})=1]| \geq \epsilon(\lambda)$. We construct adversary $\mathcal{B}_{\mathsf{pke}}$ that breaks the security with leakage of the underlying PKE protocol as follows:

1. Receives $(1^{n_x}, 1^{n_w})$ as input, and pk^* from PKE challenger.
2. Computes $(\mathsf{crs}, \mathsf{td})$ similar to $\mathcal{H}_{3,K,T}$ and updates crs by replacing pk with pk^*.
3. Sends crs to $\mathcal{A}$, receives $(\overline{x}, \overline{w})$.
4. Computes (st, Π) similar to $\mathcal{H}_{3,K,T}$ except that instead of computing $\mathsf{ct}_k \in \overline{\mathsf{ct}}$, queries the challenger on $((w_k, \mathsf{ps}_k), g_k(\cdot))$, and receive $(\mathsf{ct}_k, g(r_k))$, where we define $g_k(r)$ to compute $\mathsf{HT.Hash}(\mathsf{hk}_{\mathsf{ht}}, z_k = (x_k, w_k, \mathsf{ps}_k, r_k))$ and output only rt_k (and not tree_k). Note that $g_k(\cdot)$ has $(\mathsf{hk}_{\mathsf{ht}}, x_k, w_k, \mathsf{ps}_k)$ hard-coded.
5. Outputs whatever $\mathcal{A}$ outputs.

If the PKE challenger generates the ciphertext honestly, then $\mathcal{B}_{\mathsf{pke}}$ simulates $\mathcal{H}_{4,k-1}$ for $\mathcal{A}$ and if it responds with a ciphertext of all zeros, then $\mathcal{B}_{\mathsf{pke}}$ simulates $\mathcal{H}_{4,k}$ for $\mathcal{A}$. Additionally note that $|\mathsf{rt}_k| = \lambda$, thus there is only λ bits of leakage about the randomness r_k that is used to encrypt (w_k, ps_k). Therefore $\mathcal{B}_{\mathsf{pke}}$ has a non-negligible advantage $\epsilon(\lambda)$ in breaking the security with leakage of PKE. □

In the following claim we let $k \in [K]$.

Claim C.5. If PKE is secure then for any stateful PPT adversary $\mathcal{A}$ it holds that:

$$|\Pr[\mathcal{H}_{5,k}(\mathcal{A})=1]-\Pr[\mathcal{H}_{5,k-1}(\mathcal{A})=1]| \leq \mathsf{negl}(\lambda).$$

Proof. Towards the contradictions suppose there exists an adversary $\mathcal{A}$ and a non-negligible function $\epsilon(\cdot)$ s.t. $|\Pr[\mathcal{H}_{5,k}(\mathcal{A})=1]-\Pr[\mathcal{H}_{5,k-1}(\mathcal{A})=1]| \geq \epsilon(\lambda)$. We construct adversary $\mathcal{B}_{\mathsf{pke}}$ that breaks the randomness-dependent message security of the underlying PKE protocol as follows:

1. Receives $(1^{n_x}, 1^{n_w})$ as input, and pk^* from PKE challenger.
2. Computes $(\mathsf{crs}, \mathsf{td})$ similar to $\mathcal{H}_{4,K}$ and updates crs by replacing $\mathsf{pk}' \in \mathsf{crs}_\psi$ with pk^*.
3. Sends crs to $\mathcal{A}$, receives $(\overline{x}, \overline{w})$.
4. Computes (st, Π) similar to $\mathcal{H}_{4,K,T}$ except that instead of computing $\mathsf{ct}_{\mathsf{in},k}$, interacts with the challenger as follows: sends $g_k(\cdot)$, receives $g_k(\mathsf{ps}_k)$, sends $g_k(\mathsf{ps}_k)$, and receive $\mathsf{ct}_{\mathsf{in},k}$. We define $g_k(\mathsf{ps}_k)$ to compute $\mathsf{HT.Hash}(\mathsf{hk}_{\mathsf{ht}}, z_k = (x_k, w_k, \mathsf{ps}_k, r_k))$ and output only rt_k (and not tree_k). Note that $g_k(\cdot)$ has $(\mathsf{hk}_{\mathsf{ht}}, x_k, w_k, r_k)$ hard-coded.
5. Outputs whatever $\mathcal{A}$ outputs.

If the PKE challenger generates the ciphertext honestly, then $\mathcal{B}_{\mathsf{pke}}$ simulates $\mathcal{H}_{5,k-1}$ for $\mathcal{A}$ and if it responds with a ciphertext of all zeros, then $\mathcal{B}_{\mathsf{pke}}$ simulates $\mathcal{H}_{5,k}$ for $\mathcal{A}$. Additionally note that $|\mathsf{rt}_k| = \lambda$, thus there is only λ bits of leakage about the randomness ps_k that is used to encrypt rt_k. Therefore $\mathcal{B}_{\mathsf{pke}}$ has a non-negligible advantage $\epsilon(\lambda)$ in breaking the randomness-dependent message security PKE. □

Therefore we conclude the proof of zero-knowledge property by a hybrid argument.

□