Homomorphic NIZKs and Maliciously-Secure MPC with Optimal Communication Complexity from Standard Assumptions

Abtin Afshar
UW–Madison
abtin@cs.wisc.edu

Jiaqi Cheng
UW–Madison
jiaqicheng@cs.wisc.edu

Rishab Goyal
UW–Madison
rishab@cs.wisc.edu*

Aayush Yadav
Rutgers University–Camden
aayush.yadav@rutgers.edu†

Saikumar Yadugiri
UW–Madison
saikumar@cs.wisc.edu

Abstract

We construct fully-homomorphic NIZKs and a communication-complexity-preserving semi-malicious to malicious multiparty computation (MPC) compiler from a broader range of cryptographic assumptions, including LWE, $O(1)$-LIN, and DDH (without circular security assumptions). Prior to this work, homomorphic NIZKs were only known based on pairing-based techniques while prior MPC compilers with similar efficiency required strong non-falsifiable assumptions. A key ingredient that we develop in this work is a new notion of a partially-hiding succinct delegation system, that we call encrypted RAM (eRAM) delegation. In addition to homomorphic NIZKs and optimal MPC compilers, eRAM delegation is also useful for designing composable zero-knowledge batch arguments and multi-hop homomorphic signatures with strong-context-hiding.

*Support for this research was provided by OVCRGE at UW-Madison with funding from the Wisconsin Alumni Research Foundation.

†Work done while at UW–Madison. Funded by NSF #2143287 and #1942575.

Contents

1 Introduction **1**
- 1.1 Concurrent Works . . . 2

2 Technical Overview **3**
- 2.1 Homomorphic NIZKs . . . 3
- 2.2 Introducing encrypted RAM delegation . . . 6
- 2.3 Complexity-Preserving Semi-Malicious to Malicious MPC . . . 8
- 2.4 Other Applications . . . 10
- 2.5 Related Work . . . 11

3 Preliminaries **12**
- 3.1 RAM Programs . . . 12
- 3.2 Somewhere Extractable (Zero-Knowledge) Batch Arguments . . . 13
- 3.3 Hash Tree . . . 14
- 3.4 Non-Interactive Zero-Knowledge Arguments (NIZK) . . . 15
- 3.5 Rate-1 Message Encoding . . . 17
- 3.6 Public-Key Encryption System . . . 17
- 3.7 RAM Delegation Scheme . . . 18

4 Defining (Reusable) Encrypted RAM Delegation **19**
- 4.1 Encrypted RAM Delegation . . . 19
- 4.2 Reusable Encrypted RAM Delegation . . . 20

5 Constructing (Reusable) Encrypted RAM Delegation **22**
- 5.1 Encrypted RAM Delegation . . . 23
  - 5.1.1 Security Analysis. . . . 24
- 5.2 Reusable Encrypted RAM Delegation . . . 29
  - 5.2.1 Security Analysis. . . . 30

6 Homomorphic Non-Interactive Zero-Knowledge **33**
- 6.1 Definitions: Homomorphic NIZKs . . . 33
- 6.2 Tools Needed: Optimally Efficient NIZK and Rate-1 Message Encodings . . . 34
  - 6.2.1 Security Analysis . . . 36
- 6.3 Boolean Circuit Decomposition . . . 38
- 6.4 Constructing Homomorphic NIZKs . . . 39
- 6.5 Security Analysis . . . 41

7 Complexity-Preserving Semi-Malicious to Malicious Compiler for mrNISC **43**
- 7.1 Definition . . . 44
- 7.2 Construction . . . 45
- 7.3 Security Analysis of Construction 7.4 . . . 47
- 7.4 Semi-Maliciously Secure mrNISC with Compact Communication . . . 52

8 Multi-Hop zkBARG with Somewhere-Extraction **55**
8.1 Definition . . . 56
8.2 Construction . . . 58
8.3 Security Analysis . . . 59

9 Publicly-Verifiable Succinct Delegation of Committed Programs **63**
9.1 Definition . . . 64
9.2 Construction . . . 66

A Additional Definitions **74**
A.1 Somewhere Extractable Hash Families . . . 74
A.2 Multi-Key Fully Homomorphic Encryption . . . 75

B Split Memory RAM Delegation **76**
B.1 Split Memory RAM Delegation Construction . . . 76
B.2 Security Analysis . . . 79

1 Introduction

Non-interactive zero-knowledge (NIZK) [GMR85, BFM88, BDSMP91] and secure multiparty computation (MPC) [Yao82, Yao86, GMW87] protocols are two foundational pillars of modern cryptography. While the feasibility of these primitives from standard cryptographic assumptions is well established, constructing them with advanced properties, specifically *homomorphism* for NIZKs [CKLM12, CKLM13b, ADKL19] and *optimal communication* for malicious MPC [Gen09, BGI16], is far less understood.

Homomorphism for NIZKs is particularly useful when statements and their proofs evolve through a sequence of transformations. For example, under the C2PA standard as described in VerITAS [DCB25], cameras sign photos at capture, and VerITAS uses zk-SNARKs to prove that a published image is a permissible edit of a signed original without revealing it. In such a system, each proof must currently be recomputed from the original, even though editorial pipelines may span many editors. Homomorphic NIZKs instead allow each editor to derive a proof directly from the incoming proof, while unlinkability can hide all intermediate images. Similar needs arise in aggregating proofs over committed data [ADKL19] and in re-randomizing proofs for reverse firewalls and rerandomizable encryption [CHL+25].

State of the art. Despite these applications, the only known approach to constructing homomorphic NIZKs remains narrowly tied to pairing-based techniques [CKLM12, CKLM13a, CKLM13b, AN11, AGP14, ACJ17, AGM18, ADKL19]. This leaves homomorphic NIZKs as the only primitive in the broader family of homomorphic cryptographic primitives [RAD+78, Gen09, JMSW02, ABC+07, AB09, BFKW09, GVW15b, BGI15, BGI16, DHRW16] without a quantum-safe construction.

In the same vein, to the best of our knowledge, the only approach to designing a maliciously-secure MPC with optimal communication complexity (two rounds of interaction, and total number of bits sent by each party is asymptotically equal to the input plus output length) relies on strong non-falsifiable assumptions [DGMR22].

This work. We resolve these open problems by introducing a new core primitive: *Encrypted RAM (eRAM) Delegation*. We show that eRAM is a versatile tool. It enables a transition from *static* to *homomorphic* NIZKs, and it bridges the gap between semi-malicious and malicious MPC while preserving (asymptotic) communication complexity. Our main results can be summarized as follows.

- **Homomorphic NIZKs from Standard Assumptions.** We construct the first fully-homomorphic NIZK arguments from polynomially-hard LWE, or sub-exponentially hard DDH. Previously, fully homomorphic NIZKs could only be built using bilinear pairings.
- **Rate-Preserving Malicious MPC Compiler from Standard Assumptions.** We construct the first compiler that transforms any semi-malicious MPC protocol into a maliciously secure protocol *without increasing the asymptotic communication complexity or the number of rounds*. When applied to the recent rate-1 semi-malicious MPC protocol [AG26], this yields the first rate-1 maliciously secure MPC from standard lattice assumptions (LWE).

Additionally, we provide many other immediate applications of eRAM such as realizing strong context hiding in multi-hop homomorphic signatures [GVW15b] and designing composable zero-knowledge non-interactive batch arguments [DGKV22, PP22].

The Core Technique: Succinctness with Zero-Knowledge via eRAM. The common technical barrier in both applications is the "verification bottleneck." In MPC, verifying honest behavior typically requires communicating the entire circuit size, destroying the communication rate. In NIZKs, recursing on proofs requires the verifier to be smaller than the circuit it checks. We overcome this by constructing eRAM, a primitive that allows a prover to delegate the execution of a RAM program over encrypted memory, with a proof size and verification time independent of the computation size.

Roadmap. The rest of the paper is organized as follows. In the next section, we provide an overview of our results and main techniques. In Sections 4 and 5, we formally introduce our notion of eRAM delegation and construct it from a variety of standard assumptions. In Section 6, we design extractable NIZKs with optimal efficiency, and use them to build fully homomorphic NIZKs. In Section 7, we present our semi-malicious to malicious compiler for mrNISC and related MPC protocols. We provide some useful preliminaries in Section 3. And, later in Sections 8 and 9, we provide additional results such as composable zero-knowledge batch arguments and delegation protocols for committed programs. We provide a pictorial description of the roadmap in Fig. 1.

1.1 Concurrent Works

Rate-1 NIZKs. Branco, Döttling, and Srinivasan [BDS25] concurrently constructed rate-1 NIZK arguments with slightly super-linear proof size ($|\pi| = |\omega| + |\omega|^{\epsilon}\mathsf{poly}(\lambda)$) from the *sub-exponential* hardness of $\mathsf{LWE}/\mathsf{DDH}/O(1)\text{-}\mathsf{LIN}$. While their work focuses on achieving statistical zero-knowledge, our work targets the construction of homomorphic NIZKs and applications to MPC.

The pivotal difference between the two works lies in **verification efficiency** and its impact on **recursion**.

- [BDS25]: Their verification time scales linearly with the size of the NP verification circuit $|C|$. While they sketch a computational NIZK variant, it inherits this linear verification complexity.
- **Our Work:** We construct a computational NIZK where the verifier runs in time linear in the input and witness length ($O(|x| + |\omega|)$), but independent of the circuit size $|C|$.

This independence is not merely an optimization but a functional requirement for *unbounded recursive composition*. As detailed in the next section, our applications, specifically homomorphic NIZKs, multi-hop zkBARGs, and multi-hop homomorphic signatures fundamentally rely on the ability to recursively verify proofs without parameter explosion. Consequently, the techniques in [BDS25], which lack a succinct verifier, cannot instantiate these applications. Thus, while [BDS25] achieves the stronger property of statistical zero-knowledge (at the cost of linear verification), our work accepts computational zero-knowledge to achieve the succinct verification required for recursive composition and its downstream applications. We note that both our work and [BDS25] were concurrently submitted to CRYPTO 2025.

Malleable SNARKs. Chakraborty et al. [CHL+25] concurrently constructed malleable SNARKs to enable applications such as reverse firewalls. However, their construction relies on non-falsifiable assumptions. In contrast, our homomorphic NIZKs provide similar malleable functionality and

subsume the reverse firewall application, while relying solely on standard falsifiable assumptions (LWE/DDH/$O(1)$-LIN).

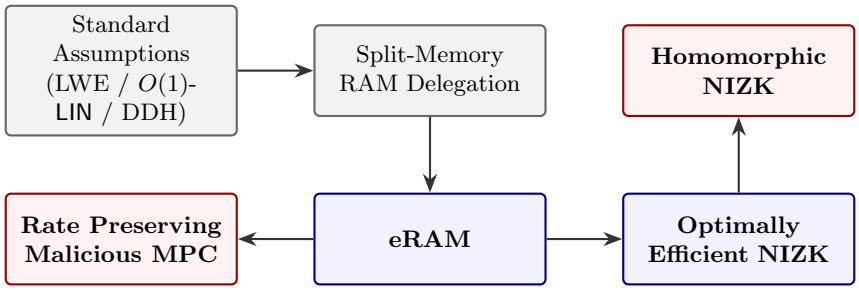

Figure 1: Constructions Roadmap.

2 Technical Overview

2.1 Homomorphic NIZKs

A fully-homomorphic NIZK system [ADKL19] is a special-purpose NIZK [BDSMP91], where an arbitrary sequence of independently generated proofs can be homomorphically combined to generate a fresh NIZK proof certifying validity of the evaluated instance-witness pair. To formally define the homomorphic evaluation process, consider the following NP language where each instance x contains a pair of circuit and boolean output (C, b), and a witness ω is an input string such that $C(\omega) = b$. Homomorphic NIZKs have a special *proof evaluation* algorithm, Eval, that takes k tuples of the form $\{(x_i = (C_i, b_i), \pi_i)\}_{i\in[k]}$ and a boolean circuit $C : \{0,1\}^k \to \{0,1\}$. The goal of the evaluation algorithm is to generate a NIZK proof for the statement (C', b'), where $b' = C(b_1, \ldots, b_k)$ and C' is the circuit C composed with circuit C_i on its i-th input wire. Such a composition maintains the invariant that (C', b') will be a valid instance if every (C_i, b_i) is a valid instance. This is because $C'(\omega_1, \ldots, \omega_k)$ is same as $C(C_1(\omega_1), \ldots, C_k(\omega_k))$, by definition. As discussed above, this captures the multi-hop proof-editing functionality that systems like VerITAS [DCB25] realize only for a single hop.

Unlike other homomorphic primitives, homomorphic NIZKs do not have an additional compactness property for evaluated proofs [ADKL19]. Thus, a more natural approach is to study and design homomorphic NIZKs with basic efficiency. Although designing other homomorphic primitives, such as encryption and signatures, without compactness is usually far easier, this is not the case for homomorphic NIZKs.

To further illustrate the difference, consider the following naïve homomorphic signature construction. An evaluator sets the evaluated proof to be the input instance-proof pairs $\{(x_i = (C_i, b_i), \pi_i)\}_i$ along with evaluation circuit C. A verifier follows a two-step approach: first, it verifies the individual NIZK proofs $\pi_1, \ldots, \pi_k$; second, it checks that $C(b_1, \ldots, b_k)$ matches the boolean output in the instance. The problem with the naïve construction is that it does not preserve zero-knowledge. An evaluated proof leaks intermediate wire values during homomorphic evaluation. E.g., if C is a tree of OR gates, then (ideally) the evaluated proof should hide which/ how many of the input bits from b_1 to b_k are 1. However, the naïve approach leaks that in the clear. More

broadly, the above evaluated proof does not resemble a fresh 'honestly generated proof' for the evaluated instance (C', b'), where $C' = C(C_1(\cdot), \ldots, C_k(\cdot))$ and $b' = C(b_1, \ldots, b_k)$. As noted in earlier works [CKLM12, CKLM13b, ADKL19], such trivial homomorphic evaluations are not interesting or useful for applications.

To eliminate such naïve designs and support interesting applications, Ananth et al. [ADKL19] proposed an unlinkability property (in addition to zero-knowledge) for homomorphic NIZKs. Unlinkability states that an evaluated proof must be indistinguishable from any honestly generated proof for the same instance. Combining the unlinkability property with the zero-knowledge property, we obtain that evaluated proofs are also indistinguishable from simulated proofs.

The current state-of-the-art for fully-homomorphic NIZKs is the pairing-based construction by Ananth et al. [ADKL19]. Their construction builds on the seminal NIZK construction by Groth-Ostrovsky-Sahai (GOS) [GOS06]. At a very high level, during each homomorphic evaluation, they follow a two-step approach: (1) use a commitment to hide the output wires of each input proof, and (2) generate a fresh NIZK proving that it knows openings to the output wire commitments such that they satisfy the circuit being evaluated. To ensure unlinkability, the evaluator also randomizes all incoming wire commitments as well as re-randomizes incoming NIZK proofs to match these randomized commitments. In a sentence, Ananth et al. used intrinsic malleability properties of GOS NIZKs to support homomorphism. For more details, readers are encouraged to refer [GOS06, ADKL19].

Quite clearly, we need to hide the incoming NIZK proofs during homomorphic evaluation, and without relying on proof malleability, hiding an incoming proof seems difficult. To the best of our knowledge, GOS-style NIZKs are the only available NIZKs that are malleable [CKLM12, CKLM13b]. Therefore, existing design templates do not appear useful for diversifying the class of assumptions sufficient for homomorphic NIZKs. To get around this, we propose a new approach.

New approach for homomorphic NIZKs. We draw some inspiration from the classic Gentry's bootstrapping approach for homomorphic encryption [Gen09], and identify a simpler building block that could be bootstrapped to fully-homomorphic NIZKs. Recall that for a regular NIZK proof system, the proof size as well as the verifier running time can scale polynomially with the time it takes to verify the actual NP relation. That is, let T denote the circuit size of an NP relation $\mathcal{R}_{\mathcal{L}}$ that outputs 1 for any valid instance-witness pair $(x, \omega) \in \mathcal{L}$. For regular NIZKs, the proof size, $|\pi|$, as well as the verifier's running time grows polynomially with T. This is the case for all standard model NIZK systems built from standard falsifiable assumptions. This also includes the FHE-based rate-1 NIZK construction by Gentry et al. [GGI+15], where the verifier homomorphically evaluates the relation $\mathcal{R}_{\mathcal{L}}$ on an FHE ciphertext during verification.

Our observation is that an optimally efficient NIZK proof system with three salient features is one such useful building block. To put it formally, if a NIZK system (i) has an *optimal-proof-rate*, which basically means the proof length is asymptotically equal to the witness length, i.e. $|\pi| = |\omega| + \mathsf{poly}(\lambda)$, (ii) has a straight-line knowledge extractor, which means we can extract a valid witness from any accepting proof, and (iii) the verifier's run-time is independent of T, actual NP verification time, then it suffices for designing homomorphic NIZKs.

Optimal-NIZKs to Homomorphism. Given a sequence of instances $\{(C_i, b_i)\}_{i\in[k]}$ and proofs $\{\pi_i\}_{i\in[k]}$, we homomorphically evaluate a circuit C as follows:

Step 1. Commit the incoming wire values $b_1, \ldots, b_k$ to $\mathsf{com}_1, \ldots, \mathsf{com}_k$, and create k NIZK proofs $\pi'_1, \ldots, \pi'_k$ to prove i-th commitment com_i opens to a boolean value b_i such that (C_i, b_i) is

a valid instance. To prove this, the evaluator uses commitment randomness for com_i along with each incoming proof π_i as the new witness.

Step 2. Create an additional NIZK proof π'' to prove that $\mathsf{com}_1, \ldots, \mathsf{com}_k$ commit to boolean values $b_1, \ldots, b_k$ such that $C(b_1, \ldots, b_k)$ matches the boolean value in the evaluated instance.

Step 3. Release $\{\mathsf{com}_i\}_i$, $\{\pi'_i\}_i$, and π'' as the evaluated proof.

Note that the evaluated proof can be verified using a piece-by-piece approach. By which we mean that first all π'_i are individually verified and then π'' is verified. Next, observe that if the underlying NIZK proof system is optimally efficient (it has optimal-proof-rate and has fast verification time), then each evaluated proof can be further evaluated many times. This is because the proof length grows by only an additive $\mathsf{poly}(\lambda)$ factor after each evaluation step due to optimal-proof-rate property. And, the evaluator's running time also can be bounded by a fixed polynomial since the underlying NIZK verifier run-time is fixed.

Moving onto the security guarantees of the above template, zero-knowledge and soundness are fairly straightforward. For zero-knowledge, one can rely on the underlying NIZK being zero-knowledge and the commitment scheme being hiding in tandem. For soundness, we can use the straight-line knowledge extractor to extract wire values for each intermediate wire. This will either give us a satisfying witness, or break binding of the commitment. Finally, moving on to the desirable "unlinkability" property, we face a problem.

The issue is that one can trivially distinguish an evaluated proof from a freshly generated proof by inspecting the proof length. An evaluated proof would be much larger than a fresh proof since during each homomorphic evaluation we pay an additive $\mathsf{poly}(\lambda)$ cost in proof length. While one might try padding to fix this issue, it is unclear whether that could work without putting artificial constraints (such as statically fix the circuit size and topology that can be homomorphically evaluated).

Our idea is to harmonize the prover and evaluator algorithm, and essentially have them run the underlying NIZK prover the same number of times. In particular, we "decompose" each circuit C into non-overlapping groups of "atomic circuits". And, to generate a NIZK proof, a prover would first decompose each circuit and use homomorphic evaluation over all atomic components to build the proof in a step-by-step manner. The important part is each homomorphic operation only evaluates an atomic circuit. To ensure the decomposition is deterministic and consistent, we view each circuit as a directed acyclic graph, and identify the largest set of vertices that can be safely 'cut' to partition the circuit into smaller sub-circuits, where each sub-circuit cannot be further decomposed.

Later in the main body, we formally provide our algorithms for atomic circuit decomposition, and plug them in our base template to design homomorphic NIZKs. For ease of writing in the main body, we merge all individual NIZK proofs created during evaluation into a single proof. We write them individually above for ease of explanation. We remark that our homomorphic NIZK construction supports unbounded homomorphism, and is truly *fully* homomorphic without relying on *any circular security type assumptions*.

We believe our template should also generalize to get non-trivial notions of succinctness for restricted types of homomorphic NIZKs (such as batched succinctness during homomorphism), and leave it as an exciting problem to study. As noted in a prior work [CG25], any non-trivial notion of succinctness for fully-homomorphic NIZKs faces strong barriers. On a separate note, we view optimally-efficient NIZKs as an analogue of bootstrappable homomorphic encryption [Gen09], since they both serve the same purpose. Next, we describe how to design such optimally-efficient

NIZKs via a new tool called eRAM.

2.2 Introducing encrypted RAM delegation

As mentioned earlier, a key ingredient that we use is a new notion of RAM delegation [Mic94, KR09, KRR14, BHK17, PR17], called *encrypted RAM* delegation.

A traditional RAM delegation system has two main algorithms, Prove and Verify.[1] Given a large input x, Prove outputs a short proof π certifying the machine $\mathcal{M}$ accepts x. Verify reads (a short digest of) input x and proof π, and outputs a bit to signal validity of computation. To make this non-trivial, one requires verifier's run-time and proof length to be independent of machine $\mathcal{M}$'s run-time. Moreover, the system should be sound, as in an adversary cannot produce an input x and proof π such that Verify accepts, yet $\mathcal{M}(x)$ rejects.

Encrypted RAM (eRAM) generalizes traditional RAM delegation to guarantee that an input x can be "partially hidden" from the verifier. To define partial hiding, we give x in two disjoint pieces, x_{pb} and x_{pr}, to the Prove algorithm. Here x_{pb} is viewed as a *large public* input, while x_{pr} is a *small private* input. We want the verifier to check validity of the computation, $\mathcal{M}(x) = \mathcal{M}(x_{\mathsf{pb}}, x_{\mathsf{pr}})$, given only the proof π and (the digest of) x_{pb}. Notably, Verify does not get x_{pr}. In all applications of eRAM that we study in this work, x_{pr} is always of fixed polynomial length, thus we only require a slightly relaxed compactness property. In particular, we require the verifier's run-time and proof length to be $\mathsf{poly}(\log T, \log |x_{\mathsf{pb}}|, |x_{\mathsf{pr}}|)$, where T is machine $\mathcal{M}$'s runtime. That is, polynomial dependence on the length of the private input is acceptable.

For security, we require eRAM to satisfy *partial* knowledge soundness and private input hiding properties. Partial knowledge soundness states that there exists an efficient *straight-line* extractor that, given an accepting proof π for public input x_{pb}, can extract a satisfying private input x^*_{pr}. Observe that since we allow the proof size to grow with $|x_{\mathsf{pr}}|$, thus straight-line partial extraction is indeed feasible in the standard model. The private input hiding property states that a proof π corresponding to input $(x_{\mathsf{pb}}, x_{\mathsf{pr}})$ should be efficiently simulatable, given only x_{pb}. Basically, any leakage about x_{pr} in π can be simulated by a zero-knowledge style simulator.

Next, we describe how eRAM can be used to design an optimally efficient NIZKs as needed for homomorphic NIZKs. Later, we discuss key ideas behind our eRAM construction from standard assumptions.

Making NIZKs optimally-efficient. The core idea is pretty straightforward. We will set the NIZK proof to be an eRAM proof, where the original NIZK witness ω will be encrypted and included in the public portion of eRAM input, while the decryption key will be used as the private eRAM input. Concretely, our new NIZK prover starts by sampling a random λ-bit PRG seed sd. It then uses sd to one-time pad encrypt ω as $\hat{\omega} = \omega \oplus \mathsf{PRG}(\mathsf{sd})$, and put the encryption $\hat{\omega}$ in the public eRAM input. Basically, it runs the eRAM prover algorithm with $x_{\mathsf{pb}} = (x, \hat{\omega})$ and $x_{\mathsf{pr}} = \mathsf{sd}$, where the RAM machine first decrypts $\hat{\omega}$ using x_{pr} and then runs the RAM machine for corresponding NP language. The final NIZK proof contains the above eRAM proof and the encrypted witness $\hat{\omega}$.

Given an eRAM proof and encrypted witness, the NIZK verifier can simply run the eRAM verifier. This relies on the eRAM property that eRAM verifier only needs x_{pb} which contains instance x and $\hat{\omega}$. The zero-knowledge of the above design follows directly from the partial input hiding property of eRAM, as the eRAM proof can be simulated without $x_{\mathsf{pr}} = \mathsf{sd}$ and once it is

[1] It additionally requires a trusted setup to sample a CRS, but we ignore it for simplicity.

simulated $\hat{\omega}$ can also be turned to garbage. Next, let us quickly verify that the above design satisfies all properties desired by our homomorphic NIZK template.

Optimal-proof-rate. Since x_{pr} is λ bits long, and $|\hat{\omega}| = |\omega|$, thus by the relaxed compactness of eRAM, we get the resulting NIZK proof is of optimal length.

Straight-line extractor. This follows by using the eRAM partial extractor to extract sd and then decrypting $\hat{w}$ using extracted sd.

Fast verification. Given eRAM verifier's run-time is $\mathsf{poly}(\log T, \log |x_{\mathsf{pb}}|, |x_{\mathsf{pr}}|)$, thus the verifier's run-time is independent of the actual NP verification time.

In summary, eRAM gives us a homomorphic NIZK by plugging in the above NIZK in our new template. To finish up the construction of our homomorphic NIZKs, we provide a construction for eRAM under a wide variety of standard assumptions (such as LWE, $O(1)$-LIN, DDH).

Constructing eRAM from standard assumptions. The starting point for our eRAM construction is the Choudhuri-Jain-Jin (CJJ) [CJJ22, CJJ21] RAM delegation construction based on non-interactive batch arguments (BARGs). Briefly, a BARG lets a prover take a batch of, say, k instance witness pairs $\{(x_i, \omega_i)\}_i$, and output a short proof π, where $|\pi| = \mathsf{poly}(\lambda, |\omega_i|, \log k)$, yet it can certify validity of all k instances $x_1, \ldots, x_k$. The CJJ RAM delegation scheme used BARGs to prove consistency of a table of RAM computation states with a proof size that is independent of the total computation time. At a high level, their delegation scheme for machine $\mathcal{M}$ can be summarized as follows.

Let cf_0 denote the initial configuration (including the input x), and cf_i denote the machine $\mathcal{M}$'s configuration after i computation steps. The idea underlying the CJJ RAM delegation scheme is to split the global consistency check between the initial and final configuration, to a sequence of local consistency checks between any two consecutive configurations cf_{i-1} and cf_i. Namely, the prover generates a short hash digest c of all configurations $(\mathsf{cf}_0, \ldots, \mathsf{cf}_T)$, then generates a BARG where the i-th instance certifies: (a) $\mathsf{cf}_i = \mathsf{nxt\text{-}cnfg}(\mathsf{cf}_{i-1})$, where $\mathsf{nxt\text{-}cnfg}$ is the step function for $\mathcal{M}$, and (b) cf_{i-1} and cf_i are $(i-1)$-th and i-th configurations as per the digest c.

Although the proof size in the above does not grow with the total number of configurations, it does scale with a single configuration size which could be larger than $\mathcal{M}$'s running time or the input length. CJJ fixed this by hashing down each configuration before creating the digest. That is, each cf_i is first hashed to hcf_i, and then all $\{\mathsf{hcf}_i\}_i$ are hashed together to obtain the final digest c. With this change, the prover generates the BARG proof where the i-th instance certifies that the $(i-1)$-th and i-th hashed configurations, hcf_{i-1} and hcf_i, are consistent as per $\mathsf{nxt\text{-}cnfg}$. Note that any two consecutive RAM machine configurations are identical except at a few places, thus their consistency can be checked efficiently by opening hcf_{i-1} and hcf_i for those few locations. The proof of soundness relies on a very delicate strategy of combining somewhere extractability of BARGs [CJJ22, CJJ21] with a strong binding property for the underlying hash function [HW15, OPWW15]. We refer to [CJJ22] for more details.

Is CJJ already partially hiding? Unfortunately, the above approach leaks the input. There are two main reasons. First, the configuration digest c is a deterministic hash value, thus it could leak the starting configuration that contains x. Second, standard BARGs do not hide the witnesses, thus the batch proof could leak information about the bits read/written during a single computation step.

In order to control this leakage, we generalize the CJJ template and rely on additional standard tools. Our idea is to encrypt each hashed configuration hcf_i and use a zero-knowledge BARG

(zkBARG) for proving local consistency. However, a straightforward execution of the above idea will not work. The reason is that the verifier does not know x_{pr}, thus it cannot check whether the final digest c was correctly computed. This is an essential step in the CJJ template. In eRAM, however, the verifier only gets x_{pb} and not x_{pr}. Therefore, we need the eRAM verifier to be able to verify the accuracy of digest c given only x_{pb}.

To address this, we partition each RAM configuration into two parts. First part contains only x_{pb}, while second part contains x_{pr} and rest of the RAM configuration (such as work tape). Next, we hash these two components separately and only encrypt the second part. This gives us a deterministic hash of x_{pb} and randomized hash of everything else (including x_{pr}, work tape). We view these two hashes jointly as the new hashed configuration hcf, which are hashed as before to create the final digest c. Now the eRAM prover uses a zkBARG to prove local correctness à la CJJ, with one modification that it additionally uses encryption randomness as a witness. Finally, the prover outputs the zkBARG proof, digest c, and a hash opening to the digest of x_{pb} w.r.t. c. Given all of this, a verifier creates a digest for x_{pb}, and checks the validity of its opening w.r.t. digest c. The rest of zkBARG proof verification proceeds as before. This seems to resolve the previous issue of eRAM verifier struggling to verify digest c given only x_{pb}. However, we do not know how to prove security of the above design. The reason is that to prove soundness CJJ crucially relied on the reduction algorithm (verifier) knowing the full input x. We don't have that in case of eRAM, thus the CJJ proof strategy does not directly apply.

Our intuition is that this final problem can be resolved if the reduction algorithm can extract x_{pr} from a valid proof by using some trapdoor. To execute this, we make one last change. We no longer hash x_{pr} before encryption, but simply encrypt it as is along with the hash of the work tape (and rest of the RAM configuration). The advantage of the above is that the reduction algorithm can use the secret key for the encryption system as the trapdoor to extract x_{pr} and use that to implement a CJJ-style proof strategy for proving soundness. We point out that since we no longer hash down x_{pr}, thus the final proof size scales at least linearly with $|x_{\mathsf{pr}}|$. However, it grows only poly-logarithmically with $|x_{\mathsf{pb}}|$ and run-time of $\mathcal{M}$, thereby satisfies relaxed compactness property for eRAM. Finally, partial input hiding property follows from semantic security of encryption and zero-knowledge property of zkBARGs.

In the main body, rather than fully opening up the CJJ construction, we provide a more direct semi-generic construction for eRAM. We prove that the CJJ RAM construction can be extended to work in a 'split memory' model and use this model to directly build eRAM. Additionally, we introduce a *reusability* notion for eRAM, and describe how our eRAM construction can be easily turned into a reusable eRAM scheme. To understand reusable eRAM, consider the following scenario: a user wants to create multiple eRAM proofs π_1, π_2 (and so on) for inputs x_1, x_2 (and so on), where the private portion of each input is identical (i.e. $x_{1,\mathsf{priv}} = x_{2,\mathsf{priv}} \ldots$). A reusable eRAM lets us amortize the cost of x_{pr} across many proofs. Basically, using reusable eRAM a user can encode x_{pr} just once, and this encoding can be reused in all proofs that share the same private input. The added advantage of generating this encoding is that the non-reusable/fresh part in each new proof is very short, and no longer grows with even $|x_{\mathsf{pr}}|$ anymore. As we explain next, reusable eRAM is very useful in MPC settings. More details are provided in Section 5.

2.3 Complexity-Preserving Semi-Malicious to Malicious MPC

Multiparty computation (MPC) [Yao86, GMW87] protocols are a staple in modern cryptography. They allow mutually distrusting parties to jointly and securely compute a function f over

distributed inputs. The most desirable notion of security for MPC protocols is of full malicious security, where adversaries can behave arbitrarily to learn the honest parties' inputs. Typically, designing a maliciously secure MPC protocol is a difficult task. Therefore, a popular approach in the literature is to first design an MPC that is secure in a weaker semi-malicious model, and later compile it to the (full) malicious model. One such compiler was originally presented by Asharov et al. [AJLA+12], but it was not very efficient. In particular, if they start with a semi-malicious protocol with optimal communication complexity, the compiled malicious protocol is very far from optimal communication complexity. While it is known that under a strong non-falsifiable assumption (such as zk-SNARKs) the blowup in communication complexity can be avoided [DGMR22], we do not yet have any approaches to achieve the same under standard falsifiable assumptions.

We show that reusable eRAM is a great tool to build (optimal) complexity-preserving compilers from semi-malicious MPC to malicious MPC. By complexity-preserving, we mean that the communication overhead added by the compiler is independent of the size of the MPC functionality, inputs, or the number of parties. To the best of our knowledge, our work provides the *first* complexity-preserving compiler for semi-malicious to malicious security from standard falsifiable assumptions.

For ease of exposition, we describe our compiler in the context of multiparty reusable non-interactive secure computation (mrNISC) protocols [BL20]. In a mrNISC protocol, each party (say P_i) encodes its private inputs (say x_i) using a special Com algorithm such that this encoding can be reused across many different protocol runs. Each encoding, $\widehat{x}_i$, is widely accessible via a public bulletin board. Consider any set S of parties, say $\{P_i\}_{i\in S}$, that want to jointly evaluate a function f on their private inputs $\{x_i\}_{i\in S}$. A mrNISC protocol allows each party, P_i, to individually (and without interaction) create an evaluated encoding $\widehat{f}_i$ using the $\mathsf{FuncEnc}$ algorithm. $\mathsf{FuncEnc}$ only requires the input encodings $\{\widehat{x}_j\}_{j\in S}$ and the *de-commitment* information corresponding to $\widehat{x}_i$. All these encodings together can be used to learn $f(x_1,\ldots,x_n)$, but nothing more. Essentially, mrNISC makes the first round of the protocol to be independent of the function f and the set S of parties. Thus, it brings down the cost of computing k functions to just $k+1$ rounds. Over the last few years, multiple constructions for mrNISC with semi-maliciously security from standard assumptions have been proposed [BL20, BJKL21, AJJM21, Shi22]. Next, we boost their security to malicious security with identical (asymptotic) communication complexity.

Malicious security via reusable eRAM. During the input encoding process, each party creates a commitment $\widehat{x}_i$ for its private input x_i using the semi-malicious protocol and, additionally, it generates a reusable eRAM encoding $\widehat{\pi}_i$ for x_i and commitment randomness as the reusable eRAM *private* input. Together $\widehat{\pi}_i$ and $\widehat{x}_i$ correspond to the new input encoding for our malicious protocol.

To generate a function encoding, each party first generates the semi-malicious function encoding $\widehat{f}_i$. Next, it generates an eRAM proof using the semi-malicious function encoding algorithm as the RAM machine and $\widehat{f}_i$ and all the encodings $\{\widehat{x}_j\}_{j\in S}$ as the eRAM *public* input. Basically, the RAM machine checks that $\widehat{f}_i$ was correctly computed given input x_i, its commitment randomness, and public encodings of rest of the parties. Each party outputs the eRAM proof and the semi-malicious function encoding as the new function encoding. An evaluator runs eRAM verifier to check each proof is valid and then uses the semi-malicious evaluator to learn the output of the computation.

The correctness of the malicious protocol follows in a straightforward way. Moreover, we can rely on the partial knowledge soundness and input hiding property for our reusable eRAM to reduce any successful attacker on our new protocol to a successful semi-malicious attacker on the base protocol. The proof relies on the fact that partial knowledge soundness can be used to extract

private inputs for malicious parties, while eRAM input hiding property and semi-malicious security of underlying MPC can be used to simulate the honest parties' messages.

Interestingly, the above compiler preserves the function class for the underlying mrNISC protocol, but it is not fully complexity-preserving yet. The reason is that it turns out the commitment randomness sampled by each party at the start must be as long as each input x_i. Thus, even when our reusable eRAM provides best possible succinctness, the final input encoding will be at least *two* times as large (i.e., a blowup of at least a multiplicative factor of 2).

Thus, to compile a semi-malicious protocol in a truly complexity-preserving manner, we need a few more ideas. At a very high level, our insight is to use another level of delegation within our compiler, and take out each parties' private input x_i and the corresponding commitment randomness from the eRAM private input, and move it to the public input. While this could possibly get around the multiplicative overhead issue, we need to ensure x_i (and the commitment randomness) is still hidden.

To this end, our final trick is to have the i-th party sample a short random seed sd_i and use it to encrypt its input x_i as $\hat{x}_i = x_i \oplus \mathsf{PRG}(\mathsf{sd}_i)$. Now rather than using x_i as the private eRAM input, each party uses sd_i as the private eRAM input and includes $\hat{x}_i$ in the public portion of the eRAM input. Thus, party P_i sets $\hat{x}_i$ in addition to the semi-malicious input encoding and an eRAM encoding for sd_i as the final input encoding for the malicious protocol. Observe that this encoding is now really complexity-preserving as each sd is of fixed polynomial length, and the actual input x_i is encrypted using a one-time pad.

Now, to make sure the correctness of the protocol stays intact, we need to slightly adjust the function encoding process as well. In particular, instead of generating function encodings for actual function f, we generate it for a new function g. This new function g has the original function f as well as the one-time input encryptions $\hat{x}_1, \ldots, \hat{x}_n$ hardwired, and it takes as input a sequence of PRG seeds, and outputs the following:

$$g_{[f,\hat{x}_1,\ldots,\hat{x}_n]}(\mathsf{sd}_1, \ldots, \mathsf{sd}_n) = f(x_1, \ldots x_n)$$

To prove security of the resulting protocol, we can rely on similar ideas as our previous protocol, except we need to additionally rely on pseudorandomness of the PRG. Later, in Section 7, we provide our full construction and discuss its extensions to other MPC protocols.

Additionally, we describe a new maliciously-secure mrNISC protocol (assuming the hardness of circularly secure LWE) where the input encoding length is $|x| + \mathsf{poly}(\lambda)$ (i.e., optimal-rate) and the function encoding length is $\ell \cdot \mathsf{poly}(\lambda)$ where ℓ denotes the function output length. Thus, our function encoding length is independent of the computation size. Prior work [AJJM21] informally sketched out a method in which function encodings grow with ℓ as well as depth of the function. However, we show that by relying of multi-key FHE [LATV12, CM15, MW16], we can avoid this dependence and realize function encodings with $\ell \cdot \mathsf{poly}(\lambda)$ size in the crs model (required for security against malicious adversaries). At a high level, the idea is to delegate the computation of f to multi-key FHE and generate function encodings for multi-key FHE's decryption function which only depends on ℓ. We provide more details in Section 7. We leave it as an interesting problem to extend this to SXDH including further reduction to the function encoding length to make it fully-optimal.

2.4 Other Applications

The notion of eRAM turns out to be immediately useful for more applications. We briefly discuss a few below and leave it as an interesting future research direction to find more applications.

Context-Hiding Multi-Hop Multi-Key Homomorphic Signatures. Homomorphic signatures [JMSW02, AB09, BFKW09, BF11, GVW15b] enable computations on *secretly* signed data. Given a signature σ for database D, they let anybody publicly compute a signature for $f(D)$, for any efficiently computable function f. A highly desirable property for homomorphic signatures is the ability to compute on evaluated signatures (called multi-hop evaluation) and a guarantee that any evaluated signature looks just like a fresh signature (called strong context hiding) [GVW15b].

By plugging our notion of eRAM delegation in prior BARG-based homomorphic signature schemes [ACG26], we obtain a homomorphic signature scheme satisfying general multi-hop homomorphic evaluation and strong context hiding, under sub-exponential hardness of $\mathsf{LWE}/\mathsf{DDH}/O(1)\text{-}\mathsf{LIN}$. Furthermore, the homomorphic signature scheme readily supports the multi-key model [GVW15b, FMNP16] as well. Prior to our work, homomorphic signatures satisfying strong multi-hop context hiding were not known from standard falsifiable assumptions. We refer the reader to [ACG26] for a detailed discussion about limitations of prior works.

Composable zkBARGs. Composable or multi-hop batch arguments were recently introduced by Devadas et al. [DGKV22] as a new tool to design incrementally verifiable computation protocols [Val08] and aggregate signatures with unbounded aggregation [BGLS03]. Informally, a multi-hop BARG allows succinct recursive composition of multiple BARG proofs.

In this work, we provide a simple construction for a multi-hop *zero-knowledge* BARG. Our construction essentially combines the techniques of [DGKV22] with our eRAM delegation object to get the desired zero-knowledge property. As an immediate application, this gives a privacy-preserving incrementally verifiable computation protocol, where by privacy-preserving we mean the incrementally computed proof can hide information about the intermediate computation steps as well as starting input. For completeness, we provide more details in Section 8.

Succinct Delegation of Committed Programs. In a recent work, Ghosal et al. [GSW23] introduced a new notion for SNARGs called semi-trusted SNARGs. Their goal was to assist users that wish to delegate the responsibility of hosting a program to a server. As an immediate application, we show that reusable eRAM immediately gives semi-trusted SNARGs with a stronger zero-knowledge guarantee. For completeness, we give our reusable eRAM based construction in Section 9.

2.5 Related Work

The concept of eRAM delegation introduced in this work can be viewed as a "partially hiding" generalization of vanilla RAM delegation [Mic94, KR09, KRR14, BHK17, PR17]. In this work, we do not require strong succinctness from eRAM (which would require proof size to be independent of private input length too). A recent work [CG25] shows any extractable argument system with non-trivial succinctness faces strong barriers under standard falsifiable assumptions [GW11]. Since we require our eRAM system to have partial extractability, thus this recent infeasibility result [CG25] also rules out any efficiency improvement to our eRAM delegation construction under standard falsifiable assumptions.

More broadly, the intuition behind our eRAM notion weakly resembles partially-hiding functional encryption (PHFE) introduced by Gorbunov-Vaikuntanathan-Wee [GVW15a]. PHFE generalizes vanilla FE, where the input to the encryption algorithm is split into two parts (public and private). Additionally, in PHFE, "large" computation can only be performed on public input and only a "small" computation can be carried on secret input. It is known [GVW15a, AJL+19, BSW11] that PHFE is far easier to design than general-purpose FE. Drawing inspiration from PHFE, eRAM

generalizes (zero-knowledge) RAM delegation in a similar manner, where the size of public input can be large, while the size of private input must be small.

3 Preliminaries

Notation. We denote the security parameter by λ. By PPT, we denote a probabilistic polynomial-time. All polynomials denoted by $\mathsf{poly}(\cdot)$ are positive polynomials. For any finite set $\mathcal{S}$, $x \leftarrow \mathcal{S}$ denotes a uniformly random element $x \in \mathcal{S}$. Similarly for any distribution $\mathcal{D}$, $x \leftarrow \mathcal{D}$ denotes an element x drawn from distribution $\mathcal{D}$. We denote the set of all positive integers up to n as $[n] := \{1, \ldots, n\}$. Also, we use $[m, n]$ where $n \geq m$ to denote the set of all integers from m to n, i.e, $[m, n] := \{m, \ldots, n\}$. By $\mathsf{negl}(\lambda)$, we define negligible functions. A function $\mathsf{negl} : \mathbb{N} \to \mathbb{R}$ is a negligible function if for every $c \in \mathbb{N}$ and for large enough λ, $\mathsf{negl}(\lambda) < \lambda^{-c}$.

3.1 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:

- 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 W_i (for $i \in [S]$) 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 $C_{\mathcal{M}}$ that computes the next step as follows:

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

We use cf_i for the configuration of the machine when run for i steps, and let cf_0 be the initial configuration of the machine. We denote by $\mathcal{M}(z; 1^t)$ running the machine $\mathcal{M}$ starting from cf_0 on input z for t steps that outputs cf_t. The language of machine $\mathcal{M}$ is defined as follows:

$$\mathcal{L}_{\mathcal{M},T} = \left\{(z, \mathsf{out}) : \mathsf{cf}_0 = \text{initial configuration} \wedge \mathcal{F}(\mathsf{cf}_T) = \mathsf{out} \wedge \mathcal{M}(z; 1^T) = \mathsf{cf}_T\right\} \tag{1}$$

where $\mathcal{F}(\cdot)$ is a deterministic function that computes the output of the machine from the final configuration.

In this work, we focus on RAM machines that output a single bit 0/1. In this case, the language would be as follows:

$$\mathcal{L}_{\mathcal{M},T} = \left\{(z,b) : \mathsf{cf}_0 = \text{initial configuration} \ \wedge \ \mathcal{F}(\mathsf{cf}_T) = b \ \wedge \ \mathcal{M}(z;1^T) = \mathsf{cf}_T\right\} \tag{2}$$

3.2 Somewhere Extractable (Zero-Knowledge) Batch Arguments

Syntax. A (publicly verifiable and non-interactive) somewhere extractable batch argument scheme seBARG for an NP language $\mathcal{L}$ 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 3.1 (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 $i_0, i_i \in [k]$ 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 $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l} \mathsf{Verify}(\mathsf{crs}, x_1, \ldots, x_k, \pi) = 1 \\ \wedge \ w^* \text{ is not a valid} \\ \text{witness for } x_{i^*} \in \mathcal{L} \end{array} : \begin{array}{l} (\mathsf{crs}, \mathsf{td}) \leftarrow \mathcal{E}(1^\lambda, k, n, i^*) \\ (x_1, \ldots, x_k, \pi) \leftarrow \mathcal{A}(\mathsf{crs}) \\ w^* \leftarrow \mathcal{E}\left(\mathsf{td}, \{x_i\}_{i\in[k]}, \pi\right) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

Remark 3.2. 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 3.3 (zkBARG)**.** A somewhere-extractable zero-knowledge batch argument scheme $\mathsf{zkBARG} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ for $\mathcal{L}$ is required to satisfy the following property in addition to all properties under Definition 3.1:

Zero-knowledge. There exists a PPT simulator $\mathcal{S}$ 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 $\lambda \in \mathbb{N}$,

$$\left|\begin{array}{l}\Pr\left[\ 1 \leftarrow \mathcal{A}(\mathsf{crs})^{\mathsf{Prove}(\mathsf{crs},\cdot,\cdot)} \ : \ \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, k, n, i^*)\ \right] \\ \quad - \Pr\left[\ 1 \leftarrow \mathcal{A}(\mathsf{crs})^{\mathcal{O}^{\mathcal{S}}(\mathsf{crs},\cdot,\cdot)} \ : \ \mathsf{crs} \leftarrow \mathcal{S}(1^\lambda, k, n)\ \right]\end{array}\right| \leq \mathsf{negl}(\lambda).$$

where $\mathcal{O}^{\mathcal{S}}((x_1, \cdots, x_k), (w_1, \ldots, w_k))$ outputs $\mathcal{S}(x_1, \ldots, x_k)$ if for every $i \in [k]$, w_i is a valid witness for $x_i \in \mathcal{L}$ and $\perp$ otherwise.

Definition 3.4 (Rate-1 seBARG)**.** An seBARG scheme $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ is said to be rate-1 if the proof generated by $\mathsf{Prove}(\mathsf{crs}, x_1, \ldots, x_k, w_1, \ldots, w_k)$ is of length $m + O(m/\lambda) + \mathsf{poly}(\lambda)$.

Theorem 3.5 ([DGKV22, PP22, KLVW23])**.** Assuming $X \in \{\mathsf{LWE}, O(1)\text{-}\mathsf{LIN}, \text{sub-exp-}\mathsf{DDH}\}$, there exists rate-1 seBARG.

3.3 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.

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

$\mathsf{Open}(\mathsf{hk}, x, i) \to (b, \mathsf{op})$**.** This is the hash opening algorithm that takes as input the hash key hk, an input $x \in \{0,1\}^*$ and an index $i \in [|x|]$, and outputs a bit $b \in \{0,1\}$ and an opening $\mathsf{op} \in \{0,1\}^{\leq \mathsf{poly}(\lambda)}$.

$\mathsf{Write}(\mathsf{hk}, x, i, b) \to (\mathsf{h}', \mathsf{op}')$**.** This is the writing algorithm that takes as input the hash key hk, an input $x \in \{0,1\}^*$ an index $i \in [|x|]$ and a bit $b \in \{0,1\}$, and outputs a hash value $\mathsf{h}' \in \{0,1\}^{\mathsf{poly}(\lambda)}$ an opening $\mathsf{op}' \in \{0,1\}^{\leq \mathsf{poly}(\lambda)}$.

$\mathsf{VerifyRead}(\mathsf{hk}, \mathsf{h}, i, b, \mathsf{op}) \to 0/1$**.** This is a read-verification algorithm that takes as input the hash key hk, a hash value $\mathsf{h} \in \{0,1\}^{\mathsf{poly}(\lambda)}$, an index $i \in [|x|]$, a bit $b \in \{0,1\}$ and an opening $\mathsf{op} \in \{0,1\}^{\leq \mathsf{poly}(\lambda)}$, and outputs 0 (reject) or 1 (accept).

$\mathsf{VerifyWrite}(\mathsf{hk}, \mathsf{h}, i, b, \mathsf{h}', \mathsf{op}') \to 0/1$**.** This is a write-verification algorithm that takes as input the hash key hk, a hash value $\mathsf{h} \in \{0,1\}^{\mathsf{poly}(\lambda)}$, an index $i \in [|x|]$, a bit $b \in \{0,1\}$, a hash value $\mathsf{h}' \in \{0,1\}^{\mathsf{poly}(\lambda)}$ and an opening $\mathsf{op}' \in \{0,1\}^{\leq \mathsf{poly}(\lambda)}$, and outputs 0 (reject) or 1 (accept).

Definition 3.6 (Hash Tree)**.** A hash tree $\mathsf{HT} = (\mathsf{Gen}, \mathsf{Hash}, \mathsf{Open}, \mathsf{Write}, \mathsf{VerifyRead}, \mathsf{VerifyWrite})$ is required to satisfy the following properties:

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

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

$$\Pr\left[\begin{array}{lcl} \mathsf{VerifyRead}(\mathsf{hk}, \mathsf{h}, i, b_i^1, \mathsf{op}_i^1) = 1, & & (\mathsf{hk}) \leftarrow \mathsf{Gen}(1^\lambda), \\ \mathsf{VerifyRead}(\mathsf{hk}, \mathsf{h}, i, b_i^2, \mathsf{op}_i^2) = 1, & : & (\mathsf{h}, i, b_i^1, \mathsf{op}_i^1, b_i^2, \\ b_i^1 \neq b_i^2 & & \mathsf{op}_i^2) \leftarrow \mathcal{A}(\mathsf{hk}) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

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

$$\Pr\left[\begin{array}{lcl} \mathsf{VerifyWrite}(\mathsf{hk}, \mathsf{h}, i, b, \mathsf{h}_1, \mathsf{op}_1) = 1, & & (\mathsf{hk}) \leftarrow \mathsf{Gen}(1^\lambda), \\ \mathsf{VerifyWrite}(\mathsf{hk}, \mathsf{h}, i, b, \mathsf{h}_2, \mathsf{op}_2) = 1, & : & (\mathsf{h}, i, b, \mathsf{h}_1, \mathsf{op}_1, \mathsf{h}_2, \mathsf{op}_2) \\ \mathsf{h}_1 \neq \mathsf{h}_2 & & \leftarrow \mathcal{A}(\mathsf{hk}) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

Remark 3.7. Reading soundness implies collision resistance of the hash tree.

Remark 3.8 ([Mer87])**.** Hash trees can be constructed using any family $\mathcal{F} : \{0,1\}^{2\lambda} \to \{0,1\}^\lambda$ of collision-resistance hash functions.

3.4 Non-Interactive Zero-Knowledge Arguments (NIZK)

Consider an NP language $\mathcal{L} = \{x \mid \exists w : \mathcal{R}(x, w) = 1\}$ defined w.r.to 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 3.9 (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 $\lambda, n_x \in \mathbb{N}$:

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

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.

Straight-line 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 $\lambda, n_x, n_w \in \mathbb{N}$:

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

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

Remark 3.10 ([CW23, BWW24, BKP+24])**.** Assuming seBARGs there exists NIZKs.

Definition 3.11 (NIZK with strong extraction)**.** A NIZK scheme $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ for language $\mathcal{L}$ is said to be a NIZK scheme with strong extraction if it satisfies Definition 3.9 and the following property:

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

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

Note that this property implies straight-line extraction property from Definition 3.9.

Remark 3.12 (NIZK with strong extraction)**.** Strong extraction is an additional property which can be achieved using a PKE scheme (Definition 3.16) in conjunction with soundness of NIZK scheme.

Definition 3.13 (Optimally Efficient NIZK)**.** A NIZK scheme $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ for language $\mathcal{L}$ is said to be an optimally efficient NIZK if it satisfies Definition 3.9 and the size of the proof π is $|w| + \mathsf{poly}(\lambda, \log|x|, \log|w|)$, i.e., rate-1 proof size and running time of Verify is independent of size of $\mathcal{R}$. Moreover, we require that the setup of such NIZK scheme be oblivious to the language, i.e., the CRS is universal and independent of the language, and in fact, the language description can depend on the CRS.

3.5 Rate-1 Message Encoding

A rate-1 message encoding is a one-time encryption system, with the special guarantee that the secret key length is independent of message length. Such an encoding follows directly by combining a one-time pad with a pseudorandom generator/function. Since we also use such encodings in our MPC compiler, we decided to abstract it out. Below is the formal definition.

Definition 3.14 (r1Enc)**.** A rate-1 message encoding r1Enc consists of two algorithms: $(\mathsf{Encode}, \mathsf{Decode})$. Encode takes as input a security parameter λ, a string $x \in \{0,1\}^*$, and outputs an encoded value ρ and secret key sk. Decode takes ρ and sk as inputs, and outputs a string x. Together they must satisfy the following:

Completeness and compactness. For any $\lambda \in \mathbb{N}$, $x \in \{0,1\}^*$, $\mathsf{Decode}(\rho, \mathsf{sk}) = x$, $|\rho| = |x|$, and $|\mathsf{sk}| = \mathsf{poly}(\lambda)$, where $(\rho, \mathsf{sk}) \leftarrow \mathsf{Encode}(1^\lambda, x)$.

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

$$\Pr\left[\begin{array}{lcl} & & x \leftarrow \mathcal{A}(1^\lambda), b \leftarrow \{0,1\} \\ \mathcal{A}(\rho_b) = b & : & (\rho_0, \mathsf{sk}) \leftarrow \mathsf{Encode}(1^\lambda, x), \\ & & \rho_1 \leftarrow \{0,1\}^{|x|} \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

Remark 3.15. A r1Enc scheme can be constructed using any family of pseudorandom functions such that $\mathsf{PRF}_\lambda : \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}$ used in counter mode. In addition, sk will be a λ-bit string.

3.6 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 3.16 (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 $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\lambda)$.

3.7 RAM Delegation Scheme

Syntax. A publicly verifable non-interactive delegation scheme for RAM machine $\mathcal{M}$ w.r.t. a hash tree HT with hash key ht.hk consists of the following PPT algorithms:

$\mathsf{Setup}(1^\lambda, T) \to \mathsf{crs}$: The setup algorithm takes as input security parameter λ and running time bound T. It outputs crs.

$\mathsf{Digest}(\mathsf{crs}, \mathsf{cf}) \to \mathsf{h}$: This is a deterministic polynomial time algorithm that takes as input a crs, and a configuration cf and outputs a digest h.

$\mathsf{Prove}(\mathsf{crs}, \mathsf{cf}_0) \to (b, \pi)$: The prover algorithm takes as input a common reference string crs, and an initial configuration cf_0, and outputs a bit b and a proof π.

$\mathsf{Verify}(\mathsf{crs}, \mathsf{h}_0, b, \pi) \to \{0,1\}$: The verifier algorithm takes as input crs, a digest of the initial configuration h_0, a bit b, and a proof π, and outputs either 0 or 1.

Definition 3.17 (RAM Delegation)**.** A publicly verifiable non-interactive RAM delegation scheme $(\mathsf{Setup}, \mathsf{Digest}, \mathsf{Prove}, \mathsf{Verify})$ for $\mathcal{M}$ with setup time T_S and proof length ℓ_π is required to satisfy the following properties:

Completeness. For every polynomial $\lambda, T \in \mathbb{N}$ s.t. $T \leq 2^\lambda$, and $\mathsf{cf}_0 \in \{0,1\}^*$ such that $\mathsf{cf}_0 \in \mathcal{L}_{\mathcal{M},T}$ it holds that

$$\Pr\left[\begin{array}{l|ll} \mathsf{Verify}(\mathsf{crs}, \mathsf{h}_0, b, \pi) = 1 & : & \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, T) \\ & & \mathsf{h}_0 \leftarrow \mathsf{Digest}(\mathsf{crs}, \mathsf{cf}_0) \\ & & (b, \pi) = \mathsf{Prove}(\mathsf{crs}, \mathsf{cf}_0) \end{array}\right] = 1.$$

Efficiency. In the completeness experiment above,

- Setup runs in time T_S.
- Digest on input cf runs in time $|\mathsf{cf}| \cdot \mathsf{poly}(\lambda)$ and outputs a digest of length λ.
- Prove runs in time $\mathsf{poly}(\lambda, T, |\mathsf{cf}|)$ and output a proof of length ℓ_π.
- Verify runs in time $\mathcal{O}(\ell_\pi) + \mathsf{poly}(\lambda)$.

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

$$\Pr\left[\begin{array}{l|ll} \mathsf{cf} \neq \mathsf{cf}' & : & \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, T) \\ \mathsf{Digest}(\mathsf{crs}, \mathsf{cf}) = \mathsf{Digest}(\mathsf{crs}, \mathsf{cf}') & & (\mathsf{cf}, \mathsf{cf}') \leftarrow \mathcal{A}(\mathsf{crs}) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

Soundness. For every PPT adversary $\mathcal{A}$ and pair of polynomials $T = T(\lambda)$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that for every $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l|l} \mathsf{Verify}(\mathsf{crs}, \mathsf{h}_0, b, \pi) = 1 & \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, T) \\ \mathsf{cf}_0 \notin \mathcal{L}_{\mathcal{M},T} & (\mathsf{cf}_0, b, \pi) \leftarrow \mathcal{A}(\mathsf{crs}) \\ & \mathsf{h}_0 \leftarrow \mathsf{Digest}(\mathsf{crs}, \mathsf{cf}_0) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

Strong Soundness. For every PPT adversary $\mathcal{A}$ and pair of polynomials $T = T(\lambda)$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that for every $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l|l} \mathsf{Verify}(\mathsf{crs}, \mathsf{h}_0, 0, \pi^{(0)}) = 1 & \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, T) \\ \mathsf{Verify}(\mathsf{crs}, \mathsf{h}_0, 1, \pi^{(1)}) = 1 & (\mathsf{h}_0, \pi^{(0)}, \pi^{(1)}) \leftarrow \mathcal{A}(\mathsf{crs}) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

Remark 3.18 ([CJJ22])**.** Assuming seBARGs and SEH there exists RAM delegation.

4 Defining (Reusable) Encrypted RAM Delegation

In this section we formally define the notions of encrypted RAM delegation and reusable encrypted RAM delegation.

4.1 Encrypted RAM Delegation

Syntax. An encrypted RAM delegation scheme for a RAM machine $\mathcal{M}$ consists of the following PPT algorithms:

$\mathsf{Setup}(1^\lambda, n_{\mathsf{pb}}, 1^{n_{\mathsf{pr}}}, S, T) \to \mathsf{crs}$**.** The probabilistic setup algorithm takes as input a security parameter 1^λ, a public input length n_{pb}, a private input length n_{pr}, a max configuration size S for the RAM machine, and the running time T of the machine. It outputs a common reference string crs.

$\mathsf{Prove}(\mathsf{crs}, x_{\mathsf{pb}}, x_{\mathsf{pr}}) \to (b, \pi)$**.** The proving algorithm takes as input the common reference string crs, a public input value x_{pb} and a private input value x_{pr}. It outputs a bit b and a proof π.

$\mathsf{Verify}(\mathsf{crs}, \mathsf{h}_{\mathsf{pb}}, b, \pi) \to 0/1$**.** The verification algorithm takes as input the common reference string crs, a digest h_{pb} of the public input, a bit b and a proof π. It outputs 0 (reject) or 1 (accept).

Definition 4.1 (Encrypted RAM Delegation)**.** An encrypted RAM delegation scheme $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ for $\mathcal{M}$ associated with a hash tree HT with hash key $\mathsf{ht.hk}$ has to satisfy the following property:

Completeness. For every $\lambda, T, S, n_{\mathsf{pb}}, n_{\mathsf{pr}} \in \mathbb{N}$ s.t. $T, S, n_{\mathsf{pb}} \leq 2^\lambda$, $n_{\mathsf{pr}} = \mathsf{poly}(\lambda)$, $x_{\mathsf{pb}} \in \{0,1\}^{n_{\mathsf{pb}}}$, $x_{\mathsf{pr}} \in \{0,1\}^{n_{\mathsf{pr}}}$, $(x_{\mathsf{pb}}, x_{\mathsf{pr}}) \in \mathcal{L}_{\mathcal{M},T}$ it holds that:

$$\Pr\left[\begin{array}{l|l} \mathsf{Verify}(\mathsf{crs}, \mathsf{h}_{\mathsf{pb}}, b, \pi) = 1 & \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, n_{\mathsf{pb}}, 1^{n_{\mathsf{pr}}}, S, T) \\ & (b, \pi) \leftarrow \mathsf{Prove}(\mathsf{crs}, x_{\mathsf{pb}}, x_{\mathsf{pr}}) \end{array}\right] = 1.$$

Efficiency. The crs and proof size, and the setup and the verifier running time are $\mathsf{poly}(\lambda, \log T, \log n_{\mathsf{pb}}, n_{\mathsf{pr}})$, and the prover's running time is $\mathsf{poly}(\lambda, T, n_{\mathsf{pb}}, n_{\mathsf{pr}})$.

Input Hiding. 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 $\lambda, n_{\mathsf{pb}}, n_{\mathsf{pr}}, S, T \in \mathbb{N}$:

$$\left| \begin{array}{l} \Pr[1 \leftarrow \mathcal{A}^{\mathsf{Prove}(\mathsf{crs},\cdot,\cdot)}(\mathsf{crs}) \ : \ \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, n_{\mathsf{pb}}, 1^{n_{\mathsf{pr}}}, S, T)] \\ -\Pr[1 \leftarrow \mathcal{A}^{\mathcal{O}^{\mathcal{S}}(\cdot,\cdot)}(\mathsf{crs}) \ : \ \mathsf{crs} \leftarrow \mathcal{S}(1^\lambda, n_{\mathsf{pb}}, 1^{n_{\mathsf{pr}}}, S, T)] \end{array} \right| \leq \mathsf{negl}(\lambda)$$

where $\mathcal{O}^{\mathcal{S}}(x_{\mathsf{pb}}, x_{\mathsf{pr}})$ outputs $\mathcal{S}(x_{\mathsf{pb}})$ if $\mathcal{M}(x_{\mathsf{pb}}, x_{\mathsf{pr}}) = 1$ and $\perp$ otherwise.

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

$$\Pr\left[\begin{array}{l} \mathsf{Verify}(\mathsf{crs}, \mathsf{h}_{\mathsf{pb}}, b, \pi) = 1, \\ \wedge\ \forall x_{\mathsf{pr}} \ : \ (x_{\mathsf{pb}}, x_{\mathsf{pr}}) \notin \mathcal{L}_{\mathcal{M},T} \end{array} \ : \ \begin{array}{l} (n_{\mathsf{pb}}, 1^{n_{\mathsf{pr}}}, S, T) \leftarrow \mathcal{A}(1^\lambda), \\ \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, n_{\mathsf{pb}}, 1^{n_{\mathsf{pr}}}, S, T), \\ (x_{\mathsf{pb}}, b, \pi) \leftarrow \mathcal{A}(\mathsf{crs}), \\ \mathsf{h}_{\mathsf{pb}} = \mathsf{HT.Hash}(\mathsf{ht.hk}, x_{\mathsf{pb}}) \end{array}\right] \leq \mathsf{negl}(\lambda) \ .$$

We remark that the above soundness property is implied by the straight-line extraction property.

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

$$\Pr\left[\begin{array}{l} \mathsf{Verify}(\mathsf{crs}, \mathsf{h}_{\mathsf{pb}}, b, \pi) = 1 \\ \wedge\ (x_{\mathsf{pb}}, x^*_{\mathsf{pr}}) \notin \mathcal{L}_{\mathcal{M},T} \end{array} \ : \ \begin{array}{l} (n_{\mathsf{pb}}, 1^{n_{\mathsf{pr}}}, S, T) \leftarrow \mathcal{A}(1^\lambda), \\ (\mathsf{crs}, \mathsf{td}) \leftarrow \mathcal{E}(1^\lambda, n_{\mathsf{pb}}, 1^{n_{\mathsf{pr}}}, S, T), \\ (x_{\mathsf{pb}}, b, \pi) \leftarrow \mathcal{A}(\mathsf{crs}), \\ \mathsf{h}_{\mathsf{pb}} = \mathsf{HT.Hash}(\mathsf{ht.hk}, x_{\mathsf{pb}}), \\ x^*_{\mathsf{pr}} \leftarrow \mathcal{E}(\mathsf{td}, \pi) \end{array}\right] \leq \mathsf{negl}(\lambda) \ .$$

such that $\mathsf{crs}_1 \leftarrow \mathcal{E}(1^\lambda, n_{\mathsf{pb}}, 1^{n_{\mathsf{pr}}}, S, T)$ and $\mathsf{crs}_2 \leftarrow \mathsf{Setup}(1^\lambda, n_{\mathsf{pb}}, 1^{n_{\mathsf{pr}}}, S, T)$ are indistinguishable.

4.2 Reusable Encrypted RAM Delegation

Syntax. A reusable encrypted RAM delegation scheme for a RAM machine $\mathcal{M}$ consists of the following PPT algorithms:

$\mathsf{Setup}(1^\lambda, n_{\mathsf{pb}}, 1^{n_{\mathsf{pr}}}, S, T) \to \mathsf{crs}$**.** This is the same as non-reusable encrypted RAM delegation.

$\mathsf{PrivEnc}(\mathsf{crs}, x_{\mathsf{pr}}) \to (\mathsf{eh}_{\mathsf{pr}}, \pi_{\mathsf{enc}}, \mathsf{aux})$**.** The private-input encoding algorithm takes as input the common reference string crs and a private input value x_{pr}. It outputs an encrypted digest $\mathsf{eh}_{\mathsf{pr}}$, a proof π_{enc} and some auxiliary value aux.

$\mathsf{Prove}(\mathsf{crs}, x_{\mathsf{pb}}, \mathsf{aux}) \to (b, \pi_{\mathsf{edel}})$**.** The proving algorithm takes as input the common reference string crs, a public input value x_{pb} and an auxiliary value aux. It outputs a bit b and a proof π_{edel}.

$\mathsf{VerifyEnc}(\mathsf{crs}, \mathsf{eh}_{\mathsf{pr}}, \pi_{\mathsf{enc}}) \to 0/1$**.** The encoding verification algorithm takes as input the common reference string crs, an encrypted digest $\mathsf{eh}_{\mathsf{pr}}$ and a proof π_{enc}. It outputs 0 (reject) or 1 (accept).

$\mathsf{Verify}(\mathsf{crs}, \mathsf{h}_{\mathsf{pb}}, \mathsf{eh}_{\mathsf{pr}}, b, \pi_{\mathsf{edel}}) \to 0/1$**.** The verification algorithm takes as input the common reference string crs, a digest h_{pb} of the public input, an encrypted digest $\mathsf{eh}_{\mathsf{pr}}$ of the private input, a bit b, and a proof π_{edel}. It outputs 0 (reject) or 1 (accept).

Definition 4.2 (Reusable Encrypted RAM Delegation)**.** A reusable encrypted RAM delegation scheme $(\mathsf{Setup}, \mathsf{PrivEnc}, \mathsf{Prove}, \mathsf{VerifyEnc}, \mathsf{Verify})$ for $\mathcal{M}$ associated with a hash tree HT with hash key $\mathsf{ht.hk}$ has to satisfy the following property:

Completeness. For every $\lambda, T, S, n_{\mathsf{pb}}, n_{\mathsf{pr}} \in \mathbb{N}$ s.t. $T, S, n_{\mathsf{pb}} \leq 2^\lambda$, $n_{\mathsf{pr}} = \mathsf{poly}(\lambda)$, $x_{\mathsf{pb}} \in \{0,1\}^{n_{\mathsf{pb}}}$, $x_{\mathsf{pr}} \in \{0,1\}^{n_{\mathsf{pr}}}$, $(x_{\mathsf{pb}}, x_{\mathsf{pr}}) \in \mathcal{L}_{\mathcal{M},T}$ it holds that:

$$\Pr\left[\begin{array}{l|ll} \mathsf{VerifyEnc}(\mathsf{crs}, \mathsf{eh}_{\mathsf{pr}}, \pi_{\mathsf{enc}}) = 1 \wedge & : & \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, n_{\mathsf{pb}}, 1^{n_{\mathsf{pr}}}, S, T) \\ \mathsf{Verify}(\mathsf{crs}, \mathsf{h}_{\mathsf{pb}}, \mathsf{eh}_{\mathsf{pr}}, b, \pi_{\mathsf{edel}}) = 1 & & (\mathsf{eh}_{\mathsf{pr}}, \pi_{\mathsf{enc}}, \mathsf{aux}) \leftarrow \mathsf{PrivEnc}(\mathsf{crs}, x_{\mathsf{pr}}) \\ & & (b, \pi_{\mathsf{edel}}) \leftarrow \mathsf{Prove}(\mathsf{crs}, x_{\mathsf{pb}}, \mathsf{aux}) \end{array}\right] = 1.$$

Efficiency. For the completeness experiment above,

- The size of the crs and setup running time are $\mathsf{poly}(\lambda, \log T, \log n_{\mathsf{pb}}, n_{\mathsf{pr}})$.
- The size of π_{enc}, the private encoding and its verification time are $\mathsf{poly}(\lambda, n_{\mathsf{pr}})$.
- Prover's running time is $\mathsf{poly}(\lambda, T, n_{\mathsf{pb}}, n_{\mathsf{pr}})$.
- The size of π_{edel} and the verification time are $\mathsf{poly}(\lambda, \log T, \log n_{\mathsf{pb}}, \log n_{\mathsf{pr}})$.

Input Hiding. 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 $\lambda, n_{\mathsf{pb}}, n_{\mathsf{pr}}, S, T \in \mathbb{N}$:

$$\left|\Pr\left[1 \leftarrow \mathsf{Expt}_0^{\mathcal{C},\mathcal{A}}(1^\lambda, n_{\mathsf{pb}}, 1^{n_{\mathsf{pr}}}, S, T)\right] - \Pr\left[1 \leftarrow \mathsf{Expt}_1^{\mathcal{S},\mathcal{A}}(1^\lambda, n_{\mathsf{pb}}, 1^{n_{\mathsf{pr}}}, S, T)\right]\right| \leq \mathsf{negl}(\lambda)$$

where definitions of $\mathsf{Expt}_0^{\mathcal{C},\mathcal{A}}$ and $\mathsf{Expt}_1^{\mathcal{S},\mathcal{A}}$ are provided in Figure 2.

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

$$\Pr\left[\begin{array}{l|ll} \mathsf{VerifyEnc}(\mathsf{crs}, \mathsf{eh}_{\mathsf{pr}}, \pi_{\mathsf{enc}}) = 1 & : & (n_{\mathsf{pb}}, 1^{n_{\mathsf{pr}}}, S, T) \leftarrow \mathcal{A}(1^\lambda), \\ \wedge\ \mathsf{Verify}(\mathsf{crs}, \mathsf{h}_{\mathsf{pb}}, \mathsf{eh}_{\mathsf{pr}}, b, \pi_{\mathsf{edel}}) = 1 & & \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, n_{\mathsf{pb}}, 1^{n_{\mathsf{pr}}}, S, T), \\ \wedge\ \ \forall x_{\mathsf{pr}}\ :\ (x_{\mathsf{pb}}, x_{\mathsf{pr}}) \notin \mathcal{L}_{\mathcal{M},T} & & (x_{\mathsf{pb}}, \mathsf{eh}_{\mathsf{pr}}, \pi_{\mathsf{enc}}, b, \pi_{\mathsf{edel}}) \leftarrow \mathcal{A}(\mathsf{crs}), \\ & & \mathsf{h}_{\mathsf{pb}} = \mathsf{HT.Hash}(\mathsf{ht.hk}, x_{\mathsf{pb}}) \end{array}\right] \leq \mathsf{negl}(\lambda) \quad .$$

We remark that the above soundness property is implied by the straight-line extraction property explained further below.

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

$$\Pr\left[\begin{array}{l|ll} \mathsf{Verify}(\mathsf{crs}, \mathsf{h}_{\mathsf{pb}}, \mathsf{eh}_{\mathsf{pr}}, 0, \pi_{\mathsf{edel}}^{(0)}) = 1 & : & (n_{\mathsf{pb}}, n_{\mathsf{pr}}, S, T) \leftarrow \mathcal{A}(1^\lambda), \\ \wedge\ \mathsf{Verify}(\mathsf{crs}, \mathsf{h}_{\mathsf{pb}}, \mathsf{eh}_{\mathsf{pr}}, 1, \pi_{\mathsf{edel}}^{(1)}) = 1 & & \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, n_{\mathsf{pb}}, n_{\mathsf{pr}}, S, T), \\ & & (\mathsf{h}_{\mathsf{pb}}, \mathsf{eh}_{\mathsf{pr}}, \pi_{\mathsf{edel}}^{(0)}, \pi_{\mathsf{edel}}^{(1)}) \leftarrow \mathcal{A}(\mathsf{crs}) \end{array}\right] \leq \mathsf{negl}(\lambda) \quad .$$

$\mathsf{Expt}_0^{\mathcal{C},\mathcal{A}}(1^\lambda, n_{\mathsf{pb}}, 1^{n_{\mathsf{pr}}}, S, T)$. This is the real experiment parameterized by an honest challenger $\mathcal{C}$. $\mathcal{A}$ receives $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, n_{\mathsf{pb}}, 1^{n_{\mathsf{pr}}}, S, T)$ from $\mathcal{C}$. $\mathcal{A}$ makes the following queries in an adaptive manner. After this, $\mathcal{A}$ outputs guess b'. Output b'.

- PRIVATE INPUT ENCODING: $\mathcal{A}$ sends x_{pr} and receives $(\mathsf{eh}_{\mathsf{pr}}, \pi_{\mathsf{enc}}, i)$ where $(\mathsf{eh}_{\mathsf{pr}}, \pi_{\mathsf{enc}}, \mathsf{aux}_i) \leftarrow \mathsf{PrivEnc}(\mathsf{crs}, x_{\mathsf{pr}})$ and i is an index that after each query of this type is incremented.
- PROVE: $\mathcal{A}$ sends (x_{pb}, i) and receives π_{edel} where $(b, \pi_{\mathsf{edel}}) \leftarrow \mathsf{Prove}(\mathsf{crs}, x_{\mathsf{pb}}, \mathsf{aux}_i)$.

$\mathsf{Expt}_1^{\mathcal{S},\mathcal{A}}(1^\lambda, n_{\mathsf{pb}}, 1^{n_{\mathsf{pr}}}, S, T)$. This is the ideal experiment parameterized by a stateful simulator $\mathcal{S}$. $\mathcal{A}$ receives $\mathsf{crs} \leftarrow \mathcal{S}(1^\lambda, n_{\mathsf{pb}}, 1^{n_{\mathsf{pr}}}, S, T)$ from $\mathcal{C}$. $\mathcal{A}$ makes the following queries in an adaptive manner. After this, $\mathcal{A}$ outputs guess b'. Output b'.

- PRIVATE INPUT ENCODING: $\mathcal{A}$ sends x_{pr} and receives $(\mathsf{eh}_{\mathsf{pr}}, \pi_{\mathsf{enc}}, i)$ where $(\mathsf{eh}_{\mathsf{pr}}, \pi_{\mathsf{enc}}) \leftarrow \mathcal{S}(1^{|x_{\mathsf{pr}}|}, i)$ and i is an index that after each query of this type is incremented.
- PROVE: $\mathcal{A}$ sends (x_{pb}, i), and if $(x_{\mathsf{pb}}, x_{\mathsf{pr}}) \in \mathcal{L}_{\mathcal{M},T}$ (for x_{pr} corresponding to i) receives π_{edel} where $\pi_{\mathsf{edel}} \leftarrow \mathcal{S}(x_{\mathsf{pb}}, i)$, otherwise, receives $\perp$.

Figure 2: Real and ideal experiments for Input Hiding property of reusable encrypted RAM delegation.

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

$$\Pr\left[\begin{array}{l|l} \mathsf{VerifyEnc}(\mathsf{crs}, \mathsf{eh}_{\mathsf{pr}}, \pi_{\mathsf{enc}}) = 1 & (n_{\mathsf{pb}}, 1^{n_{\mathsf{pr}}}, S, T) \leftarrow \mathcal{A}(1^\lambda), \\ \wedge\ \mathsf{Verify}(\mathsf{crs}, \mathsf{h}_{\mathsf{pb}}, \mathsf{eh}_{\mathsf{pr}}, b, \pi_{\mathsf{edel}}) = 1 & (\mathsf{crs}, \mathsf{td}) \leftarrow \mathcal{E}(1^\lambda, n_{\mathsf{pb}}, 1^{n_{\mathsf{pr}}}, S, T), \\ \wedge\ \ (x_{\mathsf{pb}}, x^*_{\mathsf{pr}}) \notin \mathcal{L}_{\mathcal{M},T} & (\mathsf{eh}_{\mathsf{pr}}, \pi_{\mathsf{enc}}) \leftarrow \mathcal{A}(\mathsf{crs}), \\ & x^*_{\mathsf{pr}} \leftarrow \mathcal{E}(\mathsf{td}, \pi_{\mathsf{enc}}), \\ & (x_{\mathsf{pb}}, b, \pi_{\mathsf{edel}}) \leftarrow \mathcal{A}(\mathsf{crs}), \\ & \mathsf{h}_{\mathsf{pb}} = \mathsf{HT.Hash}(\mathsf{ht.hk}, x_{\mathsf{pb}}) \end{array}\right] \leq \mathsf{negl}(\lambda) \quad .$$

such that $\mathsf{crs}_1 \leftarrow \mathcal{E}(1^\lambda, n_{\mathsf{pb}}, 1^{n_{\mathsf{pr}}}, S, T)$ and $\mathsf{crs}_2 \leftarrow \mathsf{Setup}(1^\lambda, n_{\mathsf{pb}}, 1^{n_{\mathsf{pr}}}, S, T)$ are indistinguishable.

5 Constructing (Reusable) Encrypted RAM Delegation

In this section, we first informally define a split memory RAM delegation scheme and then show how to combine this object with PKEs and NIZKs to construct a (reusable) encrypted RAM delegation.

Informally speaking, a split memory RAM delegation scheme allows one to split the memory of a RAM computation into several partitions, then prove the correctness of a RAM computation w.r.t. the partitioned memory and verify the proof w.r.t. independent partial digests of such partitions. Namely, the digest algorithm uses a hash tree to independently generate a digest h_i of each partition P_i and we let the final digest to be $h = (h_1, \ldots, h_n)$. Syntactically, this object is the same as a RAM

delegation scheme except that the setup algorithm takes as input a partitioning $\mathbf{P} = (P_1, \ldots, P_n)$ of the memory, and generates a crs w.r.t. this partitioning. Note that the partitioning of the memory can be determined by the partition sizes, hence, in this work we use $\mathbf{s} = (s_1, \ldots, s_n)$ to define a partitioning $\mathbf{P} = (P_1, \ldots, P_n)$ where $|P_i| = s_i$ and $P_i = [\sum_{j=1}^{i-1} s_j + 1, \ldots, \sum_{j=1}^{i} s_j]$. We refer the readers to Appendix B for a formal definition and a construction based on [CJJ22].

5.1 Encrypted RAM Delegation

Language $\mathcal{L}_{\mathsf{enc}}$
Hardwired: $\mathsf{ht.hk}, \mathsf{pke.pk}$
Instance: $x_{\mathsf{enc}} = \mathsf{eh}_{\mathsf{pr}}$
Witness: $w_{\mathsf{enc}} = (x_{\mathsf{pr}}, \mathsf{h}_{\mathsf{pr}}, r)$
Output: $(x_{\mathsf{enc}}, w_{\mathsf{enc}}) \in \mathcal{L}_{\mathsf{enc}}$ if the following hold:

- $\mathsf{h}_{\mathsf{pr}} = \mathsf{HT.Hash}(\mathsf{ht.hk}, x_{\mathsf{pr}})$.
- $\mathsf{eh}_{\mathsf{pr}} = \mathsf{PKE.Enc}(\mathsf{pke.pk}, \mathsf{h}_{\mathsf{pr}}; r)$.

Language $\mathcal{L}_{\mathsf{edel}}$
Hardwired: $\mathsf{ht.hk}, \mathsf{pke.pk}, \mathsf{del.crs}$
Instance: $x_{\mathsf{edel}} = (\mathsf{h}_{\mathsf{pb}}, \mathsf{eh}_{\mathsf{pr}}, \mathsf{h}_{\Sigma}, b)$
Witness: $w_{\mathsf{edel}} = (\mathsf{h}_{\mathsf{pr}}, r, \mathsf{del}.\pi)$
Output: $(x_{\mathsf{edel}}, w_{\mathsf{edel}}) \in \mathcal{L}_{\mathsf{edel}}$ if the following hold:

- $\mathsf{eh}_{\mathsf{pr}} = \mathsf{PKE.Enc}(\mathsf{pke.pk}, \mathsf{h}_{\mathsf{pr}}; r)$.
- $\mathsf{Del.Verify}(\mathsf{del.crs}, h, b, \mathsf{del}.\pi) = 1$, where $h = (\mathsf{h}_{\mathsf{pb}}, \mathsf{h}_{\mathsf{pr}}, \mathsf{h}_{\Sigma})$.

Parameters. In the following construction we use $n_{\mathsf{enc}} = |\mathsf{eh}_{\mathsf{pr}}| = |\mathsf{PKE.Enc}(\mathsf{pke.pk}, 1^\lambda)| = \mathsf{poly}(\lambda)$, $n'_{\mathsf{enc}} = |x_{\mathsf{pr}}| + |\mathsf{h}_{\mathsf{pr}}| + |r| = n_{\mathsf{pr}} + \mathsf{poly}(\lambda)$, $n_{\mathsf{edel}} = |\mathsf{h}_{\mathsf{pb}}| + |\mathsf{eh}_{\mathsf{pr}}| + |\mathsf{h}_{\Sigma}| = \mathsf{poly}(\lambda)$, and $n'_{\mathsf{edel}} = |\mathsf{h}_{\mathsf{pr}}| + |r| + |\mathsf{del}.\pi| = \mathsf{poly}(\lambda)$.

Construction 5.1. [Encrypted RAM Delegation] Let $\mathsf{PKE} = (\mathsf{PKE.Gen}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$ be a public-key encryption scheme, $\mathsf{NIZK} = (\mathsf{NIZK.Setup}, \mathsf{NIZK.Prove}, \mathsf{NIZK.Verify})$ be a non-interactive zero-knowledge scheme, and $\mathsf{Del} = (\mathsf{Del.Setup}, \mathsf{Del.Digest}, \mathsf{Del.Prove}, \mathsf{Del.Verify})$ be a split memory RAM delegation scheme associated with a hash tree HT with hash key $\mathsf{ht.hk}$. We construct our encrypted RAM delegation scheme $\mathcal{E}\mathsf{Del} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ as follows:

$\mathsf{Setup}(1^\lambda, n_{\mathsf{pb}}, 1^{n_{\mathsf{pr}}}, S, T) \to \mathsf{crs}$. Let $\mathbf{s} = (n_{\mathsf{pb}}, n_{\mathsf{pr}}, S)$ be a partitioning of the memory. Sample a split memory RAM delegation scheme $\mathsf{del.crs} \leftarrow \mathsf{Del.Setup}(1^\lambda, T, \mathbf{s})$, a PKE scheme $(\mathsf{pke.pk}, \mathsf{pke.sk}) \leftarrow \mathsf{PKE.Gen}(1^\lambda)$, a NIZK scheme for $\mathcal{L}_{\mathsf{enc}}$ as $(\mathsf{zk.crs}_{\mathsf{enc}}, \mathsf{zk.td}_{\mathsf{enc}}) \leftarrow \mathsf{NIZK.Setup}(1^\lambda, 1^{n_{\mathsf{enc}}}, 1^{n'_{\mathsf{enc}}})$, and finally another NIZK scheme for $\mathcal{L}_{\mathsf{edel}}$ as $(\mathsf{zk.crs}_{\mathsf{edel}}, \mathsf{zk.td}_{\mathsf{edel}}) \leftarrow \mathsf{NIZK.Setup}(1^\lambda, 1^{n_{\mathsf{edel}}}, 1^{n'_{\mathsf{edel}}})$. Let $\mathsf{crs} = (\mathsf{del.crs}, \mathsf{pke.pk}, \mathsf{zk.crs}_{\mathsf{enc}}, \mathsf{zk.crs}_{\mathsf{edel}}, S)$.

$\mathsf{Prove}(\mathsf{crs}, x_{\mathsf{pb}}, x_{\mathsf{pr}}) \to (b, \pi)$. This poly-time algorithm does the following:

1. Parse $\mathsf{crs} = (\mathsf{del.crs}, \mathsf{pke.pk}, \mathsf{zk.crs}_{\mathsf{enc}}, \mathsf{zk.crs}_{\mathsf{edel}}, S)$.
2. Compute a hash digest of the private input $\mathsf{h}_{\mathsf{pr}} \leftarrow \mathsf{HT.Hash}(\mathsf{ht.hk}, x_{\mathsf{pr}})$, sample a randomness r, and compute an encryption of h_{pr} as $\mathsf{eh}_{\mathsf{pr}} \leftarrow \mathsf{PKE.Enc}(\mathsf{pke.pk}, \mathsf{h}_{\mathsf{pr}}; r)$.
3. Compute a NIZK proof for validity of $\mathsf{eh}_{\mathsf{pr}}$ as $\mathsf{zk}.\pi_{\mathsf{enc}} \leftarrow \mathsf{NIZK.Prove}(\mathsf{zk.crs}_{\mathsf{enc}}, \mathsf{eh}_{\mathsf{pr}}, (x_{\mathsf{pr}}, \mathsf{h}_{\mathsf{pr}}, r))$.
4. Let $\mathsf{cf}_0 = (x_{\mathsf{pb}}, x_{\mathsf{pr}}, 0^S)$ be the initial memory of the RAM machine and compute its partial digests $\mathsf{h}_{\mathsf{pb}} \leftarrow \mathsf{HT.Hash}(\mathsf{ht.hk}, x_{\mathsf{pb}})$ and $\mathsf{h}_{\Sigma} \leftarrow \mathsf{HT.Hash}(\mathsf{ht.hk}, 0^S)$.
5. Compute a split memory RAM delegation proof $(b, \mathsf{del}.\pi) \leftarrow \mathsf{Del.Prove}(\mathsf{del.crs}, \mathsf{cf}_0)$.
6. Compute a NIZK proof $\mathsf{zk}.\pi_{\mathsf{edel}} \leftarrow \mathsf{NIZK.Prove}(\mathsf{zk.crs}_{\mathsf{edel}}, (\mathsf{h}_{\mathsf{pb}}, \mathsf{eh}_{\mathsf{pr}}, \mathsf{h}_{\Sigma}, b), (\mathsf{h}_{\mathsf{pr}}, r, \mathsf{del}.\pi))$.

7. Output $(b, \pi = (\mathsf{eh}_{\mathsf{pr}}, \mathsf{zk}.\pi_{\mathsf{enc}}, \mathsf{zk}.\pi_{\mathsf{edel}}))$.

$\mathsf{Verify}(\mathsf{crs}, \mathsf{h}_{\mathsf{pb}}, b, \pi) \to 0/1$. Parse crs as above and compute $\mathsf{h}_\Sigma \leftarrow \mathsf{HT.Hash}(\mathsf{ht.hk}, 0^S)$. Output 1 if $\mathsf{NIZK.Verify}(\mathsf{zk.crs}_{\mathsf{enc}}, \mathsf{eh}_{\mathsf{pr}}, \mathsf{zk}.\pi_{\mathsf{enc}}) = 1$ and $\mathsf{NIZK.Verify}(\mathsf{zk.crs}_{\mathsf{edel}}, (\mathsf{h}_{\mathsf{pb}}, \mathsf{eh}_{\mathsf{pr}}, \mathsf{h}_\Sigma, b), \mathsf{zk}.\pi_{\mathsf{edel}}) = 1$; otherwise output 0.

Theorem 5.2. Assuming split memory RAM delegations, PKEs, and NIZKs, the Construction 5.1 is an encrypted RAM delegation (Definition 4.1) with strong soundness, straight-line extractor, and input hiding.

Completeness. The completeness follows by the construction and the completeness of the underlying Del and NIZK and the correctness of the underlying PKE.

Efficiency. The crs consists of $(\mathsf{del.crs}, \mathsf{pke.pk}, \mathsf{zk.crs}_{\mathsf{enc}}, \mathsf{zk.crs}_{\mathsf{edel}}, S)$, and the followings hold (1) Del setup time and crs size are bounded by $\mathsf{poly}(\lambda, \log T, \log n_{\mathsf{pb}}, \log n_{\mathsf{pr}})$, (2) PKE key generation time and pk size are bounded by $\mathsf{poly}(\lambda)$, (3) NIZK setup time and crs size are bounded by (a) $\mathsf{poly}(\lambda, n_{\mathsf{pr}})$ for $\mathcal{L}_{\mathsf{enc}}$, and (b) $\mathsf{poly}(\lambda, \log T, \log n_{\mathsf{pb}}, \log n_{\mathsf{pr}})$ for $\mathcal{L}_{\mathsf{edel}}$. Thus the crs size and setup time of our construction is bounded by $\mathsf{poly}(\lambda, \log T, \log n_{\mathsf{pb}}, \log n_{\mathsf{pr}}) + \mathsf{poly}(\lambda, n_{\mathsf{pr}})$.

The h_{pb} (resp. h_{pr}) size is λ (resp. λ), and the generation and verification time is $\mathsf{poly}(\lambda, n_{\mathsf{pb}})$ (resp. $\mathsf{poly}(\lambda, n_{\mathsf{pr}})$).

Given h_{pr}, the $\mathsf{eh}_{\mathsf{pr}}$ size, generation time and verification time are all $\mathsf{poly}(\lambda)$

The $\mathsf{zk}.\pi_{\mathsf{enc}}$ size, its generation and verification time all depend on the NIZK for $\mathcal{L}_{\mathsf{enc}}$, which depends on $\mathsf{eh}_{\mathsf{pr}}$ and h_{pr}, hence are bounded by $\mathsf{poly}(\lambda, n_{\mathsf{pr}})$.

The $\mathsf{zk}.\pi_{\mathsf{edel}}$ size, its generation and verification time all depend on the NIZK for $\mathcal{L}_{\mathsf{edel}}$, which is in return is dependent on the Del scheme. Note that he $\mathsf{del}.\pi$ size and verification time are $\mathsf{poly}(\lambda, \log T, \log n_{\mathsf{pb}}, \log n_{\mathsf{pr}})$, and generation time is $\mathsf{poly}(\lambda, T, n_{\mathsf{pb}}, n_{\mathsf{pr}})$. Hence the same holds in our construction.

Hence, in our construction, the proof size and the verification time are $\mathsf{poly}(\lambda, \log T, \log n_{\mathsf{pb}}, \log n_{\mathsf{pr}}) + \mathsf{poly}(\lambda, n_{\mathsf{pr}})$ and the generation time is $\mathsf{poly}(\lambda, T, n_{\mathsf{pb}}, n_{\mathsf{pr}})$

Note that since we have $T \leq 2^\lambda$ and $n_{\mathsf{pb}} + n_{\mathsf{pr}} \leq T$, we can get rid of all the log terms in our analysis.

5.1.1 Security Analysis.

Here we prove the straight-line extractor and input hiding properties of our construction.

Proof of Straight-Line Extractor. Let the extractor $\mathcal{E}$ do the following – (1) to generate $(\mathsf{crs}, \mathsf{td})$ proceed the same as normal setup except that use $\mathsf{NIZK}.\mathcal{E}_1$ to sample $(\mathsf{zk.crs}_{\mathsf{enc}}, \mathsf{zk.td}_{\mathsf{enc}})$ and $(\mathsf{zk.crs}_{\mathsf{edel}}, \mathsf{zk.td}_{\mathsf{edel}})$, then construct crs similar to setup and let $\mathsf{td} = (\mathsf{pke.sk}, \mathsf{zk.td}_{\mathsf{enc}}, \mathsf{zk.td}_{\mathsf{edel}})$, (2) upon receiving $\pi = (\mathsf{eh}_{\mathsf{pr}}, \mathsf{zk}.\pi_{\mathsf{enc}}, \mathsf{zk}.\pi_{\mathsf{edel}})$ extract $(x^*_{\mathsf{pr}}, \mathsf{h}^*_{\mathsf{pr},1}, r^*_1) = \mathsf{NIZK}.\mathcal{E}_1(\mathsf{zk.td}_{\mathsf{enc}}, \mathsf{eh}_{\mathsf{pr}}, \mathsf{zk}.\pi_{\mathsf{enc}})$ and output x^*_{pr}.

First note that the crs indistinguishability is directly implied by the crs indistinguishability of the undelying NIZK schemes.

Now define the experiment $\mathsf{Exp}_{\mathcal{E}\mathsf{Del}}$ as follows – the adversary $\mathcal{A}$ on input 1^λ sends $(n_{\mathsf{pb}}, 1^{n_{\mathsf{pr}}}, S, T)$ to the challenger $\mathcal{C}$, $\mathcal{C}$ runs $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathcal{E}(1^\lambda, n_{\mathsf{pb}}, 1^{n_{\mathsf{pr}}}, S, T)$ and sends crs to $\mathcal{A}$, then $\mathcal{A}$ outputs $(x_{\mathsf{pb}}, b, \pi)$. Parse $\pi = (\mathsf{eh}_{\mathsf{pr}}, \mathsf{zk}.\pi_{\mathsf{enc}}, \mathsf{zk}.\pi_{\mathsf{edel}})$, and let $\mathsf{h}_{\mathsf{pb}} = \mathsf{HT.Hash}(\mathsf{ht.hk}, x_{\mathsf{pb}})$, and $\mathsf{h}_\Sigma \leftarrow$

$\mathsf{HT.Hash}(\mathsf{ht.hk}, 0^S)$. Then extract the values $(x^*_{\mathsf{pr}}, \mathsf{h}^*_{\mathsf{pr},1}, r^*_1) = \mathsf{NIZK}.\mathcal{E}_1(\mathsf{zk.td}_{\mathsf{enc}}, \mathsf{eh}_{\mathsf{pr}}, \mathsf{zk}.\pi_{\mathsf{enc}})$, and $(\mathsf{h}^*_{\mathsf{pr},2}, r^*_2, \mathsf{del}.\pi^*) = \mathsf{NIZK}.\mathcal{E}_2(\mathsf{zk.td}_{\mathsf{edel}}, (\mathsf{h}_{\mathsf{pb}}, \mathsf{eh}_{\mathsf{pr}}, \mathsf{h}_\Sigma, b), \mathsf{zk}.\pi_{\mathsf{edel}})$. Namely:

$$\mathsf{Exp}_{\mathcal{E}\mathsf{Del}} = \left[\begin{array}{l} (n_{\mathsf{pb}}, 1^{n_{\mathsf{pr}}}, S, T) \leftarrow \mathcal{A}(1^\lambda), \\ (\mathsf{crs}, \mathsf{td}) \leftarrow \mathcal{E}(1^\lambda, n_{\mathsf{pb}}, 1^{n_{\mathsf{pr}}}, S, T), \\ (x_{\mathsf{pb}}, b, \pi) \leftarrow \mathcal{A}(\mathsf{crs}), \\ \mathsf{h}_{\mathsf{pb}} = \mathsf{HT.Hash}(\mathsf{ht.hk}, x_{\mathsf{pb}}), \\ \mathsf{h}_\Sigma \leftarrow \mathsf{HT.Hash}(\mathsf{ht.hk}, 0^S), \\ \text{Let } \pi = (\mathsf{eh}_{\mathsf{pr}}, \mathsf{zk}.\pi_{\mathsf{enc}}, \mathsf{zk}.\pi_{\mathsf{edel}}), \\ (x^*_{\mathsf{pr}}, \mathsf{h}^*_{\mathsf{pr},1}, r^*_1) \leftarrow \mathsf{NIZK}.\mathcal{E}_1(\mathsf{zk.td}_{\mathsf{enc}}, \mathsf{eh}_{\mathsf{pr}}, \mathsf{zk}.\pi_{\mathsf{enc}}), \\ (\mathsf{h}^*_{\mathsf{pr},2}, r^*_2, \mathsf{del}.\pi^*) = \mathsf{NIZK}.\mathcal{E}_2(\mathsf{zk.td}_{\mathsf{edel}}, (\mathsf{h}_{\mathsf{pb}}, \mathsf{eh}_{\mathsf{pr}}, \mathsf{h}_\Sigma, b), \mathsf{zk}.\pi_{\mathsf{edel}}) \end{array}\right]$$

For an adversary $\mathcal{A}$ we define the output of the above experiment, $\mathsf{Exp}_{\mathcal{E}\mathsf{Del}}(\mathcal{A})$, to be 1 if it holds that $\mathsf{NIZK.Verify}(\mathsf{zk.crs}_{\mathsf{enc}}, \mathsf{eh}_{\mathsf{pr}}, \mathsf{zk}.\pi_{\mathsf{enc}}) = 1$ and $\mathsf{NIZK.Verify}(\mathsf{zk.crs}_{\mathsf{edel}}, (\mathsf{h}_{\mathsf{pb}}, \mathsf{eh}_{\mathsf{pr}}, \mathsf{h}_\Sigma, b), \mathsf{zk}.\pi_{\mathsf{edel}}) = 1$ and $(x_{\mathsf{pb}}, x^*_{\mathsf{pr}}) \notin \mathcal{L}_{\mathcal{M},T}$. Namely $\mathcal{A}$ wins if:

$$\left[\begin{array}{ll} (x_{\mathsf{pb}}, x^*_{\mathsf{pr}}) \notin \mathcal{L}_{\mathcal{M},T} \wedge & \\ \mathsf{NIZK.Verify}(\mathsf{zk.crs}_{\mathsf{enc}}, \mathsf{eh}_{\mathsf{pr}}, \mathsf{zk}.\pi_{\mathsf{enc}}) = 1 \wedge & : \ \mathsf{Exp}_{\mathcal{E}\mathsf{Del}} \\ \mathsf{NIZK.Verify}(\mathsf{zk.crs}_{\mathsf{edel}}, (\mathsf{h}_{\mathsf{pb}}, \mathsf{eh}_{\mathsf{pr}}, \mathsf{h}_\Sigma, b), \mathsf{zk}.\pi_{\mathsf{edel}}) = 1 & \end{array}\right]$$

Let $(x_{\mathsf{enc}}, w_{\mathsf{enc}}) = (\mathsf{eh}_{\mathsf{pr}}, (x^*_{\mathsf{pr}}, \mathsf{h}^*_{\mathsf{pr},1}, r^*_1))$ and $(x_{\mathsf{edel}}, w_{\mathsf{edel}}) = ((\mathsf{h}_{\mathsf{pb}}, \mathsf{eh}_{\mathsf{pr}}, \mathsf{h}_\Sigma, b), (\mathsf{h}^*_{\mathsf{pr},2}, r^*_2, \mathsf{del}.\pi^*))$. Define the following variables:

$$\epsilon_1 = \Pr[\mathsf{Exp}_{\mathcal{E}\mathsf{Del}}(\mathcal{A}) = 1] - \Pr\left[\begin{array}{ll} (x_{\mathsf{pb}}, x^*_{\mathsf{pr}}) \notin \mathcal{L}_{\mathcal{M},T} \ \wedge \ (x_{\mathsf{enc}}, w_{\mathsf{enc}}) \in \mathcal{L}_{\mathsf{enc}} \wedge & \\ \mathsf{NIZK.Verify}(\mathsf{zk.crs}_{\mathsf{edel}}, x_{\mathsf{edel}}, \mathsf{zk}.\pi_{\mathsf{edel}}) = 1 & : \ \mathsf{Exp}_{\mathcal{E}\mathsf{Del}} \end{array}\right]. \quad (3)$$

$$\epsilon_2 = \Pr\left[\begin{array}{ll} (x_{\mathsf{pb}}, x^*_{\mathsf{pr}}) \notin \mathcal{L}_{\mathcal{M},T} \ \wedge \ (x_{\mathsf{enc}}, w_{\mathsf{enc}}) \in \mathcal{L}_{\mathsf{enc}} \wedge & \\ \mathsf{NIZK.Verify}(\mathsf{zk.crs}_{\mathsf{edel}}, x_{\mathsf{edel}}, \mathsf{zk}.\pi_{\mathsf{edel}}) = 1 & : \ \mathsf{Exp}_{\mathcal{E}\mathsf{Del}} \end{array}\right] -$$
$$\Pr\left[\begin{array}{ll} (x_{\mathsf{pb}}, x^*_{\mathsf{pr}}) \notin \mathcal{L}_{\mathcal{M},T} \ \wedge \ (x_{\mathsf{enc}}, w_{\mathsf{enc}}) \in \mathcal{L}_{\mathsf{enc}} \ \wedge (x_{\mathsf{edel}}, w_{\mathsf{edel}}) \in \mathcal{L}_{\mathsf{edel}} & : \ \mathsf{Exp}_{\mathcal{E}\mathsf{Del}} \end{array}\right]. \quad (4)$$

$$\epsilon_3 = \Pr\left[\begin{array}{ll} (x_{\mathsf{pb}}, x^*_{\mathsf{pr}}) \notin \mathcal{L}_{\mathcal{M},T} \ \wedge \ (x_{\mathsf{enc}}, w_{\mathsf{enc}}) \in \mathcal{L}_{\mathsf{enc}} \ \wedge (x_{\mathsf{edel}}, w_{\mathsf{edel}}) \in \mathcal{L}_{\mathsf{edel}} & : \ \mathsf{Exp}_{\mathcal{E}\mathsf{Del}} \end{array}\right] -$$
$$\Pr\left[\begin{array}{ll} (x_{\mathsf{pb}}, x^*_{\mathsf{pr}}) \notin \mathcal{L}_{\mathcal{M},T} \ \wedge \ \mathsf{h}^*_{\mathsf{pr},2} = \mathsf{HT.Hash}(\mathsf{ht.hk}, x^*_{\mathsf{pr}}) & \\ \mathsf{Del.Verify}(\mathsf{del.crs}, (\mathsf{h}_{\mathsf{pb}}, \mathsf{h}^*_{\mathsf{pr},2}, \mathsf{h}_\Sigma), b, \mathsf{del}.\pi) = 1 & : \ \mathsf{Exp}_{\mathcal{E}\mathsf{Del}} \end{array}\right]. \quad (5)$$

$$\epsilon_4 = \Pr\left[\begin{array}{ll} (x_{\mathsf{pb}}, x^*_{\mathsf{pr}}) \notin \mathcal{L}_{\mathcal{M},T} \ \wedge \ \mathsf{h}^*_{\mathsf{pr},2} = \mathsf{HT.Hash}(\mathsf{ht.hk}, x^*_{\mathsf{pr}}) & \\ \mathsf{Del.Verify}(\mathsf{del.crs}, (\mathsf{h}_{\mathsf{pb}}, \mathsf{h}^*_{\mathsf{pr},2}, \mathsf{h}_\Sigma), b, \mathsf{del}.\pi) = 1 & : \ \mathsf{Exp}_{\mathcal{E}\mathsf{Del}} \end{array}\right]. \quad (6)$$

Now since $\Pr[\mathsf{Exp}_{\mathcal{E}\mathsf{Del}}(\mathcal{A}) = 1] = \epsilon_1 + \epsilon_2 + \epsilon_3 + \epsilon_4$ we only need to show that for $i \in [4]$, ϵ_i is negligible.

Case 1. Suppose towards the contradiction that ϵ_1 is non-negligible, then by Eq. (3) it holds that:

$$\Pr[\mathsf{NIZK.Verify}(\mathsf{zk.crs_{enc}}, \mathsf{eh_{pr}}, \mathsf{zk}.\pi_{\mathsf{enc}}) = 1 \wedge (x_{\mathsf{enc}}, w_{\mathsf{enc}}) \notin \mathcal{L}_{\mathsf{enc}}] \geq \epsilon_1$$

Now we construct adversary $\mathcal{B}^{\mathcal{E}}_{\mathsf{zk}}$ that breaks the straight-line extractor property of the underlying NIZK scheme as follows:

1. Receive $(n_{\mathsf{pb}}, 1^{n_{\mathsf{pr}}}, S, T)$ from $\mathcal{A}$.
2. Receive $\mathsf{zk.crs}^*_{\mathsf{enc}}$ from $\mathcal{C}$.
3. Construct crs similar to experiment $\mathsf{Exp}_{\mathcal{E}\mathsf{Del}}$ where $\mathsf{zk.crs_{enc}} = \mathsf{zk.crs}^*_{\mathsf{enc}}$, and send it to $\mathcal{A}$.
4. Receive $(x_{\mathsf{pb}}, b, \pi)$ from $\mathcal{A}$.
5. Parse $\pi = (\mathsf{eh_{pr}}, \mathsf{zk}.\pi_{\mathsf{enc}}, \mathsf{zk}.\pi_{\mathsf{edel}})$.
6. Send $(\mathsf{eh_{pr}}, \mathsf{zk}.\pi_{\mathsf{enc}})$ to $\mathcal{C}$.

Note that $\mathcal{B}^{\mathcal{E}}_{\mathsf{zk}}$ perfectly simulates the experiment $\mathsf{Exp}_{\mathcal{E}\mathsf{Del}}$ for $\mathcal{A}$, thus has the same advantage ϵ_1 in breaking the straight-line extractor property of the underlying NIZK scheme.

Case 2. Suppose towards the contradiction that ϵ_2 is non-negligible, then by Eq. (4) it holds that:

$$\Pr[\mathsf{NIZK.Verify}(\mathsf{zk.crs_{edel}}, x_{\mathsf{edel}}, \mathsf{zk}.\pi_{\mathsf{edel}}) = 1 \wedge (x_{\mathsf{edel}}, w_{\mathsf{edel}}) \notin \mathcal{L}_{\mathsf{edel}}] \geq \epsilon_2$$

Now we construct adversary $\mathcal{B}^{\mathcal{E}}_{\mathsf{zk}}$ that breaks the straight-line extractor property of the underlying NIZK scheme as follows:

1. Receive $(n_{\mathsf{pb}}, 1^{n_{\mathsf{pr}}}, S, T)$ from $\mathcal{A}$.
2. Receive $\mathsf{zk.crs}^*_{\mathsf{edel}}$ from $\mathcal{C}$.
3. Construct crs similar to experiment $\mathsf{Exp}_{\mathcal{E}\mathsf{Del}}$ where $\mathsf{zk.crs_{edel}} = \mathsf{zk.crs}^*_{\mathsf{edel}}$, and send it to $\mathcal{A}$.
4. Receive $(x_{\mathsf{pb}}, b, \pi)$ from $\mathcal{A}$.
5. Parse $\pi = (\mathsf{eh_{pr}}, \mathsf{zk}.\pi_{\mathsf{enc}}, \mathsf{zk}.\pi_{\mathsf{edel}})$.
6. Compute $\mathsf{h_{pb}} = \mathsf{HT.Hash}(\mathsf{ht.hk}, x_{\mathsf{pb}})$ and $\mathsf{h}_\Sigma \leftarrow \mathsf{HT.Hash}(\mathsf{ht.hk}, 0^S)$.
7. Send $((\mathsf{h_{pb}}, \mathsf{eh_{pr}}, \mathsf{h}_\Sigma, b), \mathsf{zk}.\pi_{\mathsf{edel}})$ to $\mathcal{C}$.

Note that $\mathcal{B}^{\mathcal{E}}_{\mathsf{zk}}$ perfectly simulates the experiment $\mathsf{Exp}_{\mathcal{E}\mathsf{Del}}$ for $\mathcal{A}$, thus has the same advantage ϵ_2 in breaking the straight-line extractor property of the underlying NIZK scheme.

Case 3. Suppose towards the contradiction that ϵ_1 is non-negligible. First note that $(x_{\mathsf{enc}}, w_{\mathsf{enc}}) \in \mathcal{L}_{\mathsf{enc}}$ is equivalent with:

$$\mathsf{eh}_{\mathsf{pr}} = \mathsf{PKE.Enc}(\mathsf{pke.pk}, \mathsf{h}^*_{\mathsf{pr},1}; r^*_1) \quad \wedge \quad \mathsf{h}^*_{\mathsf{pr},1} = \mathsf{HT.Hash}(\mathsf{ht.hk}, x^*_{\mathsf{pr}})$$

and $(x_{\mathsf{edel}}, w_{\mathsf{edel}}) \in \mathcal{L}_{\mathsf{edel}}$ is equivalent with:

$$\mathsf{eh}_{\mathsf{pr}} = \mathsf{PKE.Enc}(\mathsf{pke.pk}, \mathsf{h}^*_{\mathsf{pr},2}; r^*_2) \quad \wedge \quad \mathsf{Del.Verify}(\mathsf{del.crs}, (\mathsf{h}_{\mathsf{pb}}, \mathsf{h}^*_{\mathsf{pr},2}, \mathsf{h}_\Sigma), b, \mathsf{del}.\pi) = 1$$

then by Eq. (5) it holds that

$$\Pr\left[\begin{array}{l} \mathsf{eh}_{\mathsf{pr}} = \mathsf{PKE.Enc}(\mathsf{pke.pk}, \mathsf{h}^*_{\mathsf{pr},1}; r^*_1) \wedge \mathsf{h}^*_{\mathsf{pr},1} = \mathsf{HT.Hash}(\mathsf{ht.hk}, x^*_{\mathsf{pr}}) \wedge \\ \mathsf{eh}_{\mathsf{pr}} = \mathsf{PKE.Enc}(\mathsf{pke.pk}, \mathsf{h}^*_{\mathsf{pr},2}; r^*_2) \wedge \mathsf{h}^*_{\mathsf{pr},2} \neq \mathsf{HT.Hash}(\mathsf{ht.hk}, x^*_{\mathsf{pr}}) \end{array} : \mathsf{Exp}_{\mathcal{E}\mathsf{Del}}\right] \geq \epsilon_3$$

Note that this implies that $\mathsf{h}^*_{\mathsf{pr},1} \neq \mathsf{h}^*_{\mathsf{pr},2}$, however, $\mathsf{PKE.Enc}(\mathsf{pke.pk}, \mathsf{h}^*_{\mathsf{pr},1}; r^*_1) = \mathsf{PKE.Enc}(\mathsf{pke.pk}, \mathsf{h}^*_{\mathsf{pr},2}; r^*_2)$. This breaks the perfect correctness of the underlying PKE scheme.

Case 4. Suppose towards the contradiction that ϵ_4 is non-negligible, then we construct adversary $\mathcal{B}^{\mathsf{sound}}_{\mathsf{del}}$ that breaks the soundness property of the underlying Del scheme as follows:

1. Receive $(n_{\mathsf{pb}}, 1^{n_{\mathsf{pr}}}, S, T)$ from $\mathcal{A}$.
2. Send $(1^\lambda, T)$ to $\mathcal{C}$ and receive $\mathsf{del.crs}^*$ from $\mathcal{C}$.
3. Construct crs similar to experiment $\mathsf{Exp}_{\mathcal{E}\mathsf{Del}}$ where $\mathsf{del.crs} = \mathsf{del.crs}^*$, and send it to $\mathcal{A}$.
4. Receive $(x_{\mathsf{pb}}, b, \pi)$ from $\mathcal{A}$.
5. Parse $\pi = (\mathsf{eh}_{\mathsf{pr}}, \mathsf{zk}.\pi_{\mathsf{enc}}, \mathsf{zk}.\pi_{\mathsf{edel}})$.
6. Compute $\mathsf{h}_{\mathsf{pb}} = \mathsf{HT.Hash}(\mathsf{ht.hk}, x_{\mathsf{pb}})$ and $\mathsf{h}_\Sigma \leftarrow \mathsf{HT.Hash}(\mathsf{ht.hk}, 0^S)$.
7. Compute $(\mathsf{h}^*_{\mathsf{pr},2}, r^*_2, \mathsf{del}.\pi^*) = \mathsf{NIZK}.\mathcal{E}_2(\mathsf{zk.td}_{\mathsf{edel}}, (\mathsf{h}_{\mathsf{pb}}, \mathsf{eh}_{\mathsf{pr}}, \mathsf{h}_\Sigma, b), \mathsf{zk}.\pi_{\mathsf{edel}})$ and $(x^*_{\mathsf{pr}}, \mathsf{h}^*_{\mathsf{pr},1}, r^*_1) \leftarrow \mathsf{NIZK}.\mathcal{E}_1(\mathsf{zk.td}_{\mathsf{enc}}, \mathsf{eh}_{\mathsf{pr}}, \mathsf{zk}.\pi_{\mathsf{enc}})$.
8. Send $(\mathsf{cf}_0 = (x_{\mathsf{pb}}, x^*_{\mathsf{pr}}, 0^S), b, \mathsf{del}.\pi^*)$ to $\mathcal{C}$.

Note that $(\mathsf{h}_{\mathsf{pb}}, \mathsf{h}^*_{\mathsf{pr},2}, \mathsf{h}_\Sigma) = \mathsf{Del.Digest}(\mathsf{cf}_0)$, $(x_{\mathsf{pb}}, x^*_{\mathsf{pr}}) \notin \mathcal{L}_{\mathcal{M},T}$ implies that $\mathsf{cf}_0 \notin \mathcal{L}_{\mathcal{M},T}$, and $\mathcal{B}^{\mathsf{sound}}_{\mathsf{del}}$ perfectly simulates the experiment $\mathsf{Exp}_{\mathcal{E}\mathsf{Del}}$ for $\mathcal{A}$. Thus $\mathcal{B}^{\mathsf{sound}}_{\mathsf{del}}$ has the same advantage ϵ_4 in breaking the soundness property of the underlying Del scheme.

□

Proof of Input Hiding. We will define the simulator as follows – (1) sample the crs similar to the Setup except that generate $\mathsf{zk.crs}_{\mathsf{enc}}$ and $\mathsf{zk.crs}_{\mathsf{edel}}$ using $\mathsf{zk.crs}_{\mathsf{enc}} \leftarrow \mathsf{NIZK}.\mathcal{S}_1(1^\lambda, 1^{n_{\mathsf{enc}}})$ and $\mathsf{zk.crs}_{\mathsf{edel}} \leftarrow \mathsf{NIZK}.\mathcal{S}_2(1^\lambda, 1^{n_{\mathsf{edel}}})$, and (2) generate the proof by computing $\mathsf{eh}_{\mathsf{pr}} \leftarrow \mathsf{PKE.Enc}(\mathsf{pke.pk}, 0^{|\mathsf{h}_{\mathsf{pr}}|})$, $\mathsf{zk}.\pi_{\mathsf{enc}} \leftarrow \mathsf{NIZK}.\mathcal{S}_1(\mathsf{eh}_{\mathsf{pr}})$, $\mathsf{h}_{\mathsf{pb}} \leftarrow \mathsf{HT.Hash}(\mathsf{ht.hk}, x_{\mathsf{pb}})$, $\mathsf{h}_\Sigma \leftarrow \mathsf{HT.Hash}(\mathsf{ht.hk}, 0^S)$, and $\mathsf{zk}.\pi_{\mathsf{edel}} \leftarrow \mathsf{NIZK}.\mathcal{S}_2(\mathsf{h}_{\mathsf{pb}}, \mathsf{eh}_{\mathsf{pr}}, \mathsf{h}_\Sigma, b)$ and letting $\pi = (\mathsf{eh}_{\mathsf{pr}}, \mathsf{zk}.\pi_{\mathsf{enc}}, \mathsf{zk}.\pi_{\mathsf{edel}})$.

We will define the following hybrids:

hyb_0**.** This is the real experiment where the $\mathcal{A}$ receives $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda, n_{\mathsf{pb}}, 1^{n_{\mathsf{pr}}}, S, T)$, then makes queries $(x_{\mathsf{pb}}, x_{\mathsf{pr}})$ and receives $(b, \pi) \leftarrow \mathsf{Prove}(\mathsf{crs}, x_{\mathsf{pb}}, x_{\mathsf{pr}})$.

hyb_1**.** Similar to hyb_0 except that $\mathsf{zk.crs}_{\mathsf{enc}} \leftarrow \mathsf{NIZK}.\mathcal{S}_1(1^\lambda, 1^{n_{\mathsf{enc}}})$ in the crs, and in the query responses, $\mathsf{zk}.\pi_{\mathsf{enc}}$ is generated using $\mathsf{zk}.\pi_{\mathsf{enc}} \leftarrow \mathsf{NIZK}.\mathcal{S}_1(\mathsf{eh}_{\mathsf{pr}})$.

hyb_2**.** Similar to hyb_1 except that $\mathsf{zk.crs}_{\mathsf{edel}} \leftarrow \mathsf{NIZK}.\mathcal{S}_2(1^\lambda, 1^{n_{\mathsf{edel}}})$ in the crs, and in the query responses, $\mathsf{zk}.\pi_{\mathsf{edel}}$ is generated using $\mathsf{zk}.\pi_{\mathsf{edel}} \leftarrow \mathsf{NIZK}.\mathcal{S}_2(\mathsf{h}_{\mathsf{pb}}, \mathsf{eh}_{\mathsf{pr}}, \mathsf{h}_\Sigma, b)$.

hyb_3**.** Similar to hyb_2 except that $\mathsf{eh}_{\mathsf{pr}} = \mathsf{PKE.Enc}(\mathsf{pke.pk}, 0^{|\mathsf{h}_{\mathsf{pr}}|})$. Note that this is the simulated experiment.

Let $\mathcal{A}$ output a bit $b \in \{0,1\}$ at the end of each experiment and denote this output by $\mathsf{hyb}_i(\mathcal{A})$.

Lemma 5.3. If NIZK is zero-knowledge, then for any adversary $\mathcal{A}$ it holds that

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

Proof. Suppose towards the contradiction that for some adversary $\mathcal{A}$ and some non-negligible function $\epsilon(\cdot)$ it holds that $|\Pr[\mathsf{hyb}_0(\mathcal{A}) = 1] - \Pr[\mathsf{hyb}_1(\mathcal{A}) = 1]| \geq \epsilon(\lambda)$. We construct an adversary $\mathcal{B}_{\mathsf{zk}}$ against the zero-knowledge property of the underlying NIZK scheme as follows:

1. Receive $(1^\lambda, n_{\mathsf{pb}}, 1^{n_{\mathsf{pr}}}, S, T)$ from the adversary.
2. Receive $\mathsf{zk.crs}_{\mathsf{enc}} \leftarrow \mathsf{NIZK.Setup}(1^\lambda, 1^{n_{\mathsf{enc}}})$ from $\mathcal{C}$.
3. Construct the rest of the crs similar to the Setup.
4. On any query $(x_{\mathsf{pb}}, x_{\mathsf{pr}})$ generate the proof similar to $\mathsf{Prove}(\mathsf{crs}, x_{\mathsf{pb}}, x_{\mathsf{pr}})$ except that to generate $\mathsf{zk}.\pi_{\mathsf{enc}}$, make a query to $\mathcal{C}$ on $(\mathsf{eh}_{\mathsf{pr}}, (x_{\mathsf{pr}}, \mathsf{h}_{\mathsf{pr}}, r))$.
5. Send π to $\mathcal{A}$ and output whatever $\mathcal{A}$ outputs.

Note that if the challenger uses $\mathsf{NIZK.Setup}$ and $\mathsf{NIZK.Prove}$ (resp. $\mathcal{S}_1$) in the experiment then $\mathcal{B}_{\mathsf{zk}}$ perfectly simulates hyb_0 (resp. hyb_1). Thus $\mathcal{B}_{\mathsf{zk}}$ has the same advantage $\epsilon(\lambda)$ as $\mathcal{A}$ in breaking the zero-knowledge property of the NIZK scheme. □

Lemma 5.4. If NIZK is zero-knowledge, then for any adversary $\mathcal{A}$ it holds that

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

Proof. Suppose towards the contradiction that for some adversary $\mathcal{A}$ and some non-negligible function $\epsilon(\cdot)$ it holds that $|\Pr[\mathsf{hyb}_1(\mathcal{A}) = 1] - \Pr[\mathsf{hyb}_2(\mathcal{A}) = 1]| \geq \epsilon(\lambda)$. We construct an adversary $\mathcal{B}_{\mathsf{zk}}$ against the zero-knowledge property of the underlying NIZK scheme as follows:

1. Receive $(1^\lambda, n_{\mathsf{pb}}, 1^{n_{\mathsf{pr}}}, S, T)$ from the adversary.
2. Receive $\mathsf{zk.crs}_{\mathsf{edel}} \leftarrow \mathsf{NIZK.Setup}(1^\lambda, 1^{n_{\mathsf{edel}}})$ from $\mathcal{C}$.
3. Construct the rest of the crs similar to hyb_1.
4. On any query $(x_{\mathsf{pb}}, x_{\mathsf{pr}})$ generate the proof similar to hyb_1 except that to generate $\mathsf{zk}.\pi_{\mathsf{edel}}$, make a query to $\mathcal{C}$ on $((\mathsf{h}_{\mathsf{pb}}, \mathsf{eh}_{\mathsf{pr}}, \mathsf{h}_\Sigma, b), (x_{\mathsf{pr}}, \mathsf{h}_{\mathsf{pr}}, r))$.
5. Send π to $\mathcal{A}$ and output whatever $\mathcal{A}$ outputs.

Note that if the challenger uses $\mathsf{NIZK.Setup}$ and $\mathsf{NIZK.Prove}$ (resp. $\mathcal{S}_2$) in the experiment then $\mathcal{B}_{\mathsf{zk}}$ perfectly simulates hyb_1 (resp. hyb_2). Thus $\mathcal{B}_{\mathsf{zk}}$ has the same advantage $\epsilon(\lambda)$ as $\mathcal{A}$ in breaking the zero-knowledge property of the NIZK scheme. □

Lemma 5.5. If PKE is secure, then for any adversary $\mathcal{A}$ it holds that

$$|\Pr[\mathsf{hyb}_2(\mathcal{A}) = 1] - \Pr[\mathsf{hyb}_3(\mathcal{A}) = 1]| \leq \mathsf{negl}(\lambda).$$

Proof. Suppose towards the contradiction that for some adversary $\mathcal{A}$ and some non-negligible function $\epsilon(\cdot)$ it holds that $|\Pr[\mathsf{hyb}_2(\mathcal{A}) = 1] - \Pr[\mathsf{hyb}_3(\mathcal{A}) = 1]| \geq \epsilon(\lambda)$. We construct an adversary $\mathcal{B}_{\mathsf{pke}}$ against the security of the underlying PKE scheme as follows:

1. Receive $(1^\lambda, n_{\mathsf{pb}}, 1^{n_{\mathsf{pr}}}, S, T)$ from the adversary.
2. Receive $\mathsf{pke.pk} \leftarrow \mathsf{PKE.Enc}(1^\lambda)$ from $\mathcal{C}$.
3. Construct the rest of the crs similar to hyb_1.
4. On any query $(x_{\mathsf{pb}}, x_{\mathsf{pr}})$ compute $\mathsf{h}_{\mathsf{pr}} \leftarrow \mathsf{HT.Hash}(\mathsf{ht.hk}, x_{\mathsf{pr}})$ send a query to $\mathcal{C}$ on h_{pr} and receive $\mathsf{eh}_{\mathsf{pr}}$. Generate the rest of the proof similar to hyb_2.
5. Send π to $\mathcal{A}$ and output whatever $\mathcal{A}$ outputs.

Note that if the challenger uses $\mathsf{PKE.Enc}(\mathsf{pke.pk}, \cdot)$ (resp. $\mathsf{PKE.Enc}(\mathsf{pke.pk}, 0^{|\mathsf{h}_{\mathsf{pr}}|})$) in the experiment then $\mathcal{B}_{\mathsf{pke}}$ perfectly simulates hyb_2 (resp. hyb_3). Thus $\mathcal{B}_{\mathsf{pke}}$ has the same advantage $\epsilon(\lambda)$ as $\mathcal{A}$ in breaking the security of the PKE scheme. □

We conclude the input hiding proof by Lemmas 5.3 to 5.5 and a hybrid argument.

□

Corollary 5.6. Assuming $X \in \{\mathsf{LWE}, O(1)\text{-}\mathsf{LIN}, \text{sub-exp-}\mathsf{DDH}\}$, there exists an encrypted RAM delegation with straight-line extractor, and input hiding. Additionally, if we plug-in our optimally efficient NIZK from Section 6.2, the crs size and the setup running-time will only grow with $\log n_{\mathsf{pr}}$.[2]

Remark 5.7. If we use NIZKs with statistical zero-knowledge property and instead of PKEs we use statistically-hiding commitments, then our construction achieves statistical input-hiding property.

5.2 Reusable Encrypted RAM Delegation

We observe that our construction of encrypted RAM delegation is already reusable. In the following we show how to partition the algorithms in Construction 5.1 to match the syntax of reusable encrypted RAM delegation.

Construction 5.8. [Reusable Encrypted RAM Delegation] Let $\mathsf{PKE} = (\mathsf{PKE.Gen}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$ be a public-key encryption scheme, $\mathsf{Del} = (\mathsf{Del.Setup}, \mathsf{Del.Digest}, \mathsf{Del.Prove}, \mathsf{Del.Verify})$ be a split memory RAM delegation scheme associated with a hash tree HT, and $\mathsf{NIZK} = (\mathsf{NIZK.Setup}, \mathsf{NIZK.Prove}, \mathsf{NIZK.Verify})$ be a non-interactive zero-knowledge scheme. We construct an encrypted RAM delegation $\mathcal{E}\mathsf{Del} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ as follows:

[2]We avoid circularity by first bootstrapping an inefficient Initial eRAM using basic NIZKs. This initial primitive is sufficient to build optimally efficient NIZKs, which are then used to construct the final efficient eRAM.

$\mathsf{Setup}(1^\lambda, n_{\mathsf{pb}}, 1^{n_{\mathsf{pr}}}, S, T) \to (\mathsf{crs})$**.** Same as in Construction 5.1.

$\mathsf{PrivEnc}(\mathsf{crs}, x_{\mathsf{pr}}) \to (\mathsf{eh}_{\mathsf{pr}}, \pi_{\mathsf{enc}}, \mathsf{aux})$**.** Perform Items 1 to 3 from the prover's algorithm in Construction 5.1 and output $(\mathsf{eh}_{\mathsf{pr}}, \pi_{\mathsf{enc}} = \mathsf{zk}.\pi_{\mathsf{enc}}, \mathsf{aux} = (x_{\mathsf{pr}}, r))$.

$\mathsf{Prove}(\mathsf{crs}, x_{\mathsf{pb}}, \mathsf{aux}) \to (b, \pi_{\mathsf{edel}})$**.** Same as in Construction 5.1 except that it takes $\mathsf{aux} = (x_{\mathsf{pr}}, r)$ as input and doesn't compute $\mathsf{zk}.\pi_{\mathsf{enc}}$.

$\mathsf{VerifyEnc}(\mathsf{crs}, \mathsf{eh}_{\mathsf{pr}}, \pi_{\mathsf{enc}}) \to 0/1$**.** Parse $\mathsf{crs} = (\mathsf{del.crs}, \mathsf{pke.pk}, \mathsf{zk.crs}_{\mathsf{enc}}, \mathsf{zk.crs}_{\mathsf{edel}}, S)$. Output 1 if $\mathsf{NIZK}.\mathsf{Verify}(\mathsf{zk.crs}_{\mathsf{enc}}, \mathsf{eh}_{\mathsf{pr}}, \pi_{\mathsf{enc}}) = 1$; else output 0.

$\mathsf{Verify}(\mathsf{crs}, \mathsf{h}_{\mathsf{pb}}, \mathsf{eh}_{\mathsf{pr}}, b, \pi_{\mathsf{edel}}) \to 0/1$**.** Parse crs as above. Let $\mathsf{h}_\Sigma \leftarrow \mathsf{HT}.\mathsf{Hash}(\mathsf{ht.hk}, 0^S)$. Output 1 if $\mathsf{NIZK}.\mathsf{Verify}(\mathsf{zk.crs}_{\mathsf{edel}}, (\mathsf{h}_{\mathsf{pb}}, \mathsf{eh}_{\mathsf{pr}}, \mathsf{h}_\Sigma, b), \pi_{\mathsf{edel}}) = 1$; else output 0.

Theorem 5.9. Assuming split memory RAM delegations, PKEs, and NIZKs, the Construction 5.8 is a reusable encrypted RAM delegation (Definition 4.2) with strong soundness, straight-line extractor, and input hiding.

Completeness. The completeness follows by the construction and the completeness of the underlying Del and NIZK and the correctness of the underlying PKE.

Efficiency. The proof is similar to the proof as in the encrypted RAM delegation. Refer to Section 5.1 for details.

5.2.1 Security Analysis.

Proof of Straight-Line Extractor. The proof is similar to the proof as in the encrypted RAM delegation. Refer to Section 5.1.1 for details. □

Proof of Input Hiding. The proof is similar to the proof as in the encrypted RAM delegation. Refer to Section 5.1.1 for details. □

Proof of Strong Soundness. First define the following hybrids:

- hyb_0**.** This is the real experiment where the adversary $\mathcal{A}$ on input 1^λ sends $(n_{\mathsf{pb}}, 1^{n_{\mathsf{pr}}}, S, T)$ to the challenger $\mathcal{C}$. Then $\mathcal{C}$ runs $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, n_{\mathsf{pb}}, 1^{n_{\mathsf{pr}}}, S, T)$ and sends crs to $\mathcal{A}$. Upon receiving crs, $\mathcal{A}$ outputs $(x_{\mathsf{pb}}, \mathsf{eh}_{\mathsf{pr}}, \pi^{(0)}_{\mathsf{edel}}, \pi^{(1)}_{\mathsf{edel}})$. $\mathcal{A}$ wins if for $b \in \{0,1\}$, $\mathsf{Verify}(\mathsf{zk.crs}_{\mathsf{edel}}, \mathsf{h}_{\mathsf{pb}}, \mathsf{eh}_{\mathsf{pr}}, b, \pi^{(b)}_{\mathsf{edel}}) = 1$.
- hyb_1**.** Similar to hyb_0 except that $\mathcal{C}$ generates $\mathsf{zk.crs}_{\mathsf{edel}}$ in the crs by running $\mathsf{NIZK}.\mathcal{E}$ and appends $\mathsf{zk.td}_{\mathsf{edel}}$ to td.

The indistinguishability of the hyb_0 and hyb_1 directly follows from the crs indistinguishability of the underlying NIZK scheme.

To conclude the proof we only need to show that the winning probability of $\mathcal{A}$ in hyb_1 is negligible. Let Setup' be the setup as defined in hyb_1. Define the experiment $\mathsf{Exp}_{\mathcal{E}\mathsf{Del}}$ be the same as in hyb_1 except that $\mathcal{C}$ additionally computes h_Σ and extracts the witness from NIZK proofs as follows:

$$\mathsf{Exp}_{\mathcal{E}\mathsf{Del}} = \left[\begin{array}{l} (n_{\mathsf{pb}}, 1^{n_{\mathsf{pr}}}, S, T) \leftarrow \mathcal{A}(1^\lambda), \\ (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Setup}'(1^\lambda, n_{\mathsf{pb}}, 1^{n_{\mathsf{pr}}}, S, T), \\ (\mathsf{h}_{\mathsf{pb}}, \mathsf{eh}_{\mathsf{pr}}, \pi^{(0)}_{\mathsf{edel}}, \pi^{(1)}_{\mathsf{edel}}) \leftarrow \mathcal{A}(\mathsf{crs}), \\ \mathsf{h}_\Sigma \leftarrow \mathsf{HT.Hash}(\mathsf{ht.hk}, 0^S), \\ (\mathsf{h}^{(b)}_{\mathsf{pr}}, r^{(b)}, \mathsf{del}.\pi^{(b)}) = \mathsf{NIZK}.\mathcal{E}(\mathsf{zk.td}_{\mathsf{edel}}, (\mathsf{h}_{\mathsf{pb}}, \mathsf{eh}_{\mathsf{pr}}, \mathsf{h}_\Sigma, b), \pi^{(b)}_{\mathsf{edel}}) \text{ for } b \in \{0,1\} \end{array}\right]$$

For an adversary $\mathcal{A}$ we define the output of the above experiment, $\mathsf{Exp}_{\mathcal{E}\mathsf{Del}}(\mathcal{A})$, to be 1, if $\mathcal{A}$ wins (similar to hyb_2), namely:

$$\left[\ \mathsf{NIZK.Verify}(\mathsf{zk.crs}_{\mathsf{edel}}, (\mathsf{h}_{\mathsf{pb}}, \mathsf{eh}_{\mathsf{pr}}, \mathsf{h}_\Sigma, b), \pi^{(b)}_{\mathsf{edel}}) = 1 \text{ for } b \in \{0,1\} \quad : \quad \mathsf{Exp}_{\mathcal{E}\mathsf{Del}}\right]$$

Let $(x^{(b)}_{\mathsf{edel}}, w^{(b)}_{\mathsf{edel}}) = ((\mathsf{h}_{\mathsf{pb}}, \mathsf{eh}_{\mathsf{pr}}, \mathsf{h}_\Sigma, b), (\mathsf{h}^{(b)}_{\mathsf{pr}}, r^{(b)}, \mathsf{del}.\pi^{(b)}))$. Define the following variables:

$$\epsilon_1 = \Pr[\mathsf{Exp}_{\mathcal{E}\mathsf{Del}}(\mathcal{A}) = 1] - \Pr\left[\begin{array}{ll} (x^{(0)}_{\mathsf{edel}}, w^{(0)}_{\mathsf{edel}}) \in \mathcal{L}_{\mathsf{edel}} & \\ \mathsf{NIZK.Verify}(\mathsf{zk.crs}_{\mathsf{edel}}, (\mathsf{h}_{\mathsf{pb}}, \mathsf{eh}_{\mathsf{pr}}, \mathsf{h}_\Sigma, 1), \pi^{(1)}_{\mathsf{edel}}) = 1 & : \ \mathsf{Exp}_{\mathcal{E}\mathsf{Del}} \end{array}\right]. \quad (7)$$

$$\epsilon_2 = \Pr\left[\begin{array}{ll} (x^{(0)}_{\mathsf{edel}}, w^{(0)}_{\mathsf{edel}}) \in \mathcal{L}_{\mathsf{edel}} & \\ \mathsf{NIZK.Verify}(\mathsf{zk.crs}_{\mathsf{edel}}, (\mathsf{h}_{\mathsf{pb}}, \mathsf{eh}_{\mathsf{pr}}, \mathsf{h}_\Sigma, 1), \pi^{(1)}_{\mathsf{edel}}) = 1 & : \ \mathsf{Exp}_{\mathcal{E}\mathsf{Del}} \end{array}\right] - \Pr\left[\ (x^{(b)}_{\mathsf{edel}}, w^{(b)}_{\mathsf{edel}}) \in \mathcal{L}_{\mathsf{edel}} \text{ for } b \in \{0,1\} \quad : \quad \mathsf{Exp}_{\mathcal{E}\mathsf{Del}}\ \right]. \quad (8)$$

$$\epsilon_3 = \Pr\left[\ (x^{(b)}_{\mathsf{edel}}, w^{(b)}_{\mathsf{edel}}) \in \mathcal{L}_{\mathsf{edel}} \text{ for } b \in \{0,1\} \quad : \quad \mathsf{Exp}_{\mathcal{E}\mathsf{Del}}\ \right] - \Pr\left[\begin{array}{ll} \mathsf{h}^{(0)}_{\mathsf{pr}} = \mathsf{h}^{(1)}_{\mathsf{pr}} & \\ \mathsf{Del.Verify}(\mathsf{del.crs}, (\mathsf{h}_{\mathsf{pb}}, \mathsf{h}^{(b)}_{\mathsf{pr}}, \mathsf{h}_\Sigma), b, \mathsf{del}.\pi^{(b)}) = 1 \text{ for } b \in \{0,1\} & : \ \mathsf{Exp}_{\mathcal{E}\mathsf{Del}} \end{array}\right]. \quad (9)$$

$$\epsilon_4 \ = \ \Pr\left[\begin{array}{ll} \mathsf{h}^{(0)}_{\mathsf{pr}} = \mathsf{h}^{(1)}_{\mathsf{pr}} & \\ \mathsf{Del.Verify}(\mathsf{del.crs}, (\mathsf{h}_{\mathsf{pb}}, \mathsf{h}^{(b)}_{\mathsf{pr}}, \mathsf{h}_\Sigma), b, \mathsf{del}.\pi^{(b)}) = 1 \text{ for } b \in \{0,1\} & : \ \mathsf{Exp}_{\mathcal{E}\mathsf{Del}} \end{array}\right]. \quad (10)$$

Now since $\Pr[\mathsf{Exp}_{\mathcal{E}\mathsf{Del}}(\mathcal{A}) = 1] = \epsilon_1 + \epsilon_2 + \epsilon_3 + \epsilon_4$ we only need to show that for $i \in [4]$, ϵ_i is negligible.

Case 1. Suppose towards the contradiction that ϵ_1 is non-negligible, then by Eq. (7) it holds that:

$$\Pr[\mathsf{NIZK.Verify}(\mathsf{zk.crs}_{\mathsf{edel}}, (\mathsf{h}_{\mathsf{pb}}, \mathsf{eh}_{\mathsf{pr}}, \mathsf{h}_\Sigma, 0), \pi^{(0)}_{\mathsf{edel}}) = 1 \wedge (x^{(0)}_{\mathsf{edel}}, w^{(0)}_{\mathsf{edel}}) \notin \mathcal{L}_{\mathsf{edel}}] \geq \epsilon_1$$

Now we construct adversary $\mathcal{B}^{\mathcal{E}}_{\mathsf{zk}}$ that breaks the straight-line extractor property of the underlying NIZK scheme as follows:

1. Receive $(n_{\mathsf{pb}}, 1^{n_{\mathsf{pr}}}, S, T)$ from $\mathcal{A}$.
2. Receive $\mathsf{zk.crs}^*_{\mathsf{edel}}$ from $\mathcal{C}$.
3. Construct crs similar to experiment $\mathsf{Exp}_{\mathcal{E}\mathsf{Del}}$ where $\mathsf{zk.crs}_{\mathsf{edel}} = \mathsf{zk.crs}^*_{\mathsf{edel}}$, and send it to $\mathcal{A}$.
4. Receive $(\mathsf{h}_{\mathsf{pb}}, \mathsf{eh}_{\mathsf{pr}}, \pi^{(0)}_{\mathsf{edel}}, \pi^{(1)}_{\mathsf{edel}})$ from $\mathcal{A}$.
5. Compute $\mathsf{h}_\Sigma = \mathsf{HT.Hash}(\mathsf{ht.hk}, 0^S)$.
6. Send $((\mathsf{h}_{\mathsf{pb}}, \mathsf{eh}_{\mathsf{pr}}, \mathsf{h}_\Sigma, 0), \pi^{(0)}_{\mathsf{edel}})$ to $\mathcal{C}$.

Note that $\mathcal{B}^{\mathcal{E}}_{\mathsf{zk}}$ perfectly simulates the experiment $\mathsf{Exp}_{\mathcal{E}\mathsf{Del}}$ for $\mathcal{A}$, thus has the same advantage ϵ_1 in breaking the straight-line extractor property of the underlying NIZK scheme.

Case 2. Similar to case 1.

Case 3. Suppose towards the contradiction that ϵ_3 is non-negligible. First note that $(x^{(b)}_{\mathsf{edel}}, w^{(b)}_{\mathsf{edel}}) \in \mathcal{L}_{\mathsf{edel}}$ is equivalent with:

$$\mathsf{eh}_{\mathsf{pr}} = \mathsf{PKE.Enc}(\mathsf{pke.pk}, \mathsf{h}^{(b)}_{\mathsf{pr}}; r^{(b)}) \quad \wedge \quad \mathsf{Del.Verify}(\mathsf{del.crs}, (\mathsf{h}_{\mathsf{pb}}, \mathsf{h}^{(b)}_{\mathsf{pr}}, \mathsf{h}_\Sigma), b, \mathsf{del}.\pi^{(b)}) = 1$$

then by Eq. (9) it holds that

$$\Pr\left[\begin{array}{ll} \mathsf{eh}_{\mathsf{pr}} = \mathsf{PKE.Enc}(\mathsf{pke.pk}, \mathsf{h}^{(0)}_{\mathsf{pr}}; r^{(0)}) \wedge & \\ \mathsf{eh}_{\mathsf{pr}} = \mathsf{PKE.Enc}(\mathsf{pke.pk}, \mathsf{h}^{(1)}_{\mathsf{pr}}; r^{(1)}) \wedge \mathsf{h}^{(0)}_{\mathsf{pr}} \neq \mathsf{h}^{(1)}_{\mathsf{pr}} & : \quad \mathsf{Exp}_{\mathcal{E}\mathsf{Del}} \end{array}\right] \geq \epsilon_3$$

This breaks the perfect correctness of the underlying PKE scheme.

Case 4. Suppose towards the contradiction that ϵ_4 is non-negligible, then we construct adversary $\mathcal{B}^{\mathsf{sound}}_{\mathsf{del}}$ that breaks the strong soundness property of the underlying Del scheme as follows:

1. Receive $(n_{\mathsf{pb}}, 1^{n_{\mathsf{pr}}}, S, T)$ from $\mathcal{A}$.
2. Send $(1^\lambda, T)$ to $\mathcal{C}$ and receive $\mathsf{del.crs}^*$ from $\mathcal{C}$.
3. Construct crs similar to experiment $\mathsf{Exp}_{\mathcal{E}\mathsf{Del}}$ where $\mathsf{del.crs} = \mathsf{del.crs}^*$, and send it to $\mathcal{A}$.
4. Receive $(\mathsf{h}_{\mathsf{pb}}, \mathsf{eh}_{\mathsf{pr}}, \pi^{(0)}_{\mathsf{edel}}, \pi^{(1)}_{\mathsf{edel}})$ from $\mathcal{A}$.
5. Compute $(\mathsf{h}^{(b)}_{\mathsf{pr}}, r^{(b)}, \mathsf{del}.\pi^{(b)}) = \mathsf{NIZK}.\mathcal{E}(\mathsf{zk.td}_{\mathsf{edel}}, (\mathsf{h}_{\mathsf{pb}}, \mathsf{eh}_{\mathsf{pr}}, \mathsf{h}_\Sigma, b), \pi^{(b)}_{\mathsf{edel}})$ for $b \in \{0, 1\}$.
6. Compute $\mathsf{h}_\Sigma = \mathsf{HT.Hash}(\mathsf{ht.hk}, 0^S)$.
7. Send $((\mathsf{h}_{\mathsf{pb}}, \mathsf{h}^{(0)}_{\mathsf{pr}}, \mathsf{h}_\Sigma), \pi^{(0)}_{\mathsf{edel}}, \pi^{(1)}_{\mathsf{edel}})$ to $\mathcal{C}$.

Note that $\mathcal{B}^{\mathsf{sound}}_{\mathsf{del}}$ perfectly simulates the experiment $\mathsf{Exp}_{\mathcal{E}\mathsf{Del}}$ for $\mathcal{A}$. Thus $\mathcal{B}^{\mathsf{sound}}_{\mathsf{del}}$ has the same advantage ϵ_4 in breaking the strong soundness property of the underlying Del scheme.

□

Corollary 5.10. Assuming $X \in \{\mathsf{LWE}, O(1)\text{-}\mathsf{LIN}, \text{sub-exp-}\mathsf{DDH}\}$, there exists a reusable encrypted RAM delegation with strong soundness, straight-line extractor, and input hiding. Additionally, by using our optimally efficient NIZK from Section 6.2, the crs size and setup's running-time will only grow with $\log n_{\mathsf{pr}}$.

6 Homomorphic Non-Interactive Zero-Knowledge

We begin by recalling the definition of homomorphic NIZKs as introduced in [ADKL19]. Next, we describe some useful tools that we rely on to construct homomorphic NIZKs. The main tool that we rely on are optimally-efficient NIZKs. We provide a simple construction for optimally-efficient NIZKs using encrypted RAM delegation scheme in Section 6.2, and later use them to design fully-homomorphic NIZKs.

6.1 Definitions: Homomorphic NIZKs

We start by defining some useful notation for circuits that we will rely on throughout this section.

Notation: circuits as graphs. We interpret each circuit C as a directed acyclic graph. We say a directed acyclic graph $G = (V, E)$ represents a boolean circuit $C : \{0,1\}^k \to \{0,1\}$ if G is defined as follows. In graph G, each vertex $v \in V$ of in-degree 0 is an input node labeled as $i \in [k]$, there are k such vertices. For each remaining vertex of G, it is labeled as a logic gate (NAND gate for simplicity). Each internal wire of the circuit corresponds to a directed edge in G, where the direction of such edge indicates the direction of signal. The node v_{out} which has an out-degree of 0 represents the output gate.

Merge nodes in graphs. Given graph $G = (V, E)$, the merge of vertices $u, v \in V$ is defined as follows: First, for each $w \in V \setminus \{u, v\}$, replace each edge (w, v) (resp. (v, w)) with a new edge (w, u) (resp. (u, w)) if there exists such edges. Next, remove node v, and edges $(u, v), (v, u)$ if there exists such edges.

Split nodes. Given $G = (V, E)$, the split of a vertex $u \in V$ is as follows: The first step is adding a new vertex u'. Next for each vertex v such that $(u, v) \in E$, remove such edge, and add an additional edge (u', v). Intuitively, the node u is "split" into two disconnected nodes u, u'.

Cuttable node. "Cuttable node" is considered as a duality term of an edge cut. Given $G = (V, E)$, then vertex $u \in V$ is a cuttable node if and only if splitting vertex u in G results in an additional maximally connected component [3] in the original graph.[4] Note that in our work, a node with in-degree or out-degree 0 will not be considered as a cuttable node.

Defining homomorphic NIZKs. For some boolean circuit family $\mathcal{C}$ that takes n-bits of input, consider the following NP language:

$$\mathcal{L} = \{(C, b) \in \mathcal{C} \times \{0,1\} : \exists \omega \in \{0,1\}^n \text{ s.t. } C(\omega) = b\} \tag{11}$$

A NIZK proof system $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ for any such language $\mathcal{L}$ is a fully-homomorphic NIZK system if there also exists a polynomial algorithm, Eval, with the following syntax:

$\mathsf{Eval}(\mathsf{crs}, C, \{(C_j, b_j, \pi_j)\}_{j\in[k]}) \to (C', b, \pi)$. The evaluation algorithm takes as input the CRS crs, k instances $\{(C_i, b_i)\}_{i=1}^k$, their proofs $\{\pi_i\}_{i=1}^k$, and a circuit $C : \{0,1\}^k \to \{0,1\}$. It outputs the composed instance (C', b) where $C' = \mathsf{Compose}(C, \{(C_i, b_i)\}_{i=1}^k)$ and a proof π. The $\mathsf{Compose}$ procedure is described below.

$\mathsf{Compose}(C, C_1, \ldots, C_k) \to C'$. Let graph G represent the circuit C and graph G_i represent the circuit C_i for all $i \in [k]$. It proceeds as follows:

[3] The term maximally connected component refers to the connected subgraph to which no vertex can be added and it remains connected.

[4] When we say connected component, we abuse the notation and consider the graph as an undirected one.

1. Define a new graph $G' = (V', E')$ where $V' = (\bigcup_{i=1}^{k} V_i) \cup V$ and $E' = (\bigcup_{i=1}^{k} E_i) \cup E$.
2. For all $i \in [k]$, let $u_i \in V_i$ be the output gate of C_i, and $v_i \in V$ be the i-th input gate of C. Next for all $i \in [k]$, it merges every pair of u_i, v_i in graph G' and labels the merged vertex as a NAND gate. It outputs C' which corresponds to graph G' as the resulting circuit.

In addition to the regular completeness, soundness, and zero-knowledge properties for standard NIZKs (Definition 3.9), homomorphic NIZKs must satisfy two additional properties called completeness of Eval and unlinkability defined as follows:

Completeness of Eval. For any PPT adversary $\mathcal{A}$ and for all $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l} \exists i \in [k] \text{ s.t. } \mathsf{Verify}(\mathsf{crs}, (C_i, b_i), \pi_i) = 0 \\ \vee \left(\begin{array}{c} \mathsf{Verify}(\mathsf{crs}, (C', b), \pi) = 1 \\ \wedge\, C' = \mathsf{Compose}(C, C_1, \ldots, C_k) \\ \wedge\, b = C(b_1, \ldots, b_k) \end{array}\right) \end{array} : \begin{array}{l} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda) \\ (C, \{(C_j, b_j, \pi_j)\}_{j\in[k]}) \leftarrow \mathcal{A}(\mathsf{crs}) \\ (C', b, \pi) \leftarrow \mathsf{Eval}(\mathsf{crs}, C, \{(C_j, b_j, \pi_j)\}_{j\in[k]}) \end{array}\right] = 1.$$

Unlinkability. This property states that in any homomorphic NIZK scheme a proof obtained by Eval to be indistinguishable from a freshly generated proof. Formally, for any stateful admissible PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, the following equation holds:

$$\Pr\left[\mathcal{A}(C^*_\beta, b^*_\beta, \pi^*_\beta) = \beta : \begin{array}{l} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda), \beta \leftarrow \{0,1\} \\ (C, \{(C_i, b_i, \omega_i, \pi_i)\}_{i\in[k]}) \leftarrow \mathcal{A}(\mathsf{crs}) \\ (C^*_0, b^*_0, \pi^*_0) \leftarrow \mathsf{Eval}(\mathsf{crs}, C, \{(C_i, b_i, \pi_i)\}_{i\in[k]}) \\ C^*_1 = \mathsf{Compose}(C, C_1, \ldots, C_k) \\ b^*_1 = C^*_1(\omega_1, \ldots, \omega_k) \\ \pi^*_1 \leftarrow \mathsf{Prove}(\mathsf{crs}, (C^*_1, b^*_1), (\omega_1, \ldots, \omega_k)) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda),$$

where $\mathcal{A}$ is admissible if $\mathsf{Verify}(\mathsf{crs}, (C_i, b_i), \pi_i) = 1$ and $C_i(\omega_i) = b_i$ for all $i \in [k]$.

6.2 Tools Needed: Optimally Efficient NIZK and Rate-1 Message Encodings

In this section, we construct an optimally efficient NIZK using our encrypted RAM delegation scheme and rate-1 message encodings.

Construction 6.1. Let $\mathsf{r1Enc} = (\mathsf{Encode}, \mathsf{Decode})$ be a secure rate-1 message encoding scheme (Definition 3.14) and $\mathcal{E}\mathsf{Del} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ be an encrypted RAM delegation scheme w.r.to HT (Definition 4.1) for machine $\mathcal{M}'$ defined as follows – $\mathcal{M}'$ has $(\lambda, \mathcal{M})$ hardwired, takes (x, ρ) as public input, and sk as private input, computes $w = \mathsf{r1Enc.Decode}(1^\lambda, \rho, \mathsf{sk})$ and output 1 if and only if $\mathcal{M}(x, w) = 1$. We construct an optimally efficient NIZK for the language $\mathcal{L}_{\mathcal{M},T} = \{(x, w) : \mathcal{M}(x, w, T) = 1\}$ as follows:

$\mathsf{Setup}(1^\lambda, T) \to \mathsf{crs}$**.** This probabilistic algorithm samples $\mathcal{E}\mathsf{Del}$ scheme's crs, $\mathsf{del.crs} \leftarrow \mathcal{E}\mathsf{Del.Setup}(1^\lambda, 1^\lambda, T)$. Output $\mathsf{crs} = (\lambda, \mathsf{del.crs})$.

$\mathsf{Prove}(\mathsf{crs}, x, w) \to \pi$. This probabilistic algorithm parses crs as $(\lambda, \mathsf{del.crs})$ and does the following.

1. Let $(\rho, \mathsf{sk}) \leftarrow \mathsf{r1Enc.Encode}(1^\lambda, w)$. Set $x_{\mathsf{pb}} := (x, \rho)$ and $x_{\mathsf{pr}} := \mathsf{sk}$.
2. Sample $\mathcal{E}\mathsf{Del}$ proof $\mathsf{del}.\pi \leftarrow \mathcal{E}\mathsf{Del.Prove}(\mathsf{del.crs}, x_{\mathsf{pb}}, x_{\mathsf{pr}})$ and output the proof $\pi = (\rho, \mathsf{del}.\pi)$.

$\mathsf{Verify}(\mathsf{crs}, x, \pi) \to 0/1$. This poly-time algorithm parses crs as $(\lambda, \mathsf{del.crs})$, π as $(\rho, \mathsf{del}.\pi)$, computes $\mathsf{h}_{\mathsf{pb}} = \mathsf{HT.Hash}(\mathsf{ht.hk}, (x, \rho))$, and outputs $\mathcal{E}\mathsf{Del.Verify}(\mathsf{del.crs}, \mathsf{h}_{\mathsf{pb}}, 1, \mathsf{del}.\pi)$.

Theorem 6.2. If $\mathcal{E}\mathsf{Del}$ is a secure encrypted RAM delegation scheme w.r.to hash tree HT for language $\mathcal{L}_{\mathcal{M}'}$ (Definition 4.1) and $\mathsf{r1Enc}$ is a secure rate-1 message encoding scheme (Definition 3.14), then Construction 6.1 is an optimally efficient NIZK scheme for language $\mathcal{L}_{\mathcal{M}}$.

Remark 6.3. If we rely on a multi-theorem NIZK argument and plug-in the optimally efficient NIZK from Construction 6.1 in the reusable $\mathcal{E}\mathsf{Del}$ scheme (Definition 4.2), we get an optimal rate $\mathsf{r}\mathcal{E}\mathsf{Del}$ scheme.

Proof. We show that Construction 6.1 is correct and efficient as follows:

Correctness. We have that $\mathsf{crs} = (\lambda, \mathsf{del.crs})$ and $\pi = (\rho, \mathcal{E}\mathsf{Del.Prove}(\mathsf{del.crs}, (x, \rho), \mathsf{sk})$ where $(\rho, \mathsf{sk}) \leftarrow \mathsf{r1Enc.\ Encode}(1^\lambda, w)$. By correctness of $\mathcal{E}\mathsf{Del}, \mathsf{r1Enc}$ for machine $\mathcal{M}'$, if $\mathcal{M}(x, w) = 1$, then $\mathcal{M}'((x, \rho), \mathsf{sk}) = 1$ and the verification algorithm outputs 1.

Efficiency. Note that the crs of the NIZK scheme is nothing but the crs for $\mathcal{E}\mathsf{Del}$. Hence, $|\mathsf{crs}| = |\mathsf{del.crs}| = \mathsf{poly}(\lambda, \log T)$. Similarly, running time of Setup follows from running time of $\mathcal{E}\mathsf{Del.Setup}$'s running time. In Prove, we are using $\mathcal{E}\mathsf{Del.Prove}$ whose running time is $\mathsf{poly}(\lambda, T, (|x| + |w|))$ and $\mathsf{r1Enc}$ whose running time $\mathsf{poly}(\lambda, |w|)$. It follows that the running time of Prove is $\mathsf{poly}(\lambda, T, |x|, |w|)$. The size of the proof is $|\rho| + |\mathsf{del}.\pi| = |w| + \mathsf{poly}(\lambda, \log T, \log(|x| + |w|))$. The Verify algorithm computes using HT which takes $O((|x| + |w|)\mathsf{poly}(\lambda))$ time and then runs $\mathcal{E}\mathsf{Del.Verify}$. Hence the running time of Verify follows from the running time $\mathcal{E}\mathsf{Del.Verify}$ and $\mathsf{HT.Hash}$. In summary, Construction 6.1 has the following efficiency properties:

- $|\mathsf{crs}| = \mathsf{poly}(\lambda, \log T)$.
- $|\pi| = |w| + \mathsf{poly}(\lambda, \log T, \log(|x| + |w|))$.
- Running time of Setup is $\mathsf{poly}(\lambda, \log T)$.
- Running time of Prove is $\mathsf{poly}(\lambda, T, |x|, |w|)$.
- Running time of Verify is $O((|x| + |w|)\mathsf{poly}(\lambda)) + \mathsf{poly}(\lambda, \log T, \log(|x| + |w|))$.

Due to the poly-logarithmic dependence $|\mathsf{crs}|$ and Setup algorithm's running time T, we can parse λ many crs with exponentially increasing T, i.e, $T = 1, \ldots, T = 2^\lambda$ and use the crs that appropriately bounds the running time of $\mathcal{A}$. This is the popular powers-of-2 technique which is used in various contexts to reduce/remove the dependency of certain parameters in the analysis of algorithm which depend poly-logarithmically in the length of said parameters. We can leverage this technique to remove the dependence of T and we follow this convention for the rest of the paper. □

6.2.1 Security Analysis

Here we show that Construction 6.1 satisfies straight-line extraction and computational zero-knowledge.

Proof of straight-line extraction. We show that if $\mathcal{E}\mathsf{Del}$ satisfies straight-line extraction and $\mathsf{r1Enc}$ is correct, then Construction 6.1 satisfies straight-line extraction. Let the extractor do the following – (1) generate $(\mathsf{crs}, \mathsf{td})$ using $\mathcal{E}\mathsf{Del}.\mathcal{E}$. Set $\mathsf{crs} = \mathsf{del.crs}$ and $\mathsf{td} = \mathsf{del.td}$, (2) to extract w^* upon receiving $\pi = (\rho, \mathsf{del}.\pi)$, compute $\mathsf{sk}^* \leftarrow \mathcal{E}\mathsf{Del}.\mathcal{E}(\mathsf{del.td}, \mathsf{del}.\pi)$ and output $w^* = \mathsf{r1Enc.Decode}(1^\lambda, \rho, \mathsf{sk}^*)$.

Note that the crs indistinguishability is implied by the crs indistinguishability of $\mathcal{E}\mathsf{Del}$. We define the experiment $\mathsf{Exp}_{\mathsf{NIZK}}$ as follows –

$$\mathsf{Exp}_{\mathsf{NIZK}} = \left[\begin{array}{l} T \leftarrow \mathcal{A}(1^\lambda), (\mathsf{crs}, \mathsf{td}) \leftarrow \mathcal{E}(1^\lambda, T), (x, \pi) \leftarrow \mathcal{A}(\mathsf{crs}), \\ \text{Let } \pi = (\rho, \mathsf{del}.\pi), \mathsf{h}_{\mathsf{pb}} = \mathsf{HT.Hash}(\mathsf{ht.hk}, (x, \rho)), \\ \mathsf{sk}^* \leftarrow \mathcal{E}\mathsf{Del}.\mathcal{E}(\mathsf{td}, \pi), w^* \leftarrow \mathsf{r1Enc.Decode}(1^\lambda, \rho, \mathsf{sk}^*) \end{array}\right]$$

We say that $\mathcal{A}$ wins if

$$\Pr\left[\begin{array}{l} (x, w^*) \notin \mathcal{L}_{\mathcal{M}} \wedge \\ \mathcal{E}\mathsf{Del.Verify}(\mathsf{del.crs}, \mathsf{h}_{\mathsf{pb}}, \mathsf{del}.\pi) = 1 \end{array} : \mathsf{Exp}_{\mathsf{NIZK}}\right] \geq \epsilon(\lambda)$$

for some non-negligible function $\epsilon(\cdot)$. Then we define the following advantages of the adversary $\mathcal{A}$,

$$\mathsf{Adv}_1 = |\Pr[\mathcal{A} \text{ wins}] - \Pr[(x, w^*) \notin \mathcal{L}_{\mathcal{M}} \ \wedge ((x, \rho), \mathsf{sk}^*) \in \mathcal{L}_{\mathcal{M}'} : \mathsf{Exp}_{\mathsf{NIZK}}]| \tag{12}$$

$$\begin{aligned} \mathsf{Adv}_2 = |\Pr[(x, w^*) \notin \mathcal{L}_{\mathcal{M}} \ \wedge ((x, \rho), \mathsf{sk}^*) \in \mathcal{L}_{\mathcal{M}'} : \mathsf{Exp}_{\mathsf{NIZK}}] - \\ \Pr[(x, w^*) \notin \mathcal{L}_{\mathcal{M}} \ \wedge w^* = \mathsf{r1Enc.Decode}(1^\lambda, \rho, \mathsf{sk}^*) : \mathsf{Exp}_{\mathsf{NIZK}}\ ]| \end{aligned} \tag{13}$$

If $\mathcal{A}$ wins in $\mathsf{Exp}_{\mathsf{NIZK}}$, then either $\mathcal{A}$ breaks the straight-line extraction property of $\mathcal{E}\mathsf{Del}$ (Type 1, $\mathsf{Adv}_1 \geq \epsilon/2$) or it breaks the correctness of $\mathsf{r1Enc}$ (Type 2, $\mathsf{Adv}_2 \geq \epsilon/2$).

Type 1. Suppose Eq. (12) is correct, then it holds that

$$\Pr[((x, \rho), \mathsf{sk}^*) \notin \mathcal{L}_{\mathcal{M}'} \wedge \mathcal{E}\mathsf{Del.Verify}(\mathsf{del.crs}, \mathsf{h}_{\mathsf{pb}}, \mathsf{del}.\pi) = 1] \geq \epsilon/2$$

Now we construct an adversary $\mathcal{B}_{\mathcal{E}\mathsf{Del}.\mathcal{E}}$ that breaks the straight-line extraction property of $\mathcal{E}\mathsf{Del}$ for machine $\mathcal{M}'$ as follows:

1. Receive T from $\mathcal{A}$ and send it to $\mathcal{C}$.
2. Receive crs from $\mathcal{C}$ and pass it on to $\mathcal{A}$.
3. Receive (x, π) from $\mathcal{A}$. Parse π as $(\rho, \mathsf{del}.\pi)$. Send $((x, \rho), \mathsf{del}.\pi)$ to $\mathcal{C}$.

Note that $\mathcal{B}_{\mathcal{E}\mathsf{Del}.\mathcal{E}}$ perfectly simulates $\mathsf{Exp}_{\mathsf{NIZK}}$ for $\mathcal{A}$ thus having advantage $\epsilon/2$ in breaking the straight-line extraction for $\mathcal{E}\mathsf{Del}$.

Type 2. Suppose Eq. (13) is correct, note that $((x,\rho),\mathsf{sk}^*) \in \mathcal{L}_{\mathcal{M}'}$ implies that $w^* = \mathsf{r1Enc.Decode}(1^\lambda, \rho, \mathsf{sk}^*)$. However, by Eq. (13), $\mathcal{A}$ can find $w^* \neq \mathsf{r1Enc.Decode}(1^\lambda, \rho, \mathsf{sk}^*)$ with probability $\epsilon/2$, which breaks the perfect correctness of $\mathsf{r1Enc}$.

□

Proof of zero-knowledge. We show that if $\mathcal{E}\mathsf{Del}$ is input-hiding and $\mathsf{r1Enc}$ is secure, then Construction 6.1 satisfies computational zero-knowledge. We define the following hybrids:

- $\mathsf{hyb}_0^{\mathcal{A}}$ This is the original experiment. Namely, in this experiment, the adversary $\mathcal{A}$ sends T to the challenger $\mathcal{C}$. $\mathcal{C}$ runs $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, T)$ and sends crs to $\mathcal{A}$. $\mathcal{A}$ queries $\mathcal{C}$ with (x,w) and $\mathcal{C}$ responds with $\mathsf{Prove}(\mathsf{crs}, x, w)$ if $\mathcal{M}(x,w) = 1$.
- $\mathsf{hyb}_1^{\mathcal{A}}$ This is same as $\mathsf{hyb}_0^{\mathcal{A}}$ except that we use $\mathcal{E}\mathsf{Del}.\mathcal{S}$ to simulate $\mathcal{E}\mathsf{Del}$ instantiation.
- $\mathsf{hyb}_2^{\mathcal{A}}$ This is same as $\mathsf{hyb}_1^{\mathcal{A}}$ except we sample random $\rho \leftarrow \{0,1\}^m$ instead of $\mathsf{r1Enc.Encode}$. This is the description of the simulator $\mathcal{S}$.

Let $\mathsf{hyb}_i^{\mathcal{A}}(1^\lambda)$ denote the output of the experiment $\mathsf{hyb}_i^{\mathcal{A}}$.

Claim 6.4. If $\mathcal{E}\mathsf{Del}$ is an input-hiding encrypted RAM delegation scheme, then for any PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for every $\lambda \in \mathbb{N}$:

$$\left|\Pr\left[1 \leftarrow \mathsf{hyb}_0^{\mathcal{A}}(1^\lambda)\right] - \Pr\left[1 \leftarrow \mathsf{hyb}_1^{\mathcal{A}}(1^\lambda)\right]\right| \leq \mathsf{negl}(\lambda)$$

Proof. Assuming that there exists an adversary $\mathcal{A}$ which can distinguish between $\mathsf{hyb}_0^{\mathcal{A}}$ and $\mathsf{hyb}_1^{\mathcal{A}}$ such that $\left|\Pr\left[1 \leftarrow \mathsf{hyb}_0^{\mathcal{A}}(1^\lambda)\right] - \Pr\left[1 \leftarrow \mathsf{hyb}_1^{\mathcal{A}}(1^\lambda)\right]\right| = \epsilon(\lambda)$ for some non-negligible function $\epsilon(\cdot)$, we construct an adversary $\mathcal{B}_{\mathsf{del}}$ against the input-hiding property of $\mathcal{E}\mathsf{Del}$ as follows:

1. $\mathcal{A}$ sends T to $\mathcal{B}_{\mathsf{del}}$. $\mathcal{B}$ sends T to $\mathcal{C}$. $\mathcal{B}_{\mathsf{del}}$ passes the crs received from the challenger, $\mathcal{C}$, to $\mathcal{A}$.
2. $\mathcal{A}$ queries (x,w) to $\mathcal{B}_{\mathsf{del}}$. If $\mathcal{M}(x,w) = 1$, query $\mathcal{C}$ with $((x,\rho),\mathsf{sk})$ where $(\rho,\mathsf{sk}) \leftarrow \mathsf{r1Enc.Encode}(1^\lambda, w)$. Otherwise, respond with $\perp$.
3. Send $(\rho, \mathsf{del}.\pi)$ to $\mathcal{A}$ where $\mathsf{del}.\pi$ is send by $\mathcal{C}$. Output whatever $\mathcal{A}$ outputs.

If $\mathcal{C}$ uses $\mathcal{E}\mathsf{Del.Setup}$ and $\mathcal{E}\mathsf{Del.Prove}$ (resp. $\mathcal{E}\mathsf{Del}.\mathcal{S}$) in the experiment then $\mathcal{B}_{\mathsf{del}}$ perfectly simulates $\mathsf{hyb}_0^{\mathcal{A}}$ (resp. $\mathsf{hyb}_1^{\mathcal{A}}$). Thus $\mathcal{B}_{\mathsf{del}}$ has advantage $\epsilon(\lambda)$ in breaking the input-hiding property of $\mathcal{E}\mathsf{Del}$. □

Claim 6.5. If $\mathsf{r1Enc}$ is a secure rate-1 message encoding scheme, then for any PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for every $\lambda \in \mathbb{N}$:

$$\left|\Pr\left[1 \leftarrow \mathsf{hyb}_1^{\mathcal{A}}(1^\lambda)\right] - \Pr\left[1 \leftarrow \mathsf{hyb}_2^{\mathcal{A}}(1^\lambda)\right]\right| \leq \mathsf{negl}(\lambda)$$

Proof. Assuming that there exists an adversary $\mathcal{A}$ which can distinguish between $\mathsf{hyb}_1^{\mathcal{A}}$ and $\mathsf{hyb}_2^{\mathcal{A}}$ such that $\left|\Pr\left[1 \leftarrow \mathsf{hyb}_1^{\mathcal{A}}(1^\lambda)\right] - \Pr\left[1 \leftarrow \mathsf{hyb}_2^{\mathcal{A}}(1^\lambda)\right]\right| = \epsilon(\lambda)$ for some non-negligible function $\epsilon(\cdot)$, we construct an adversary $\mathcal{B}_{\mathsf{r1Enc}}$ that breaks the security of $\mathsf{r1Enc}$ as follows:

1. $\mathcal{A}$ sends T to $\mathcal{B}_{\mathsf{r1Enc}}$. $\mathcal{B}$ samples $\mathsf{del.crs} \leftarrow \mathcal{E}\mathsf{Del}.\mathcal{S}(1^\lambda, 1^\lambda, T)$ and sends $\mathsf{del.crs}$ to $\mathcal{A}$.

2. $\mathcal{A}$ queries (x, w). If $\mathcal{M}(x, w) = 1$, query the challenger, $\mathcal{C}$, with w to receive ρ and send $(\rho, \mathsf{del}.\pi)$ where $\mathsf{del}.\pi \leftarrow \mathcal{E}\mathsf{Del}.\mathcal{S}(\mathsf{del}.\mathsf{crs}, x, \rho)$ to $\mathcal{A}$. Otherwise, output $\perp$.

3. Output whatever $\mathcal{A}$ outputs.

If $\mathcal{C}$ uses $\mathsf{r1Enc}.\mathsf{Encode}$ (resp. using random ρ) in the experiment then $\mathcal{B}_{\mathsf{r1Enc}}$ perfectly simulates $\mathsf{hyb}_1^{\mathcal{A}}$ (resp. $\mathsf{hyb}_2^{\mathcal{A}}$). Thus $\mathcal{B}_{\mathsf{r1Enc}}$ has advantage $\epsilon(\lambda)$ in breaking the security of $\mathsf{r1Enc}$. □

□

We state the following corollary of Theorems 6.2, 5.2 and Remarks 3.12, 3.10.

Corollary 6.6. Assuming $X \in \{\mathsf{LWE}, O(1)\text{-}\mathsf{LIN}, \text{sub-exp-}\mathsf{DDH}\}$, there exists an optimally efficient non-interactive zero-knowledge argument scheme with straight-line extraction for all NP languages.

6.3 Boolean Circuit Decomposition

In the next section, we will provide our construction for fully-homomorphic NIZKs. But first, we define a new circuit decomposition algorithm $\mathsf{Decompose}$, which plays an important role of achieving the unlinkability property. At a high level, by splitting each cuttable vertex of G (which corresponds to circuit C), $\mathsf{Decompose}(C) \to (T, \{C_v\}_{v \in V_T})$ parses circuit C into smallest chunks of circuits $\{C_v\}_{v \in V_T}$ such that their combination forms a tree structure $T = (V_T, V_E)$. Fig. 3 is an example of the circuit decomposition algorithm.

$\mathsf{Decompose}(C) \to (T, \{C_v\}_{v \in V_T})$.

1. Let directed acyclic graph $G = (V, E)$ represent the boolean circuit $C : \{0,1\}^k \to \{0,1\}$.

2. For each cuttable node $u \in V$ of graph G, the $\mathsf{Decompose}$ algorithm simply splits node u as u, u', and labels u as an output gate and u' as an input node. After splitting all of the cuttable nodes, it obtains a new graph G'.

3. Next, each connected component of graph G' can be represented as a boolean circuit. Consider each of these boolean circuits as a vertex. Then these vertices will naturally form a tree $T = (V_T, E_T)$, where for each $u, v \in V_T$, u is the parent of v if and only if the output gate of circuit C_v and the input gate of circuit C_u are splits of a single vertex in the previous step.

4. The $\mathsf{Decompose}$ algorithm outputs $T, \{C_v\}_{v \in V_T}$.

Theorem 6.7. Given any set of boolean circuits $\{C, C_1, \ldots, C_\ell\}$ where $C = \{0,1\}^\ell \to \{0,1\}$, let $(T_0, \{C_v\}_{v \in V_{T_0}})$ be the decomposition by $\mathsf{Decompose}(C)$, and for each $i \in [\ell]$, let $(T_i, \{C_v\}_{v \in V_{T_i}})$ be $\mathsf{Decompose}(C_i)$. Define C' as $\mathsf{Compose}(C, (C_1, \ldots, C_\ell))$, and let $(T', \{C_v\}_{v \in V_{T'}})$ be $\mathsf{Decompose}(C')$. Then,

$$\bigcup_{i=0}^{\ell} \{C_v\}_{v \in V_{T_i}} = \{C_v\}_{v \in V_{T'}},$$

Proof. We note that on composing $C, \{C_i\}_{i \in [\ell]}$, one is merging the output gate of C_i with the input gate of circuit C and make it a single node in the composed circuit C'. Such node must be a cuttable node, since the node is the only node connecting circuit C and C_i. Thus the decomposer splits all such nodes in C', and our theorem directly follows. □

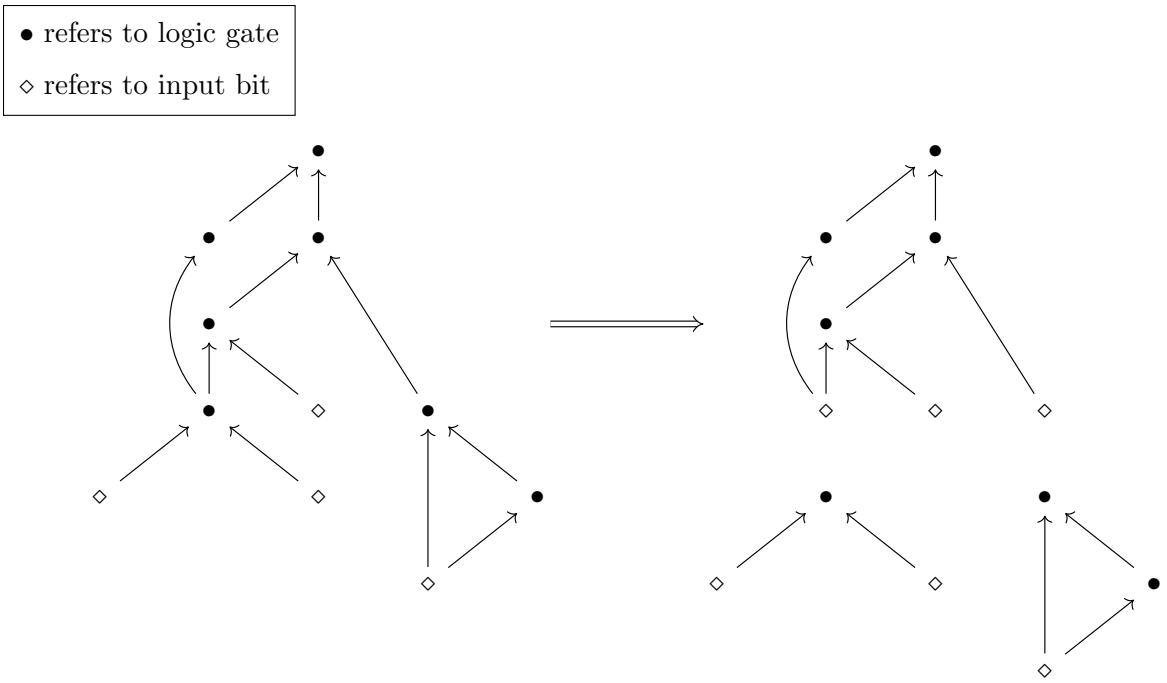

Figure 3: Demonstration of Decompose for 'cuttable' nodes

6.4 Constructing Homomorphic NIZKs

As explained in Section 2, in order to ensure unlinkability property, we make the 'behavior' of Prove and Eval algorithms of our homomorphic NIZK scheme to 'align' with each other. Expanding on this further, consider circuits $\{C_i : \{0,1\}^{\ell_i} \to \{0,1\}\}_{i\in[k]}, C : \{0,1\}^k \to \{0,1\}$. Furthermore, define the circuit $C' = \{0,1\}^{\sum_{i=1}^{k}\ell_i} \to \{0,1\}$ as $C' = \mathsf{Compose}(C, C_1, \ldots, C_k)$. Consider the witnesses $w_i \in \{0,1\}^{\ell}$ for each $i \in [k]$, output bits $b_1, \ldots, b_k, b'$ such that $C_i(\omega_i) = b_i$ and $C'(\omega_1, \ldots, \omega_k) = C(b_1, \ldots, b_k) = b'$. A valid homomorphic NIZK proof π' for circuit satisfiability instance (C', b') (Eq. (11)) could be either generated from $\mathsf{Prove}(\mathsf{crs}, (C', b'), \{\omega_i\}_{i\in[k]})$ algorithm taking (C', b') as instance and $\{\omega_i\}_{i\in[k]}$ as witness, or homomorphically evaluated from $\mathsf{Eval}(\mathsf{crs}, C, \{(C_i, b_i, \pi_i)\}_{i\in[k]})$ as $C' = \mathsf{Compose}(C, C_1, \ldots, C_k)$ and $C_i(\omega_i) = b_i$. To achieve unlinkability property, a freshly generated proof must be indistinguishable from any evaluated proof. As discussed in Section 2, the naive approach won't work as one could not argue unlinkability since a freshly generated NIZK proof is from the language

$$\exists \omega_1, \ldots, \omega_k, \text{ s.t. } C'(\omega_1, \ldots, \omega_k) = b',$$

whereas evaluated proof would be for the language

$$\forall i \in [k], \exists \pi_i \text{ s.t. } \mathsf{Verify}(\mathsf{crs}, (C_i, b_i), \pi_i) = 1 \text{ and } C(b_1, \ldots, b_k) = b'.$$

The main idea is that we let the prover compute and decompose the circuit C' into the smallest atomic circuits and generate the fresh homomorphic NIZK proof π by performing algorithm Eval level by level in the resultant tree, i.e., following the topological order of those atomic circuits obtained by the Decompose algorithm. As a result, a freshly generated proof and a evaluated proof

Language $\mathcal{L}$

Instance: $\mathsf{nizk.crs}, C, b$
Witness: $\{(b_i, \pi_i)\}_{i\in[k]}$.
Output: To check validity of the instance-witness pair, proceed as follows:

- It runs $\mathsf{Decompose}(C)$ and obtains $(T, \{C_v\}_{v\in V_T})$.
- Let Rt be the root node of T and check if C_{Rt} takes k bits of input. If the check fails, it outputs 0 and aborts.
- Let $v_1, \ldots, v_k$ be the k child nodes of Rt in T. For each $i \in [k]$, let circuit C_i be the circuit defined by the subtree with root v_i (note that we are combining all the circuits within that subtree).
- For each $i \in [k]$, if C_i is not empty, it checks if $\mathsf{NIZK.Verify}(\mathsf{nizk.crs}, (\mathsf{nizk.crs}, C_i, b_i), \pi_i) = 1$. If any of the checks fails, it outputs 0 and aborts.
- It outputs 1 if and only if $C_{\mathsf{Rt}}(b_1, \ldots, b_k) = b$.

Figure 4: Description of language $\mathcal{L}$ for optimally efficient NIZK

would be generated by the underlying optimally efficient NIZK scheme from a universal language. The unlinkability property immediately holds by the zero-knowledge property of optimally efficient NIZK. Our construction is formally described below.

Construction 6.8. Let $\mathsf{NIZK} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ be an optimally efficient NIZK scheme (Definition 3.13) for language $\mathcal{L}$ defined in Fig. 4. We construct a homomorphic NIZK scheme for language from Eq. (11) as follows:

$\mathsf{Setup}(1^\lambda) \to \mathsf{crs}$**.** The Setup algorithm generates crs with respect to language $\mathcal{L}$ as $\mathsf{nizk.crs} \leftarrow \mathsf{NIZK.Setup}(1^\lambda)$ and outputs $\mathsf{crs} = \mathsf{nizk.crs}$.

$\mathsf{Prove}(\mathsf{crs}, (C, b), \omega) \to \pi$**.** Parse crs as $\mathsf{nizk.crs}$ and witness ω as $(b_1, \ldots, b_\ell) \in \{0,1\}^\ell \ \forall\ i \in [\ell]$, the prover's algorithm sets C_i as an empty circuit, π_i as an empty string. It runs $(C', b', \pi') \leftarrow \mathsf{Eval}(\mathsf{nizk.crs}, C, \{(C_i, b_i, \pi_i)\}_{i\in[\ell]})$ and outputs proof π as π'.

$\mathsf{Eval}(\mathsf{crs}, C, \{(C_i, b_i, \pi_i)\}_{i\in[k]}) \to (C', b, \pi)$**.** The algorithm Eval parses crs as $\mathsf{nizk.crs}$ and does the following:

1. Set $(T, \{C_v\}_{v\in V_T})$ to be $\mathsf{Decompose}(C)$.
2. For i-th leaf node $v \in V_T$, set $C'_v = C_i$, b_v as b_i, π_v as π_i.
3. Sort all the internal nodes in T in a topological order (ending with the root) and iteratively do the following on all levels starting from the leaves:
   - If $v \in V_T$ has an in-degree ℓ, let $(v_1, \ldots, v_\ell)$ be the child nodes of v, i.e., C_v takes $(b_{v_1}, \ldots, b_{v_\ell})$ as input. Then compute

     $$b_v = C_v(b_{v_1}, \ldots, b_{v_\ell}), C'_v = \mathsf{Compose}(C_v, (C'_{v_1}, \ldots, C'_{v_\ell})),$$

     and generate $\pi_v \leftarrow \mathsf{NIZK.Prove}(\mathsf{nizk.crs}, (\mathsf{nizk.crs}, C'_v, b_v), \{(b_{v_i}, \pi_{v_i})\}_{i\in[\ell]})$.
4. Let node Rt be the root of T. Output $C' = C'_{\mathsf{Rt}}, b = b_{\mathsf{Rt}}$, and $\pi = \pi_{\mathsf{Rt}}$.

$\mathsf{Verify}(\mathsf{crs}, (C, b), \pi) \to 0/1$**.** Parse crs as $\mathsf{nizk.crs}$ and output $\mathsf{NIZK.Verify}(\mathsf{nizk.crs}, (\mathsf{nizk.crs}, C, b), \pi)$.

Theorem 6.9. If NIZK is a optimally efficient NIZK scheme (Definition 3.13) for language $\mathcal{L}$ (Fig. 4), then Construction 6.8 is a homomorphic NIZK scheme for the NP language defined in Eq. (11).

Completeness. The design described above implements a recursive zero-knowledge proof over the tree of the decomposed circuit. The functionality of the decomposed circuit is equivalent to the original circuit, and the completeness of the above design is implied by the completeness of the underlying NIZK.

Efficiency. The succinctness of the above design follows from the optimal efficiency property of the underlying NIZK which is specified by the efficiency of optimally efficient NIZK.

Lemma 6.10. Assume that NIZK satisfies optimal efficiency property, then for every $\lambda \in \mathbb{N}$, the following holds:

- For $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda)$, $|\mathsf{crs}| \leq \mathsf{poly}(\lambda)$.
- For boolean circuit $C : \{0,1\}^k \to \{0,1\}$, the running time of $\mathsf{Prove}(\mathsf{crs}, C, b, \{b_i\}_{i\in[k]})$ is at most $\mathsf{poly}(\lambda, |C|)$.
- For proof $\pi \leftarrow \mathsf{Prove}(\mathsf{crs}, C, b, \{b_i\}_{i\in[k]})$, $|\pi| \leq \mathsf{poly}(\lambda, |C|)$.

Proof. By optimal efficiency of NIZK, the crs size is at most $\mathsf{poly}(\lambda)$ since the running time of the above language $\mathcal{L}$ is no more than 2^λ. Next we consider the running time of Prove, observe that language $\mathcal{L}$ contains a $\mathsf{NIZK.Verify}$ step. Again by efficiency of optimally efficient NIZK, the NIZK verifier running time is bounded by $\mathsf{poly}(\lambda, |x|, |\omega|)$ for some $\mathsf{poly}(\cdot,\cdot,\cdot)$. Thus the size of language $\mathcal{L}$ is at most $\mathsf{poly}(\lambda, |C|)$. By the optimal efficiency of NIZK, there is a $\mathsf{poly}(\lambda)$-bit overhead of the zero knowledge proof. Otherwise, the proof size is exactly equivalent to the witness size. We observe that the maximum number of NIZK proofs applied is fewer than the number of nodes in the $\mathsf{Decompose}(C)$ tree. Thus, the overall overhead size is bounded by $\mathsf{poly}(\lambda, |C|)$. Since the overall witness length $k \leq |C|$, we conclude that $|\pi| \leq \mathsf{poly}(\lambda, |C|)$. □

6.5 Security Analysis

Lemma 6.11 (Argument of knowledge)**.** Assume that NIZK satisfies straight-line extraction, then Construction 6.8 satisfies argument of knowledge.

Proof. We will show how to construct an extractor $\mathcal{E}$ for our scheme. Before doing that, we start by defining a recursive extractor $\mathcal{E}'$ for our scheme.

$\mathcal{E}'(1^\lambda) \to (\mathsf{crs}, \mathsf{td})$**:** The extractor first sets up NIZK CRS and trapdoor with respect to language $\mathcal{L}$:

$$(\mathsf{nizk.crs}, \mathsf{nizk.td}) \leftarrow \mathsf{NIZK}.\mathcal{E}(1^\lambda).$$

Next, it outputs crs as $\mathsf{nizk.crs}$ and td as $(\mathsf{nizk.crs}, \mathsf{nizk.td})$.

$\mathcal{E}'(\mathsf{td}, (C, b), \pi) \to (T, \{(C'_v, b_v, \omega_v)\}_{v\in V_T})$. Below is a step-by-step explanation:

1. It first computes $(T, \{C_v\}_{v \in V_T}) = \mathsf{Decompose}(C)$. Next for each leaf node v, it sets $C'_v = C_v$. For each non-leaf node v with child nodes $(v_1, \ldots, v_\ell)$, it sets $C'_v = \mathsf{Compose}(C_v, (C'_{v_1}, \ldots, C'_{v_\ell}))$. It sets C'_v for each $v \in V_T$ in a topological order.

2. Let Rt be the root node of T, then $C'_{\mathsf{Rt}} = C$ by definition of step 1. It sets π_{Rt} as π and b_{Rt} as b. It parses td as $(\mathsf{nizk.crs}, \mathsf{nizk.td})$, and we note that crs is set as $\mathsf{nizk.crs}$ in our scheme.

3. Then for each internal node $v \in V_T$, consider that it has been assigned π_v, b_v and that C_v takes ℓ bits of input. The extractor runs $\mathsf{NIZK}.\mathcal{E}(\mathsf{nizk.td}, (\mathsf{nizk.crs}, C'_v, b_v), \pi_v)$ and obtains $\{(b_i, \pi_i)\}_{i \in [k]}$. It sets ω_v as $(b_1, \ldots, b_\ell)$. Since C'_v takes ℓ bits of input, v also has a number of ℓ child nodes $(v_1, \ldots, v_\ell)$. It sets b_{v_i} as b_i and π_{v_i} as π_i for all $i \in [\ell]$.

4. It runs step 3 for each non-leaf node $v \in V_T$ in a reverse topological order. For each leaf node v, it sets ω_v as an empty string. It outputs T and $\{(C'_v, b_v, \omega_v)\}_{v \in V_T}$.

Claim 6.12. Assuming that NIZK satisfies adaptive argument of knowledge, then there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, the following holds for all non-leaf node $v \in V_T$:

$$\Pr\left[\begin{array}{l|l} \mathsf{Verify}(\mathsf{crs}, (C, b), \pi) = 1 & (\mathsf{crs}, \mathsf{td}) \leftarrow \mathcal{E}'(1^\lambda) \\ \wedge\, \omega_v \text{ is not a valid witness for } (\mathsf{crs}, C'_v, b_v) \in \mathcal{L} & (C, b, \pi) \leftarrow \mathcal{A}(\mathsf{crs}, \mathsf{td}) \\ & (T, \{(C'_v, b_v, \omega_v)\}_{v \in V_T}) \leftarrow \mathcal{E}'(\mathsf{td}, (C, b), \pi) \end{array}\right] \leq \mathsf{negl}(\lambda),$$

where in the above equation $T = (V_T, E_T)$.

Proof. We construct a proof by induction over each $v \in V_T$, in a reverse topological order.

Base Case ($v = \mathsf{Rt}$)**:** We first show that the lemma holds for $v = \mathsf{Rt}$. Assume that $\mathsf{Verify}(\mathsf{crs}, (C, b), \pi) = 1$ and ω_{Rt} is not a valid witness for $(\mathsf{crs}, C'_{\mathsf{Rt}}, b_{\mathsf{Rt}}) \in \mathcal{L}$, where $\omega_{\mathsf{Rt}} \leftarrow \mathsf{NIZK}.\mathcal{E}(\mathsf{nizk.td}, (\mathsf{nizk.crs}, C'_{\mathsf{Rt}}, b_{\mathsf{Rt}}), \pi_{\mathsf{Rt}})$. Assume that there exists a PPT adversary $\mathcal{A}$ that breaks the security property in our lemma, we build a reduction algorithm $\mathcal{B}$ that breaks the proof of knowledge property of NIZK.

$\mathcal{A}$ starts by setting language $\mathcal{L}$ and security parameter λ. Reduction algorithm $\mathcal{B}$ then queries the NIZK challenger with 1^λ. The challenger outputs $\mathsf{nizk.crs}, \mathsf{nizk.td}$. $\mathcal{B}$ sets $\mathsf{crs} = \mathsf{nizk.crs}$, $\mathsf{td} = (\mathsf{nizk.crs}, \mathsf{nizk.td})$ and sends $\mathsf{crs}, \mathsf{td}$ to $\mathcal{A}$. $\mathcal{A}$ outputs (C, b, π).

Consider that $\mathcal{E}'(\mathsf{td}, (C, b), \pi)$ outputs tree $T = (V_T, E_T)$ and $\{(C'_v, b_v, \omega_v)\}_{v \in V_T}$. Let Rt be the root of T. Then, based on our design of $\mathcal{E}'$, ω_{Rt} is extracted using $\mathsf{NIZK}.\mathcal{E}(\mathsf{nizk.td}, (\mathsf{crs}, C'_{\mathsf{Rt}}, b_{\mathsf{Rt}}), \pi_{\mathsf{Rt}})$. Note that $C'_{\mathsf{Rt}} = C$, $b_{\mathsf{Rt}} = b$ and $\pi_{\mathsf{Rt}} = \pi$, thus $\mathsf{Verify}(\mathsf{crs}, (C, b), \pi) = 1$ implies that $\mathsf{NIZK.Verify}(\mathsf{nizk.crs}, (\mathsf{crs}, C'_{\mathsf{Rt}}, b_{\mathsf{Rt}}), \pi_{\mathsf{Rt}}) = 1$. $\mathcal{B}$ could break the proof of knowledge property by outputting $(\mathsf{crs}, C'_{\mathsf{Rt}}, b_{\mathsf{Rt}})$ and π_{Rt}.

Inductive Step for non-leaf node v**:** Suppose that the lemma holds for every node that comes before node v in the reverse topological order. We show that the lemma also holds for v. Assume that for some PPT adversary $\mathcal{A}$, $\mathsf{Verify}(\mathsf{crs}, (C, b), \pi) = 1$ but ω_v is not a valid witness for $(\mathsf{crs}, C'_v, b_v) \in \mathcal{L}$ where $\omega_v \leftarrow \mathsf{NIZK}.\mathcal{E}(\mathsf{nizk.td}, (\mathsf{crs}, C'_v, b_v), \pi_v)$. We build a reduction algorithm $\mathcal{B}$ that breaks the proof of knowledge property of NIZK.

$\mathcal{A}$ sets $\mathcal{L}$ and λ. $\mathcal{B}$ queries the NIZK challenger with 1^λ and the challenger responses with $\mathsf{nizk.crs}, \mathsf{nizk.td}$. $\mathcal{B}$ sets and outputs $\mathsf{crs} = \mathsf{nizk.crs}$ and $\mathsf{td} = (\mathsf{nizk.crs}, \mathsf{nizk.td})$. $\mathcal{A}$ then outputs (C, b, π). Let u be the parental node of v. Then by the above inductive hypothesis, with all but negligible probability, ω_u is a valid witness for $(\mathsf{crs}, C'_u, b_u) \in \mathcal{L}$, which already implies that

$\mathsf{NIZK.Verify}(\mathsf{nizk.crs}, (\mathsf{nizk.crs}, C'_v, b_v), \pi_v) = 1$. If ω_v is not a valid witness, then $\mathcal{B}$ breaks the argument of knowledge property of nizk by extracting and outputting $(\mathsf{crs}, C'_v, b_v)$. □

The recursive extractor $\mathcal{E}'$ falls short as a sufficient extractor, as it only produces the instance and witness for individual nodes. We introduce the actual extractor $\mathcal{E}$. $\mathcal{E}$ takes as input (C, b, π), where circuit C takes k input bits. Ideally, $\mathcal{E}$ outputs a valid witness $(b_1, \ldots, b_k)$ satisfying $C(b_1, \ldots, b_k) = b$.

$\mathcal{E}(\mathsf{td}, (C, b), \pi) \to (b_1, \ldots, b_k)$. It runs $\mathcal{E}'(\mathsf{td}, (C, b), \pi)$ and obtains $T, \{(C'_v, b_v, \omega_v)\}_{v \in V_T}$. Next for all $i \in [k]$, let v_i be the i-th leaf node, it outputs $(b_{v_1}, \ldots, b_{v_k})$.

Let $\{C_v\}_{v \in V_T}$ be the output of $\mathsf{Decompose}(C)$. We note that by Claim 6.12, for each internal node v with child nodes $(v_1, \ldots, v_\ell)$, it holds that $C_v(b_{v_1}, \ldots, b_{v_\ell}) = 1$. Since circuits $\{C_v\}_{v \in V_T}$ is a decomposition of C, we have $C(b_{v_1}, \ldots, b_{v_k}) = 1$. Our theorem follows. □

Lemma 6.13 (Unlinkability)**.** Assume that NIZK satisfies zero-knowledge property, then Construction 6.8 satisfies unlinkability.

Proof. Before proceeding to the proof, we define the term atomic circuit: A circuit C is considered atomic if output of $\mathsf{Decompose}(C)$ includes the circuit C itself. We argue the following: For any set of boolean circuits $C, C_1, \ldots, C_\ell$, where $C' = \mathsf{Compose}(C, C_1, \ldots, C_\ell)$, the output by $\mathsf{Prove}(\mathsf{crs}, C', b, \{b_i\}_{i \in [k]})$ is indistinguishable from $\mathsf{Eval}(\mathsf{crs}, C, \{(C_i, b_i, \pi_i)\}_{i \in [k]})$, thereby unlinkability holds. By Theorem 6.7, whether we decompose $C, C_1, \ldots, C_\ell$ or directly decompose C', the resulting atomic circuit sets are the same. Thus by definition of algorithm $\mathsf{Compose}$, the circuits C'_{Rt} at the roots of both $\mathsf{Prove}(\mathsf{crs}, C', b, \{b_i\}_{i \in [k]})$ and $\mathsf{Eval}(\mathsf{crs}, C, \{(C_i, b_i, \pi_i)\}_{i \in [k]})$ are equivalent to C'. Furthermore, the output proof $\pi = \pi_{\mathsf{Rt}}$ at the final hop by both Prove and Eval is generated using the zero-knowledge proof system NIZK under the same instance $(\mathsf{crs}, C'_{\mathsf{Rt}}, b_{\mathsf{Rt}})$. Since language $\mathcal{L}$ is deterministic, given a fixed instance, our design already satisfies unlinkability by the zero-knowledge property of NIZK. □

Lemma 6.14 (Zero knowledge)**.** Assume that NIZK satisfies zero-knowledge property, then Construction 6.8 satisfies zero-knowledge property.

Proof. Given instance (C, b), the proof π by our design is itself a NIZK proof π_{Rt} under instance $(C'_{\mathsf{Rt}}, b_{\mathsf{Rt}})$ where $C'_{\mathsf{Rt}} = C$ and $b_{\mathsf{Rt}} = b$. Thus our design immediately satisfies zero-knowledge property. □

We close the section with the following corollaries, which are immediate by Corollary 6.6.

Corollary 6.15. Assuming $X \in \{\mathsf{LWE}, O(1)\text{-}\mathsf{LIN}, \text{sub-exp-}\mathsf{DDH}\}$, there exists a homomorphic NIZK scheme for any NP language.

7 Complexity-Preserving Semi-Malicious to Malicious Compiler for mrNISC

In this section, we provide the definition, construction, and analysis of a maliciously secure multi-party non-interactive secure computation scheme (mrNISC) for n parties using a semi-maliciously secure mrNISC scheme for n parties and reusable Encrypted RAM Delegation scheme ($\mathsf{r\mathcal{E}Del}$) w.r.to HT.

7.1 Definition

Syntax. A multi-party reusable non-interactive secure computation scheme for n parties (NISC) consists for the following polynomial time algorithms:

$\mathsf{Setup}(1^\lambda) \to \mathsf{crs}$**.** The probabilistic setup algorithm takes as input the security parameter and outputs the common random string crs.

$\mathsf{InpEnc}(\mathsf{crs}, \mathsf{id}, x; r) \to (\widehat{x}, \tau)$**.** The probabilistic input encoding algorithm takes as input a crs, identity of the party id, input x, randomness r, and outputs the public encoding of x, $\widehat{x}$ and trapdoor τ.

$\mathsf{FuncEnc}(\mathsf{crs}, \mathsf{id}, f, \{\widehat{x}_{\mathsf{idx}}\}_{\mathsf{idx}\in I}, \tau_{\mathsf{id}}) \to \widehat{f}_{\mathsf{id}}$**.** The function encoding algorithm takes as input a crs, identity of the party id, description of function f, public encodings for all $\mathsf{idx} \in I \subseteq [n]$, $\{\widehat{x}_{\mathsf{idx}}\}_{\mathsf{idx}}$ ($\mathsf{id} \in I$), trapdoor for the id-th party τ_{id}, and outputs the function encoding for the id-th party, $\widehat{f}_{\mathsf{id}}$.

$\mathsf{Eval}(\mathsf{crs}, f, \{\widehat{f}_{\mathsf{id}}\}_{\mathsf{id}\in I}, \{\widehat{x}_{\mathsf{id}}\}_{\mathsf{id}\in I}) \to y$**.** The evaluation algorithm takes as input a crs, the function encodings, $\{\widehat{f}_{\mathsf{id}}\}_{\mathsf{id}}$ and input encodings $\{\widehat{x}_{\mathsf{id}}\}_{\mathsf{id}}$ for all $\mathsf{id} \in I$, and outputs a value y.

Definition 7.1. An NISC scheme for n parties $(\mathsf{Setup}, \mathsf{InpEnc}, \mathsf{FuncEnc}, \mathsf{Eval})$ is required to satisfy the following properties:

Correctness. For any $\lambda \in \mathbb{N}$, $n = n(\lambda)$, $I \subseteq [n]$, efficient computation f, we have:

$$\Pr\left[\begin{array}{ll} f(\{x_{\mathsf{id}}\}_{\mathsf{id}\in I}) = & \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda), \\ \mathsf{Eval}(\mathsf{crs}, f, \{\widehat{f}_{\mathsf{id}}\}_{\mathsf{id}\in I}, & : \ \forall \mathsf{id} \in I, (\widehat{x}_{\mathsf{id}}, \tau_{\mathsf{id}}) \leftarrow \mathsf{InpEnc}(\mathsf{crs}, \mathsf{id}, x_{\mathsf{id}}; r_{\mathsf{id}}), \\ \{\widehat{x}_{\mathsf{id}}\}_{\mathsf{id}\in I}) & \widehat{f}_{\mathsf{id}} \leftarrow \mathsf{FuncEnc}(\mathsf{crs}, \mathsf{id}, f, \{\widehat{x}_{\mathsf{idx}}\}_{\mathsf{idx}\in I}, \tau_{\mathsf{id}}) \end{array}\right] = 1$$

Static Corruptions, Semi-Malicious, Adaptive Security. For any $\lambda \in \mathbb{N}$, any $n = n(\lambda)$, any $H \subseteq [n]$, $|H| \geq 1$ and admissible adversary $\mathcal{A}$, there exists a stateful simulator $\mathcal{S}$, a negligible function $\mathsf{negl}(\cdot)$ such that:

$$\left|\Pr\left[1 \leftarrow \mathsf{Expt}_0^{\mathcal{C},\mathcal{A}}(1^\lambda)\right] - \Pr\left[1 \leftarrow \mathsf{Expt}_1^{\mathcal{S},\mathcal{A}}(1^\lambda)\right]\right| \leq \mathsf{negl}(\lambda)$$

where the definitions of admissible adversary, $\mathsf{Expt}_0^{\mathcal{C},\mathcal{A}}$ and $\mathsf{Expt}_1^{\mathcal{S},\mathcal{A}}$ are provided in Figure 5. If the adversary instead behaves maliciously, we say the scheme is maliciously secure.

Remark 7.2. A semi-maliciously secure mrNISC scheme's crs can be empty (or trivial). For malicious security, mrNISC scheme has to be in the crs model for PPT simulation security.

Theorem 7.3 ([BL20, BJKL21, AJJM21, Shi22])**.** Assuming $X \in \{\mathsf{SXDH}, \mathsf{LWE}\}$, there exists a mrNISC scheme for n parties for all efficient functions that is semi-maliciously secure.

Admissible Adversary. An admissible adversary is a stateful PPT machine that queries for a single input encoding from all parties (corrupt and honest) and unbounded functional encodings.

$\mathsf{Expt}_0^{\mathcal{C},\mathcal{A}}(1^\lambda)$**.** This is the Real experiment parameterized by an honest challenger $\mathcal{C}$. $\mathcal{A}$ submits the set of honest parties H and receives $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda)$ from $\mathcal{C}$. $\mathcal{A}$ makes the following queries in an adaptive manner. After this, output $\mathcal{A}$'s guess b'.

- Honest Input Encoding: $\mathcal{A}$ sends $(\mathsf{id}, x_{\mathsf{id}})$ for $\mathsf{id} \in H$ and receives $\widehat{x}_{\mathsf{id}}$ where $(\widehat{x}_{\mathsf{id}}, \tau_{\mathsf{id}}) \leftarrow \mathsf{InpEnc}(\mathsf{crs}, \mathsf{id}, x_{\mathsf{id}}; r_{\mathsf{id}})$ for some randomness $r_{\mathsf{id}} \leftarrow \{0,1\}^\lambda$.
- Corrupt Input Encoding: $\mathcal{A}$ sends $(\mathsf{id}, x_{\mathsf{id}}, r_{\mathsf{id}})$ for $\mathsf{id} \notin H$ to $\mathcal{C}$.
- Function Encoding: $\mathcal{A}$ sends (id, f, I) for $\mathsf{id} \in H$ such that input encodings for all $\mathsf{idx} \in I$ are queried, and receives $\widehat{f}_{\mathsf{id}} \leftarrow \mathsf{FuncEnc}(\mathsf{crs}, \mathsf{id}, f, \{\widehat{x}_{\mathsf{idx}}\}_{\mathsf{idx}\in I}, \tau_{\mathsf{id}})$.

$\mathsf{Expt}_1^{\mathcal{S},\mathcal{A}}(1^\lambda)$**.** This is the Ideal experiment parameterized by a stateful simulator $\mathcal{S}$. $\mathcal{A}$ submits the set of honest parties H and receives $\mathsf{crs} \leftarrow \mathcal{S}(1^\lambda)$. $\mathcal{A}$ makes the following queries in an adaptive manner. After this, output $\mathcal{A}$'s guess b'.

- Honest Input Encoding: $\mathcal{A}$ sends $(\mathsf{id}, x_{\mathsf{id}})$ for $\mathsf{id} \in H$ and receives $\widehat{x}_{\mathsf{id}}$ where $\widehat{x}_{\mathsf{id}} \leftarrow \mathcal{S}(\mathsf{id})$.
- Corrupt Input Encoding: $\mathcal{A}$ sends $(\mathsf{id}, x_{\mathsf{id}}, r_{\mathsf{id}})$ for $\mathsf{id} \notin H$.
- Function Encoding: $\mathcal{A}$ sends (id, f, I) for $\mathsf{id} \in H$ such that input encodings for all $\mathsf{idx} \in I$ are queried. If $\forall\ \mathsf{idx} \in I \cap H, \mathsf{idx} \neq \mathsf{id}$, function encodings are queried, set $y = f(\{x_{\mathsf{id}}\}_{\mathsf{id}\in I})$. Otherwise, set $y = \bot$. Send $\widehat{f}_{\mathsf{id}} \leftarrow \mathcal{S}(\mathsf{id}, f, I, y)$ to $\mathcal{A}$.

Figure 5: Real and Ideal experiments for security of SMSC

7.2 Construction

Construction 7.4 (Malicious NISC)**.** Let $\mathsf{r1Enc} = (\mathsf{Encode}, \mathsf{Decode})$ be a secure rate-1 message encoding scheme (Definition 3.14) and $\mathsf{SMSC} = (\mathsf{InpEnc}, \mathsf{FuncEnc}, \mathsf{Eval})$, be a semi-maliciously secure mrNISC scheme (Definition 7.1). Using a reusable encrypted RAM delegation scheme $\mathsf{r}\mathcal{E}\mathsf{Del} = (\mathsf{Setup}, \mathsf{PrivEnc}, \mathsf{Prove}, \mathsf{VerifyEnc}, \mathsf{Verify})$ w.r.to HT for language $\mathcal{L}^{\mathsf{NISC}}_{\mathcal{M},T} = \{(x_{\mathsf{pb}}, x_{\mathsf{pr}}, T) : \mathcal{M}_{\mathsf{NISC}}(x_{\mathsf{pb}}, x_{\mathsf{pr}}, T) = 1\}$ where $\mathcal{M}_{\mathsf{NISC}}$ is defined in Figure 7.4, we provide the construction of an maliciously secure mrNISC scheme NISC for n parties for the function f below.

RAM Machine $\mathcal{M}_{\mathsf{NISC}}$

Public Input: $\mathsf{id}, f, \{\widehat{x}_{\mathsf{idx}}\}_{\mathsf{idx}\in I}, \widehat{f}_{\mathsf{id}}$
Private Input: x, r, τ
Output: Output 1 if and only if

- $(\widehat{x}_{\mathsf{id}}, \tau) = \mathsf{InpEnc}(\mathsf{id}, x; r)$.
- $\widehat{f}_{\mathsf{id}} = \mathsf{FuncEnc}(\mathsf{id}, f, \{\widehat{x}_{\mathsf{idx}}\}_{\mathsf{idx}\in I}, \tau; r)$.

Function g

Hardwired: $\lambda, \{\rho_{\mathsf{id}}\}_{\mathsf{id}\in I}, f$
Input: $\{\mathsf{sk}_{\mathsf{id}}\}_{\mathsf{id}\in I}$
Output:

- $\forall \mathsf{id} \in I$, $x_{\mathsf{id}} = \mathsf{r1Enc.Decode}(1^\lambda, \rho_{\mathsf{id}}, \mathsf{sk}_{\mathsf{id}})$.
- Output $f(\{x_{\mathsf{id}}\}_{\mathsf{id}\in I})$.

$\mathsf{Setup}(1^\lambda) \to \mathsf{crs}$**.** This probabilistic algorithm samples a reusable encrypted RAM delegation crs for each $\mathsf{id} \in [n]$, $\mathsf{del.crs}_{\mathsf{id}} \leftarrow \mathsf{r}\mathcal{E}\mathsf{Del.Setup}(1^\lambda, 1^{\ell_{\mathsf{pr}}})$ where ℓ_{pr} is the maximum length of the private input for $\mathcal{M}_{\mathsf{NISC}}$ across all n parties. Also, sample SMSC's crs, $\mathsf{smsc.crs} \leftarrow \mathsf{SMSC.Setup}(1^\lambda)$. Output $\mathsf{crs} = (\mathsf{smsc.crs}, (\mathsf{del.crs}_{\mathsf{id}})_{\mathsf{id}\in[n]})$.

$\mathsf{InpEnc}(\mathsf{crs}, \mathsf{id}, x; r) \to (\widehat{x}, \tau)$**.** This probabilistic algorithm parses crs, using $\mathsf{smsc.crs}$ and $\mathsf{del.crs}_{\mathsf{id}}$ and does the following.

1. Sample $r' \leftarrow \{0,1\}^\lambda$. Compute $(\rho, \mathsf{sk}) = \mathsf{r1Enc.Encode}(1^\lambda, x)$.
2. Sample input encoding for SMSC for sk, $(\mathsf{smsc.}\widehat{\mathsf{sk}}, \mathsf{smsc.}\tau) \leftarrow \mathsf{SMSC.InpEnc}(\mathsf{smsc.crs}, \mathsf{sk}; r')$.
3. Compute private encoding from $\mathsf{r}\mathcal{E}\mathsf{Del}$, $(\mathsf{del.eh}_{\mathsf{pr}}, \mathsf{del.}\pi_{\mathsf{enc}}, \mathsf{del.aux}) \leftarrow \mathsf{r}\mathcal{E}\mathsf{Del.PrivEnc}(\mathsf{del.crs}_{\mathsf{id}}, (\mathsf{sk}, r', \mathsf{smsc.}\tau))$.
4. Output $\widehat{x} = (\rho, \mathsf{smsc.}\widehat{\mathsf{sk}}, \mathsf{del.eh}_{\mathsf{pr}}, \mathsf{del.}\pi_{\mathsf{enc}})$, $\tau = \mathsf{del.aux}$.

$\mathsf{FuncEnc}(\mathsf{crs}, \mathsf{id}, f, \{\widehat{x}_{\mathsf{idx}}\}_{\mathsf{idx}\in I}, \tau_{\mathsf{id}}) \to \widehat{f}_{\mathsf{id}}/\perp$**.** This probabilistic algorithm parses the crs, uses $\mathsf{smsc.crs}$, $(\mathsf{del.crs}_{\mathsf{idx}})_{\mathsf{idx}}$, τ_{id} as $\mathsf{del.aux}_{\mathsf{id}}$, $\widehat{x}_{\mathsf{idx}}$ as $(\rho_{\mathsf{idx}}, \mathsf{smsc.}\widehat{\mathsf{sk}}_{\mathsf{idx}}, \mathsf{del.eh}_{\mathsf{pr},\mathsf{idx}}, \mathsf{del.}\pi_{\mathsf{enc},\mathsf{idx}})$ for $\mathsf{idx} \in I$, and does the following.

1. If for any $\mathsf{idx} \in I$, $\mathsf{r}\mathcal{E}\mathsf{Del.VerifyEnc}(\mathsf{del.crs}_{\mathsf{idx}}, \mathsf{del.eh}_{\mathsf{pr},\mathsf{idx}}, \mathsf{del.}\pi_{\mathsf{enc},\mathsf{idx}}) = 0$, abort and output $\perp$.
2. Otherwise, sample, using SMSC, function encoding for g defined in Figure 7.4. $\mathsf{smsc.}\widehat{g}_{\mathsf{id}} \leftarrow \mathsf{SMSC.FuncEnc}(\mathsf{smsc.crs}, \mathsf{id}, g, \{\mathsf{smsc.}\widehat{\mathsf{sk}}_{\mathsf{idx}}\}_{\mathsf{idx}}, \mathsf{smsc.}\tau_{\mathsf{id}}; r'_{\mathsf{id}})$ where $\mathsf{smsc.}\tau_{\mathsf{id}}, r'_{\mathsf{id}}$ are derived from $\mathsf{del.aux}_{\mathsf{id}}$.
3. Compute proof, $\mathsf{del.}\pi_{\mathsf{edel},\mathsf{id}} \leftarrow \mathsf{r}\mathcal{E}\mathsf{Del.Prove}(\mathsf{del.crs}_{\mathsf{id}}, (\mathsf{id}, g, \{\mathsf{smsc.}\widehat{\mathsf{sk}}_{\mathsf{idx}}\}_{\mathsf{idx}}, \mathsf{smsc.}\widehat{g}_{\mathsf{id}}), \mathsf{del.aux}_{\mathsf{id}})$.
4. Output $\widehat{f}_{\mathsf{id}} = (\mathsf{smsc.}\widehat{g}_{\mathsf{id}}, \mathsf{del.}\pi_{\mathsf{edel},\mathsf{id}})$.

$\mathsf{Eval}(\mathsf{crs}, f, \{\widehat{x}_{\mathsf{id}}\}_{\mathsf{id}\in I}, \{\widehat{f}_{\mathsf{id}}\}_{\mathsf{id}\in I}) \to y/\perp$**.** This algorithm parses crs, uses $\mathsf{smsc.crs}$, $(\mathsf{del.crs}_{\mathsf{id}})_{\mathsf{id}}$, for each $\mathsf{id} \in I$, $\widehat{x}_{\mathsf{id}}$ as $(\rho_{\mathsf{id}}, \mathsf{smsc.}\widehat{\mathsf{sk}}_{\mathsf{id}}, \mathsf{del.eh}_{\mathsf{pr},\mathsf{id}}, \mathsf{del.}\pi_{\mathsf{enc},\mathsf{id}})$, $\widehat{f}_{\mathsf{id}}$ as $(\mathsf{smsc.}\widehat{g}_{\mathsf{id}}, \mathsf{del.}\pi_{\mathsf{edel},\mathsf{id}})$ and does the following.

1. Construct g and compute for $\mathsf{id} \in I$, $\mathsf{h}_{\mathsf{pb},\mathsf{id}} = \mathsf{HT.Hash}(\mathsf{ht.hk}, \mathsf{id}, g, \{\mathsf{smsc.}\widehat{\mathsf{sk}}_{\mathsf{idx}}\}_{\mathsf{idx}\in I}, \mathsf{smsc.}\widehat{g}_{\mathsf{id}})$.
2. If for any $\mathsf{id} \in I$, $\mathsf{r}\mathcal{E}\mathsf{Del.VerifyEnc}(\mathsf{del.crs}_{\mathsf{id}}, \mathsf{del.eh}_{\mathsf{pr},\mathsf{id}}, \mathsf{del.}\pi_{\mathsf{enc},\mathsf{id}}) = 0$ or $\mathsf{r}\mathcal{E}\mathsf{Del.Verify}(\mathsf{del.crs}_{\mathsf{id}}, \mathsf{h}_{\mathsf{pb},\mathsf{id}}, \mathsf{del.eh}_{\mathsf{pr},\mathsf{id}}, 1, \mathsf{del.}\pi_{\mathsf{edel},\mathsf{id}}) = 0$, abort, output $\perp$.
3. Output $\mathsf{SMSC.Eval}(\mathsf{smsc.crs}, g, \{\mathsf{smsc.}\widehat{\mathsf{sk}}_{\mathsf{id}}\}_{\mathsf{id}}, \{\mathsf{smsc.}\widehat{g}_{\mathsf{id}}\}_{\mathsf{id}})$.

Theorem 7.5. If SMSC is a semi-maliciously secure mrNISC scheme (Definition 7.1), $\mathsf{r1Enc}$ is a secure rate-1 message encoding scheme (Definition 3.14), and $\mathsf{r\mathcal{E}Del}$ is a reusable encrypted RAM delegation scheme w.r.to HT (Definition 4.2) for machine $\mathcal{M}_{\mathsf{NISC}}$, then Construction 7.4 is a complexity-preserving maliciously secure mrNISC scheme.

Proof. We show that Construction 7.4 is correct and efficient as follows:

Correctness. We have that $\mathsf{crs} = (\mathsf{smsc.crs}, (\mathsf{del.crs_{idx}})_{\mathsf{idx}\in[n]})$, for each $\mathsf{id} \in I$,

$$\widehat{x}_{\mathsf{id}} = (\mathsf{r1Enc.Encode}(1^\lambda, x_{\mathsf{id}}), \mathsf{smsc.}\widehat{\mathsf{sk}}_{\mathsf{id}}, \mathsf{del.eh_{pr,id}}, \mathsf{del.}\pi_{\mathsf{enc,id}}, \mathsf{smsc.}\tau_{\mathsf{id}})$$
$$\widehat{f}_{\mathsf{id}} = (\mathsf{SMSC.FuncEnc}(\mathsf{smsc.crs}, \mathsf{id}, g, \{\mathsf{smsc.}\widehat{\mathsf{sk}}_{\mathsf{idx}}\}_{\mathsf{idx}\in I}, \mathsf{smsc.}\tau_{\mathsf{id}}; r'_{\mathsf{id}}),$$
$$\mathsf{r\mathcal{E}Del.Prove}(\mathsf{del.crs_{id}}, x_{\mathsf{pb,id}}, \mathsf{del.aux_{id}}))$$

where $(\mathsf{del.eh_{pr,id}}, \mathsf{del.}\pi_{\mathsf{enc,id}}, \mathsf{del.aux_{id}}) \leftarrow \mathsf{r\mathcal{E}Del.PrivEnc}(\mathsf{del.crs_{id}}, (\mathsf{sk_{id}}, r'_{\mathsf{id}}, x))$ and $(\mathsf{smsc.}\widehat{\mathsf{sk}}_{\mathsf{id}}, \mathsf{smsc.}\tau_{\mathsf{id}}) \leftarrow \mathsf{SMSC.InpEnc}(\mathsf{smsc.crs}, \mathsf{sk_{id}}; r'_{\mathsf{id}})$. By completeness of $\mathsf{r\mathcal{E}Del}$,

$$\mathsf{r\mathcal{E}Del.VerifyEnc}(\mathsf{del.crs_{id}}, \mathsf{del.eh_{pr,id}}, \mathsf{del.}\pi_{\mathsf{enc,id}}) = 1 \text{ and}$$
$$\mathsf{r\mathcal{E}Del.Verify}(\mathsf{del.crs_{id}}, \mathsf{h_{pb,id}}, \mathsf{del.eh_{pr,id}}, 1,$$
$$\mathsf{r\mathcal{E}Del.Prove}(\mathsf{del.crs_{id}}, \mathsf{id}, g, \{\mathsf{smsc.}\widehat{\mathsf{sk}}_{\mathsf{idx}}\}_{\mathsf{idx}}, \mathsf{smsc.}\widehat{g}_{\mathsf{id}}, \mathsf{del.aux_{id}})) = 1$$

By correctness of SMSC, $y = g(\{\mathsf{sk_{id}}\}_{\mathsf{id}\in I}) = f(\{x_{\mathsf{id}}\}_{\mathsf{id}\in I})$.

Efficiency. We prove that our transformation is rate-1 in the input x_{id} for input encoding, and complexity-preserving for function encoding.

Input Encoding. As $\mathsf{sk_{id}}$ and r'_{id} are λ-bit strings, length of $\mathsf{smsc.}\widehat{\mathsf{sk}}_{\mathsf{id}}$ and $\mathsf{smsc.}\tau_{\mathsf{id}}$ is polynomial in λ. Similarly, the length of $\mathsf{del.eh_{pr,id}}$ and $\mathsf{del.}\pi_{\mathsf{enc,id}}$ is polynomial in λ. However, ρ_{id} is the same length as x_{id}. Thus $\widehat{x}_{\mathsf{id}}$ is a rate-1 string and τ_{id} is independent of the length of x_{id}.

Function Encoding. Note that the size of the function g, $|g| = |f| + \mathsf{poly}(\lambda)$ where the polynomial $\mathsf{poly}(\lambda)$ depends on $\mathsf{r1Enc}$. By efficiency of $\mathsf{r\mathcal{E}Del}$, we have that size of $\mathsf{del.}\pi_{\mathsf{edel,id}}$ is $\mathsf{poly}(\lambda) + \mathsf{poly}(\log|g|, \lambda)$. Hence, the size of $\widehat{f}_{\mathsf{id}}$ is the size of function encoding of SMSC for g and some additional polynomial in λ. Thus, the function encoding for f in Construction 7.4 is complexity preserving.

□

7.3 Security Analysis of Construction 7.4

Here, we show that Construction 7.4 is secure against malicious adversaries in the static corruptions, adaptive query setting.

Proof of static corruptions, malicious, adaptive query security. We show that if $\mathsf{SMSC}, \mathsf{r1Enc}$ are secure, and $\mathsf{r\mathcal{E}Del}$ is input-hiding, then Construction 7.4 is a maliciously secure mrNISC scheme. We define the following hybrids:

$\mathsf{hyb}_0^{\mathcal{A}}$ This is the honest experiment $\mathsf{Expt}_0^{\mathcal{C},\mathcal{A}}$ with malicious adversary $\mathcal{A}$.

$\mathsf{hyb}^{\mathcal{A}}_{1,j}$ This is the same as $\mathsf{hyb}^{\mathcal{A}}_{0}$ except that for $j \in [n+1]$, we simulate $\mathsf{r}\mathcal{E}\mathsf{Del}$ instantiations for the first $j-1$ honest parties.

$\mathsf{hyb}^{\mathcal{A}}_{2,j}$ This is the same as $\mathsf{hyb}^{\mathcal{A}}_{1,n+1}$ except that for $j \in [n+1]$, we use the extractor $\mathsf{r}\mathcal{E}\mathsf{Del}$ for the first $j-1$ corrupt parties to extract the corrupt party input and randomness. If these values are not consistent with SMSC function encodings provided by the malicious adversary, abort and output $\bot$.

$\mathsf{hyb}^{\mathcal{A}}_{3}$ This is the same as $\mathsf{hyb}^{\mathcal{A}}_{2,n+1}$ except that we simulate all encodings of SMSC for honest parties.

$\mathsf{hyb}^{\mathcal{A}}_{4,j}$ This is same as $\mathsf{hyb}^{\mathcal{A}}_{3}$ except that for $j \in [n+1]$, we sample random ρ_{id} instead of using $\mathsf{r1Enc.Encode}$ for the first $j-1$ honest parties. $\mathsf{hyb}^{\mathcal{A}}_{4,n+1}$ is the description of simulator $\mathcal{S}$.

Let $\mathsf{hyb}^{\mathcal{A}}_{i(,j)}(1^\lambda)$ denote the output of the experiment $\mathsf{hyb}^{\mathcal{A}}_{i(,j)}$.

Claim 7.6. For every $\lambda \in \mathbb{N}$, any PPT adversary $\mathcal{A}$,

$$\Pr\left[1 \leftarrow \mathsf{hyb}^{\mathcal{A}}_{0}(1^\lambda)\right] = \Pr\left[1 \leftarrow \mathsf{hyb}^{\mathcal{A}}_{1,1}(1^\lambda)\right]$$

Proof. The proof of this claim is immediate from the distributions of $\mathsf{hyb}^{\mathcal{A}}_{0}$ and $\mathsf{hyb}^{\mathcal{A}}_{1,1}$. □

Claim 7.7. If $\mathsf{r}\mathcal{E}\mathsf{Del}$ is input-hiding, then for any PPT adversary $\mathcal{A}$, for any $j \in [n]$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for every $\lambda \in \mathbb{N}$:

$$\left|\Pr\left[1 \leftarrow \mathsf{hyb}^{\mathcal{A}}_{1,j}(1^\lambda)\right] - \Pr\left[1 \leftarrow \mathsf{hyb}^{\mathcal{A}}_{1,j+1}(1^\lambda)\right]\right| \leq \mathsf{negl}(\lambda)$$

Proof. Note that if $j \in \widetilde{H}$ where $\widetilde{H} = [n] \setminus H$, $\mathsf{hyb}^{\mathcal{A}}_{1,j}$ and $\mathsf{hyb}^{\mathcal{A}}_{1,j+1}$ are identical. Hence, w.l.o.g assume $j \in H$. Assuming that there exists an adversary $\mathcal{A}$ that can distinguish between $\mathsf{hyb}^{\mathcal{A}}_{1,j}$ and $\mathsf{hyb}^{\mathcal{A}}_{1,j+1}$ such that $|\Pr\left[1 \leftarrow \mathsf{hyb}^{\mathcal{A}}_{1,j}(1^\lambda)\right] - \Pr\left[1 \leftarrow \mathsf{hyb}^{\mathcal{A}}_{1,j+1}(1^\lambda)\right]| = \epsilon(\lambda)$ for some non-negligible function $\epsilon(\cdot)$, we construct an adversary $\mathcal{B}_{\mathsf{del}}$ that breaks the input-hiding property of $\mathsf{r}\mathcal{E}\mathsf{Del}$ as follows:

1. $\mathcal{B}_{\mathsf{del}}$ receives crs from the challenger $\mathcal{C}$. $\mathcal{A}$ submits the number of parties n and the set of honest parties H. If $\mathsf{id} \in H, \mathsf{id} < j$, sample $\mathsf{del.crs}_{\mathsf{id}} \leftarrow \mathsf{r}\mathcal{E}\mathsf{Del}.\mathcal{S}(1^\lambda, 1^{\ell_{\mathsf{pr}}})$. Otherwise if $\mathsf{id} = j$, set $\mathsf{del.crs}_{\mathsf{id}} := \mathsf{crs}$. Otherwise, if $\mathsf{id} > j$, sample $\mathsf{del.crs}_{\mathsf{id}} \leftarrow \mathsf{r}\mathcal{E}\mathsf{Del.Setup}(1^\lambda, 1^{\ell_{\mathsf{pr}}})$ in accordance with $\mathcal{M}_{\mathsf{NISC}}$. Sample $\mathsf{smsc.crs} \leftarrow \mathsf{SMSC.Setup}(1^\lambda)$ and send $(\mathsf{smsc.crs}, (\mathsf{del.crs}_{\mathsf{id}})_{\mathsf{id}})$ to $\mathcal{A}$.

2. When $\mathcal{A}$ queries for input encoding of $(\mathsf{id}, x_{\mathsf{id}})$ for $\mathsf{id} \in H$, do the following.

   - Sample $r'_{\mathsf{id}} \leftarrow \{0,1\}^\lambda$. Compute $(\rho_{\mathsf{id}}, \mathsf{sk}_{\mathsf{id}}) \leftarrow \mathsf{r1Enc.Encode}(1^\lambda, x_{\mathsf{id}})$.
   - Compute $(\mathsf{smsc.}\widehat{\mathsf{sk}}_{\mathsf{id}}, \mathsf{smsc.}\tau_{\mathsf{id}}) \leftarrow \mathsf{SMSC.InpEnc}(\mathsf{smsc.crs}, \mathsf{sk}_{\mathsf{id}}; r'_{\mathsf{id}})$.
   - If $\mathsf{id} < j$, $(\mathsf{del.eh}_{\mathsf{pr},\mathsf{id}}, \mathsf{del.}\pi_{\mathsf{enc},\mathsf{id}}) \leftarrow \mathsf{r}\mathcal{E}\mathsf{Del}.\mathcal{S}(\mathsf{del.crs}_{\mathsf{id}}, 1^{|\mathsf{sk}_{\mathsf{id}}|+\lambda+|\mathsf{smsc.}\tau_{\mathsf{id}}|})$. Otherwise, if $\mathsf{id} > j$, $(\mathsf{del.eh}_{\mathsf{pr},\mathsf{id}}, \mathsf{del.}\pi_{\mathsf{enc},\mathsf{id}}, \mathsf{del.aux}_{\mathsf{id}}) \leftarrow \mathsf{r}\mathcal{E}\mathsf{Del.PrivEnc}(\mathsf{del.crs}_{\mathsf{id}}, (\mathsf{sk}_{\mathsf{id}}, r_{\mathsf{id}}, \mathsf{smsc.}\tau_{\mathsf{id}}))$. Otherwise, if $\mathsf{id} = j$, send $(\mathsf{sk}_{\mathsf{id}}, r_{\mathsf{id}}, \mathsf{smsc.}\tau_{\mathsf{id}})$ to $\mathcal{C}$ and receive $(\mathsf{del.eh}_{\mathsf{pr},\mathsf{id}}, \mathsf{del.}\pi_{\mathsf{enc},\mathsf{id}})$.

   Send $\widehat{x}_{\mathsf{id}} = (\rho_{\mathsf{id}}, \mathsf{smsc.}\widehat{\mathsf{sk}}_{\mathsf{id}}, \mathsf{del.eh}_{\mathsf{pr},\mathsf{id}}, \mathsf{del.}\pi_{\mathsf{enc},\mathsf{id}})$ to $\mathcal{A}$.

3. $\mathcal{A}$ submits input encoding for $\mathsf{id} \in \widetilde{H}$, parse $\widehat{x}_{\mathsf{id}}$ as $(\rho_{\mathsf{id}}, \mathsf{smsc.}\widehat{\mathsf{sk}}_{\mathsf{id}}, \mathsf{del.eh}_{\mathsf{pr},\mathsf{id}}, \mathsf{del.}\pi_{\mathsf{enc},\mathsf{id}})$. If $\mathsf{r}\mathcal{E}\mathsf{Del.\ VerifyEnc}(\mathsf{del.crs}_{\mathsf{id}}, \mathsf{del.eh}_{\mathsf{pr},\mathsf{id}}, \mathsf{del.}\pi_{\mathsf{enc},\mathsf{id}}) = 0$, abort and output $\bot$.

4. When $\mathcal{A}$ queries for function encoding of (id, f, I) for $\mathsf{id} \in I \cap H$, such that all the input encodings for parties $\mathsf{idx} \in I$ are known, do the following:

   - If for any $\mathsf{idx} \in I \cap \widetilde{H}$, if $\mathsf{r\mathcal{E}Del.VerifyEnc}(\mathsf{del.crs}_{\mathsf{idx}}, \mathsf{del.eh}_{\mathsf{pr},\mathsf{idx}}, \mathsf{del.}\pi_{\mathsf{enc},\mathsf{idx}}) = 0$, abort and output $\perp$.
   - Sample $\mathsf{smsc.}\widehat{g}_{\mathsf{id}} \leftarrow \mathsf{SMSC.FuncEnc}(\mathsf{smsc.crs}, \mathsf{id}, g, \{\mathsf{smsc.}\widehat{\mathsf{sk}}_{\mathsf{idx}}\}_{\mathsf{idx}}, \mathsf{smsc.}\tau_{\mathsf{id}}; r'_{\mathsf{id}})$.
   - If $\mathsf{id} < j$, sample $\mathsf{del.}\pi_{\mathsf{edel},\mathsf{id}} \leftarrow \mathsf{r\mathcal{E}Del.\mathcal{S}}(\mathsf{del.crs}_{\mathsf{id}}, (\mathsf{id}, g, \{\mathsf{smsc.}\widehat{\mathsf{sk}}_{\mathsf{idx}}\}_{\mathsf{idx}}, \mathsf{smsc.}\widehat{g}_{\mathsf{id}}), 1)$. Otherwise, if $\mathsf{id} > j$, $\mathsf{del.}\pi_{\mathsf{edel},\mathsf{id}} \leftarrow \mathsf{r\mathcal{E}Del.Prove}(\mathsf{del.crs}_{\mathsf{id}}, (\mathsf{id}, g, \{\mathsf{smsc.}\widehat{\mathsf{sk}}_{\mathsf{idx}}\}_{\mathsf{idx}}, \mathsf{smsc.}\widehat{g}_{\mathsf{id}}), \mathsf{del.aux}_{\mathsf{id}})$. Otherwise, if $\mathsf{id} = j$, send $((\mathsf{id}, g, \{\mathsf{smsc.}\widehat{\mathsf{sk}}_{\mathsf{idx}}\}_{\mathsf{idx}}, \mathsf{smsc.}\widehat{g}_{\mathsf{id}}), 1)$ to $\mathcal{C}$ and receive $\mathsf{del.}\pi_{\mathsf{edel},\mathsf{id}}$.

   Send $(\mathsf{smsc.}\widehat{g}_{\mathsf{id}}, \mathsf{del.}\pi_{\mathsf{edel},\mathsf{id}})$ to $\mathcal{A}$. Here, we use g as described in Figure 7.4.

5. $\mathcal{A}$ sends the function encoding for $\mathsf{id} \in I \cap \widetilde{H}$, such that all the input encodings for $\mathsf{idx} \in I$ are known, parse $\widehat{f}_{\mathsf{id}}$ as $(\mathsf{smsc.}\widehat{g}_{\mathsf{id}}, \mathsf{del.}\pi_{\mathsf{edel},\mathsf{id}})$. If $\mathsf{r\mathcal{E}Del.Verify}(\mathsf{del.crs}_{\mathsf{id}}, \mathsf{h}_{\mathsf{pb},\mathsf{id}}, \mathsf{del.eh}_{\mathsf{pr},\mathsf{id}}, 1, \mathsf{del.}\pi_{\mathsf{edel},\mathsf{id}}) = 0$, abort and output $\perp$ where $\mathsf{h}_{\mathsf{pb},\mathsf{id}} = \mathsf{HT.Hash}(\mathsf{ht.hk}, \mathsf{id}, g, \{\mathsf{smsc.}\widehat{x}_{\mathsf{idx}}\}_{\mathsf{idx}}, \mathsf{smsc.}\widehat{g}_{\mathsf{id}})$.

6. Output whatever $\mathcal{A}$ outputs.

If $\mathcal{C}$ uses $\mathsf{r\mathcal{E}Del.Setup}$, $\mathsf{r\mathcal{E}Del.PrivEnc}$, and $\mathsf{r\mathcal{E}Del.Prove}$ (resp. $\mathsf{r\mathcal{E}Del.\mathcal{S}}$) in the experiment then $\mathcal{B}_{\mathsf{del}}$ simulates $\mathsf{hyb}^{\mathcal{A}}_{1,j}$ (resp. $\mathsf{hyb}^{\mathcal{A}}_{1,j+1}$). Thus $\mathcal{B}_{\mathsf{del}}$ has advantage $\epsilon(\lambda)$ in breaking input hiding property of $\mathsf{r\mathcal{E}Del}$. □

Claim 7.8. For every $\lambda \in \mathbb{N}$, any PPT adversary $\mathcal{A}$,

$$\Pr\left[1 \leftarrow \mathsf{hyb}^{\mathcal{A}}_{1,n+1}(1^\lambda)\right] = \Pr\left[1 \leftarrow \mathsf{hyb}^{\mathcal{A}}_{2,1}(1^\lambda)\right]$$

Proof. The proof of this claim is immediate from the distributions of $\mathsf{hyb}^{\mathcal{A}}_{1,n+1}$ and $\mathsf{hyb}^{\mathcal{A}}_{2,1}$. □

Claim 7.9. If $\mathsf{r\mathcal{E}Del}$ satisfies straight-line extraction, then for any PPT adversary $\mathcal{A}$, any $j \in [n+1]$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for every $\lambda \in \mathbb{N}$, $\Pr\left[\perp \leftarrow \mathsf{hyb}^{\mathcal{A}}_{2,j}(1^\lambda)\right] \leq \mathsf{negl}(\lambda)$.

Proof. Note that if $j \in H$, $\mathsf{hyb}^{\mathcal{A}}_{2,j}$ will not abort as there is no extractor involved in the hybrid. Hence, w.l.o.g assume $j \in \widetilde{H}$. Assuming that there exists an adversary $\mathcal{A}$ such that $\Pr\left[\perp \leftarrow \mathsf{hyb}^{\mathcal{A}}_{2,j}(1^\lambda)\right] = \epsilon(\lambda)$ for some non-negligible function $\epsilon(\cdot)$, we construct an adversary $\mathcal{B}_{\mathsf{del}}$ that breaks the straight-line extraction property of $\mathsf{r\mathcal{E}Del}$ as follows:

1. $\mathcal{B}_{\mathsf{del}}$ receives crs from the challenger $\mathcal{C}$. $\mathcal{A}$ submits the number of parties n and the set of honest parties H. For each $\mathsf{id} \in [n]$, if $\mathsf{id} \in H$, sample $\mathsf{del.crs}_{\mathsf{id}} \leftarrow \mathsf{r\mathcal{E}Del.\mathcal{S}}(1^\lambda, 1^{\ell_{\mathsf{pr}}})$. Otherwise if $\mathsf{id} \in \widetilde{H}$ and $\mathsf{id} < j$, sample $(\mathsf{del.crs}_{\mathsf{id}}, \mathsf{del.td}_{\mathsf{id}}) \leftarrow \mathsf{r\mathcal{E}Del.\mathcal{E}}(1^\lambda, 1^{\ell_{\mathsf{pr}}})$ in accordance with $\mathcal{M}_{\mathsf{NISC}}$. Otherwise if $\mathsf{id} \in \widetilde{H}, \mathsf{id} > j$, sample $\mathsf{del.crs}_{\mathsf{id}} \leftarrow \mathsf{r\mathcal{E}Del.Setup}(1^\lambda, 1^{\ell_{\mathsf{pr}}})$. If $\mathsf{id} = j$, set $\mathsf{del.crs}_{\mathsf{id}} := \mathsf{crs}$. Sample $\mathsf{smsc.crs} \leftarrow \mathsf{SMSC.Setup}(1^\lambda)$ and send $(\mathsf{smsc.crs}, (\mathsf{del.crs}_{\mathsf{id}})_{\mathsf{id}})$ to $\mathcal{A}$.

2. When $\mathcal{A}$ queries for input encoding of $(\mathsf{id}, x_{\mathsf{id}})$ for $\mathsf{id} \in H$, do the following.

   - Sample $r'_{\mathsf{id}} \leftarrow \{0,1\}^\lambda$. Compute $(\rho_{\mathsf{id}}, \mathsf{sk}_{\mathsf{id}}) \leftarrow \mathsf{r1Enc.Encode}(1^\lambda, x_{\mathsf{id}})$.
   - Compute $(\mathsf{smsc.}\widehat{\mathsf{sk}}_{\mathsf{id}}, \mathsf{smsc.}\tau_{\mathsf{id}}) \leftarrow \mathsf{SMSC.InpEnc}(\mathsf{smsc.crs}, \mathsf{sk}_{\mathsf{id}}; r'_{\mathsf{id}})$.
   - Sample $(\mathsf{del.eh}_{\mathsf{pr},\mathsf{id}}, \mathsf{del.}\pi_{\mathsf{enc},\mathsf{id}}) \leftarrow \mathsf{r\mathcal{E}Del.\mathcal{S}}(\mathsf{del.crs}_{\mathsf{id}})$.

Send $\widehat{x}_{\mathsf{id}} = (\rho_{\mathsf{id}}, \mathsf{smsc}.\widehat{\mathsf{sk}}_{\mathsf{id}}, \mathsf{del}.\mathsf{eh}_{\mathsf{pr},\mathsf{id}}, \mathsf{del}.\pi_{\mathsf{enc},\mathsf{id}})$ to $\mathcal{A}$.

3. $\mathcal{A}$ submits input encoding for $\mathsf{id} \in \widetilde{H}$, parse $\widehat{x}_{\mathsf{id}}$ as $(\rho_{\mathsf{id}}, \mathsf{smsc}.\widehat{\mathsf{sk}}_{\mathsf{id}}, \mathsf{del}.\mathsf{eh}_{\mathsf{pr},\mathsf{id}}, \mathsf{del}.\pi_{\mathsf{enc},\mathsf{id}})$. If $\mathsf{r}\mathcal{E}\mathsf{Del}.\mathsf{VerifyEnc}(\mathsf{del}.\mathsf{crs}_{\mathsf{id}}, \mathsf{del}.\mathsf{eh}_{\mathsf{pr},\mathsf{id}}, \mathsf{del}.\pi_{\mathsf{enc},\mathsf{id}}) = 0$, abort and output $\perp$. Otherwise, if $\mathsf{id} < j$, extract $(\mathsf{sk}_{\mathsf{id}}, r'_{\mathsf{id}}, \mathsf{smsc}.\tau_{\mathsf{id}}) \leftarrow \mathsf{r}\mathcal{E}\mathsf{Del}.\mathcal{E}(\mathsf{del}.\mathsf{td}_{\mathsf{id}}, \mathsf{del}.\pi_{\mathsf{enc},\mathsf{id}})$. If $\mathsf{id} = j$, send $(\mathsf{del}.\mathsf{eh}_{\mathsf{pr},\mathsf{id}}, \mathsf{del}.\pi_{\mathsf{enc},\mathsf{id}})$ to $\mathcal{C}$.

4. When $\mathcal{A}$ queries for function encoding of (id, f, I) for $\mathsf{id} \in I \cap H$ such that all the input encodings for parties $\mathsf{idx} \in I$ are known, do the following.

   - Sample $\mathsf{smsc}.\widehat{g}_{\mathsf{id}} \leftarrow \mathsf{SMSC.FuncEnc}(\mathsf{smsc.crs}, \mathsf{id}, g, \{\mathsf{smsc}.\widehat{\mathsf{sk}}_{\mathsf{idx}}\}_{\mathsf{idx}}, \mathsf{smsc}.\tau_{\mathsf{id}}; r'_{\mathsf{id}})$.
   - Sample $\mathsf{del}.\pi_{\mathsf{edel},\mathsf{id}} \leftarrow \mathsf{r}\mathcal{E}\mathsf{Del}.\mathcal{S}(\mathsf{del}.\mathsf{crs}_{\mathsf{id}}, (\mathsf{id}, g, \{\mathsf{smsc}.\widehat{\mathsf{sk}}_{\mathsf{idx}}\}_{\mathsf{idx}}, \mathsf{smsc}.\widehat{g}_{\mathsf{id}}), 1)$.

   Send $(\mathsf{smsc}.\widehat{g}_{\mathsf{id}}, \mathsf{del}.\pi_{\mathsf{edel},\mathsf{id}})$ to $\mathcal{A}$. Here, we use g as described in Figure 7.4.

5. $\mathcal{A}$ sends the function encoding for $\mathsf{id} \in I \cap \widetilde{H}$, such that all the input encodings for $\mathsf{idx} \in I$ are known, parse $\widehat{f}_{\mathsf{id}}$ as $(\mathsf{smsc}.\widehat{g}_{\mathsf{id}}, \mathsf{del}.\pi_{\mathsf{edel},\mathsf{id}})$, and do the following:

   - Set $x_{\mathsf{pb},\mathsf{id}} := (\mathsf{id}, g, \{\mathsf{smsc}.\widehat{\mathsf{sk}}_{\mathsf{idx}}\}_{\mathsf{idx}}, \mathsf{smsc}.\widehat{g}_{\mathsf{id}})$ and compute $\mathsf{h}_{\mathsf{pb},\mathsf{id}} = \mathsf{HT.Hash}(\mathsf{ht.hk}, x_{\mathsf{pb},\mathsf{id}})$.
   - For any $\mathsf{id} \neq j$, $\mathsf{r}\mathcal{E}\mathsf{Del}.\mathsf{Verify}(\mathsf{del}.\mathsf{crs}_{\mathsf{id}}, \mathsf{h}_{\mathsf{pb},\mathsf{id}}, \mathsf{del}.\mathsf{eh}_{\mathsf{pr},\mathsf{id}}, 1, \mathsf{del}.\pi_{\mathsf{edel},\mathsf{id}}) = 0$ abort and output $\perp$.
   - If $\mathsf{id} < j$, $\mathcal{M}_{\mathsf{NISC}}(x_{\mathsf{pb},\mathsf{id}}, (\mathsf{sk}_{\mathsf{id}}, r'_{\mathsf{id}}, \mathsf{smsc}.\tau_{\mathsf{id}})) = 0$, abort and output $\perp$.
   - Otherwise, if $\mathsf{id} = j$, send $(x_{\mathsf{pb},\mathsf{id}}, \mathsf{del}.\pi_{\mathsf{edel},\mathsf{id}})$ to $\mathcal{C}$.

6. Output whatever $\mathcal{A}$ outputs.

Note that $\mathcal{B}_{\mathsf{del}}$ is a valid adversary against the straight-line extractor of $\mathsf{r}\mathcal{E}\mathsf{Del}$ with advantage $\epsilon(\lambda)$ whenever $\mathcal{A}$ can make trigger an abort in $\mathsf{hyb}^{\mathcal{A}}_{2,j}(1^\lambda)$. □

Claim 7.10. If SMSC is a semi-maliciously secure mrNISC scheme, then for any PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for every $\lambda \in \mathbb{N}$:

$$\left|\Pr\left[1 \leftarrow \mathsf{hyb}^{\mathcal{A}}_{2,n+1}(1^\lambda)\right] - \Pr\left[1 \leftarrow \mathsf{hyb}^{\mathcal{A}}_{3}(1^\lambda)\right]\right| \leq \mathsf{negl}(\lambda)$$

Proof. Assuming that there exists an adversary $\mathcal{A}$ that can distinguish between $\mathsf{hyb}^{\mathcal{A}}_{2,n+1}$ and $\mathsf{hyb}^{\mathcal{A}}_{3}$ such that $|\Pr\left[1 \leftarrow \mathsf{hyb}^{\mathcal{A}}_{2,n+1}(1^\lambda)\right] - \Pr\left[1 \leftarrow \mathsf{hyb}^{\mathcal{A}}_{3}(1^\lambda)\right]| = \epsilon(\lambda)$ for some non-negligible function $\epsilon(\cdot)$, we construct an adversary $\mathcal{B}_{\mathsf{SMSC}}$ that breaks the security of SMSC as follows:

1. $\mathcal{A}$ submits the number of parties n and the set of honest parties H. Send n, H to $\mathcal{C}$ and receives $\mathsf{smsc.crs}$. For $\mathsf{id} \in [n]$, if $\mathsf{id} \in H$, sample $\mathsf{del}.\mathsf{crs}_{\mathsf{id}} \leftarrow \mathsf{r}\mathcal{E}\mathsf{Del}.\mathcal{S}(1^\lambda, 1^{\ell_{\mathsf{pr}}})$. Otherwise, sample $(\mathsf{del}.\mathsf{crs}_{\mathsf{id}}, \mathsf{del}.\mathsf{td}_{\mathsf{id}}) \leftarrow \mathsf{r}\mathcal{E}\mathsf{Del}.\mathcal{E}(1^\lambda, 1^{\ell_{\mathsf{pr}}})$ in accordance with $\mathcal{M}_{\mathsf{NISC}}$. Send $(\mathsf{smsc.crs}, (\mathsf{del}.\mathsf{crs}_{\mathsf{id}})_{\mathsf{id}})$ to $\mathcal{A}$.

2. When $\mathcal{A}$ queries for input encoding of $(\mathsf{id}, x_{\mathsf{id}})$ for $\mathsf{id} \in H$, do the following.

   - Compute $(\rho_{\mathsf{id}}, \mathsf{sk}_{\mathsf{id}}) \leftarrow \mathsf{r1Enc.Encode}(1^\lambda, x_{\mathsf{id}})$.
   - Send $(\mathsf{id}, \mathsf{sk}_{\mathsf{id}})$ to $\mathcal{C}$ and receive $\mathsf{smsc}.\widehat{\mathsf{sk}}_{\mathsf{id}}$.
   - Sample $(\mathsf{del}.\mathsf{eh}_{\mathsf{pr},\mathsf{id}}, \mathsf{del}.\pi_{\mathsf{enc},\mathsf{id}}) \leftarrow \mathsf{r}\mathcal{E}\mathsf{Del}.\mathcal{S}(\mathsf{del}.\mathsf{crs}_{\mathsf{id}})$.

Send $\widehat{x}_{\mathsf{id}} = (\rho_{\mathsf{id}}, \mathsf{smsc}.\widehat{\mathsf{sk}}_{\mathsf{id}}, \mathsf{del}.\mathsf{eh}_{\mathsf{pr},\mathsf{id}}, \mathsf{del}.\pi_{\mathsf{enc},\mathsf{id}})$ to $\mathcal{A}$.

3. $\mathcal{A}$ submits input encoding for $\mathsf{id} \in \widetilde{H}$, parse $\widehat{x}_{\mathsf{id}}$ as $(\rho_{\mathsf{id}}, \mathsf{smsc}.\widehat{\mathsf{sk}}_{\mathsf{id}}, \mathsf{del}.\mathsf{eh}_{\mathsf{pr},\mathsf{id}}, \mathsf{del}.\pi_{\mathsf{enc},\mathsf{id}})$. If $\mathsf{r}\mathcal{E}\mathsf{Del}.\mathsf{VerifyEnc}(\mathsf{del}.\mathsf{crs}_{\mathsf{id}}, \mathsf{del}.\mathsf{eh}_{\mathsf{pr},\mathsf{id}}, \mathsf{del}.\pi_{\mathsf{enc},\mathsf{id}}) = 0$, abort and output $\perp$. Otherwise, extract $(\mathsf{sk}_{\mathsf{id}}, r'_{\mathsf{id}}, \mathsf{smsc}.\tau_{\mathsf{id}}) \leftarrow \mathsf{r}\mathcal{E}\mathsf{Del}.\mathcal{E}(\mathsf{del}.\mathsf{td}_{\mathsf{id}}, \mathsf{del}.\mathsf{eh}_{\mathsf{pr},\mathsf{id}}, \mathsf{del}.\pi_{\mathsf{enc},\mathsf{id}})$. Extract the input for id-th party, $x_{\mathsf{id}} = \mathsf{r1Enc}.\mathsf{Decode}(1^\lambda, \rho_{\mathsf{id}}, \mathsf{sk}_{\mathsf{id}})$. Send $(\mathsf{id}, \mathsf{sk}_{\mathsf{id}}, r'_{\mathsf{id}})$ to $\mathcal{C}$.

4. When $\mathcal{A}$ queries for function encoding of (id, f, I) for honest party $\mathsf{id} \in I$ such that all the input encodings for parties $\mathsf{idx} \in I$ are known, do the following.

   - Send (id, g, I) to $\mathcal{C}$ to receive $\mathsf{smsc}.\widehat{g}_{\mathsf{id}}$.
   - Sample $\mathsf{del}.\pi_{\mathsf{edel},\mathsf{id}} \leftarrow \mathsf{r}\mathcal{E}\mathsf{Del}.\mathcal{S}(\mathsf{del}.\mathsf{crs}_{\mathsf{id}}, (\mathsf{id}, g, \{\mathsf{smsc}.\widehat{\mathsf{sk}}_{\mathsf{idx}}\}_{\mathsf{idx}}, \mathsf{smsc}.\widehat{g}_{\mathsf{id}}), 1)$.

   Send $(\mathsf{smsc}.\widehat{g}_{\mathsf{id}}, \mathsf{del}.\pi_{\mathsf{edel},\mathsf{id}})$ to $\mathcal{A}$. Here, we use g as described in Figure 7.4.

5. $\mathcal{A}$ sends the function encoding for $\mathsf{id} \in I \cap \widetilde{H}$, such that all the input encodings for $\mathsf{idx} \in I$ are known, parse $\widehat{f}_{\mathsf{id}}$ as $(\mathsf{smsc}.\widehat{g}_{\mathsf{id}}, \mathsf{del}.\pi_{\mathsf{edel},\mathsf{id}})$, and do the following:

   - Set $x_{\mathsf{pb},\mathsf{id}} := (\mathsf{id}, g, \{\mathsf{smsc}.\widehat{\mathsf{sk}}_{\mathsf{idx}}\}_{\mathsf{idx}}, \mathsf{smsc}.\widehat{g}_{\mathsf{id}})$ and compute $\mathsf{h}_{\mathsf{pb},\mathsf{id}} = \mathsf{HT}.\mathsf{Hash}(\mathsf{ht}.\mathsf{hk}, x_{\mathsf{pb},\mathsf{id}})$.
   - If $\mathsf{r}\mathcal{E}\mathsf{Del}.\mathsf{Verify}(\mathsf{del}.\mathsf{crs}_{\mathsf{id}}, \mathsf{h}_{\mathsf{pb},\mathsf{id}}, \mathsf{del}.\mathsf{eh}_{\mathsf{pr},\mathsf{id}}, 1, \mathsf{del}.\pi_{\mathsf{edel},\mathsf{id}}) = 0$ or $\mathcal{M}_{\mathsf{NISC}}(x_{\mathsf{pb},\mathsf{id}}, (\mathsf{sk}_{\mathsf{id}}, r'_{\mathsf{id}}, \mathsf{smsc}.\tau_{\mathsf{id}})) = 0$ abort and output $\perp$.

6. Output whatever $\mathcal{A}$ outputs.

If $\mathcal{C}$ uses $\mathsf{SMSC}.\mathsf{InpEnc}$ and $\mathsf{SMSC}.\mathsf{FuncEnc}$ (resp. $\mathsf{SMSC}.\mathcal{S}$) in the experiment then $\mathcal{B}_{\mathsf{SMSC}}$ simulates $\mathsf{hyb}^{\mathcal{A}}_{2,n+1}$ (resp. $\mathsf{hyb}^{\mathcal{A}}_{3}$). Thus $\mathcal{B}_{\mathsf{SMSC}}$ has advantage $\epsilon(\lambda)$ in breaking the security of SMSC. □

Claim 7.11. For every $\lambda \in \mathbb{N}$, any PPT adversary $\mathcal{A}$,

$$\Pr\left[1 \leftarrow \mathsf{hyb}^{\mathcal{A}}_{3}(1^\lambda)\right] = \Pr\left[1 \leftarrow \mathsf{hyb}^{\mathcal{A}}_{4,1}(1^\lambda)\right]$$

Proof. The proof of this claim is immediate from the distributions of $\mathsf{hyb}^{\mathcal{A}}_{3}$ and $\mathsf{hyb}^{\mathcal{A}}_{4,1}$. □

Claim 7.12. If $\mathsf{r1Enc}$ is a secure rate-1 message encoding scheme, then for any PPT adversary $\mathcal{A}$, for any $j \in [n]$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for every $\lambda \in \mathbb{N}$:

$$\left|\Pr\left[1 \leftarrow \mathsf{hyb}^{\mathcal{A}}_{4,j}(1^\lambda)\right] - \Pr\left[1 \leftarrow \mathsf{hyb}^{\mathcal{A}}_{4,j+1}(1^\lambda)\right]\right| \leq \mathsf{negl}(\lambda)$$

Proof. Note that if $j \in \widetilde{H}$, $\mathsf{hyb}^{\mathcal{A}}_{4,j}$ and $\mathsf{hyb}^{\mathcal{A}}_{4,j+1}$ are identical. Hence, w.l.o.g assume $j \in H$. Assuming that there exists an adversary $\mathcal{A}$ that can distinguish between $\mathsf{hyb}^{\mathcal{A}}_{4,j}$ and $\mathsf{hyb}^{\mathcal{A}}_{4,j+1}$ such that

$$\left|\Pr\left[1 \leftarrow \mathsf{hyb}^{\mathcal{A}}_{4,j}(1^\lambda)\right] - \Pr\left[1 \leftarrow \mathsf{hyb}^{\mathcal{A}}_{4,j+1}(1^\lambda)\right]\right| = \epsilon(\lambda)$$

for some non-negligible function $\epsilon(\cdot)$, we construct an adversary $\mathcal{B}_{\mathsf{r1Enc}}$ that breaks the security of $\mathsf{r1Enc}$ as follows:

1. Sample crs similar to $\mathsf{hyb}^{\mathcal{A}}_{3}$ and send it to $\mathcal{A}$.

2. When $\mathcal{A}$ queries for input encoding of $(\mathsf{id}, x_{\mathsf{id}})$ for $\mathsf{id} \in H$, do the following.

- If $\mathsf{id} < j$, sample $\rho_{\mathsf{id}} \leftarrow \{0,1\}^{|x_{\mathsf{id}}|}$. Otherwise if $\mathsf{id} > j$, sample $(\rho_{\mathsf{id}}, \mathsf{sk}_{\mathsf{id}}) \leftarrow \mathsf{r1Enc}(1^\lambda, x_{\mathsf{id}})$. Otherwise, if $\mathsf{id} = j$, query the challenger $\mathcal{C}$ with x_{id} to receive ρ_{id}.
- Perform the rest of the steps same as $\mathsf{hyb}_3^{\mathcal{A}}$.

3. Perform the rest of the steps same as $\mathsf{hyb}_3^{\mathcal{A}}$.

If $\mathcal{C}$ uses $\mathsf{r1Enc.Encode}$ (resp. sampling ρ_j randomly) in the experiment then $\mathcal{B}_{\mathsf{r1Enc}}$ simulates $\mathsf{hyb}_{4,j}^{\mathcal{A}}$ (resp. $\mathsf{hyb}_{4,j+1}^{\mathcal{A}}$). Thus $\mathcal{B}_{\mathsf{r1Enc}}$ has advantage $\epsilon(\lambda)$ in breaking the security of $\mathsf{r1Enc}$. □

□

7.4 Semi-Maliciously Secure mrNISC with Compact Communication

In this section, we show the construction, efficiency, and security of a semi-maliciously secure mrNISC scheme where the function encodings scale only with the output size, i.e, $|\widehat{f}_i| = L \cdot \mathsf{poly}(\lambda, |I|)$ for any $i \in I$ and L is the output size of f, assuming the hardness of circularly secure LWE. Specifically, the function encodings of the mrNISC scheme depend only on the output size of f and not on the size or depth of the function f. We use a semi-maliciously secure mrNISC scheme for any poly-size function class and a multi-key homomorphic encryption scheme (Definition A.3).

Construction 7.13. Let $\mathsf{mkFHE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ be a secure multi-key fully homomorphic encryption scheme (Definition A.3) and $\mathsf{SMSC} = (\mathsf{InpEnc}, \mathsf{FuncEnc}, \mathsf{Eval})$, be a semi-maliciously secure mrNISC scheme (Definition 7.1). We provide the construction of a semi-maliciously secure mrNISC scheme with small function encodings below.

$\mathsf{Setup}(1^\lambda) \to \mathsf{crs}$**.** This probabilistic algorithm samples $\mathsf{nisc.crs} \leftarrow \mathsf{SMSC.Setup}(1^\lambda)$ and $\mathsf{fhe.pp} \leftarrow \mathsf{mkFHE.Setup}(1^\lambda)$ and outputs $\mathsf{crs} = (\mathsf{nisc.crs}, \mathsf{fhe.pp})$.

$\mathsf{InpEnc}(\mathsf{crs}, \mathsf{id}, x; r) \to (\widehat{x}, \tau)$**.** This probabilistic algorithm parses crs as $(\mathsf{nisc.crs}, \mathsf{fhe.pp})$, and does the following.

1. Sample mkFHE keys $(\mathsf{fhe.pk}, \mathsf{fhe.sk}) \leftarrow \mathsf{mkFHE.KeyGen}(\mathsf{fhe.pp})$.
2. Encrypt x using mkFHE, $\mathsf{fhe.ct} \leftarrow \mathsf{mkFHE.Enc}(\mathsf{fhe.pk}, x)$.
3. Generate input encodings $(\mathsf{nisc.}\widehat{\mathsf{sk}}, \mathsf{nisc.}\tau) \leftarrow \mathsf{SMSC.InpEnc}(\mathsf{nisc.crs}, \mathsf{id}, \mathsf{fhe.sk})$.
4. Output $\widehat{x} = (\mathsf{fhe.pk}, \mathsf{fhe.ct}, \mathsf{nisc.}\widehat{\mathsf{sk}})$ and $\tau = \mathsf{nisc.}\tau$.

$\mathsf{FuncEnc}(\mathsf{crs}, \mathsf{id}, f, \{\widehat{x}_{\mathsf{idx}}\}_{\mathsf{idx}\in I}, \tau_{\mathsf{id}}) \to \widehat{f}_{\mathsf{id}}$**.** This algorithm parses crs as $(\mathsf{nisc.crs}, \mathsf{fhe.pp})$, $\widehat{x}_{\mathsf{idx}}$ as $(\mathsf{fhe.pk}_{\mathsf{idx}}, \mathsf{fhe.ct}_{\mathsf{idx}}, \mathsf{nisc.}\widehat{\mathsf{sk}}_{\mathsf{idx}})$ for all $\mathsf{idx} \in I \subseteq [n]$, $\{\widehat{x}_{\mathsf{idx}}\}_{\mathsf{idx}}$ $(\mathsf{id} \in I)$, τ_{id} as $\mathsf{nisc.}\tau_{\mathsf{id}}$, and does the following.

1. Compute $\mathsf{fhe.}\widetilde{\mathsf{ct}}_\ell = \mathsf{mkFHE.Eval}(f_\ell, \{\mathsf{fhe.pk}_{\mathsf{idx}}, \mathsf{fhe.ct}_{\mathsf{idx}}\}_{\mathsf{idx}\in I})$ for $\ell \in [L]$ and L is the output size of f. Here, f_ℓ denotes the circuit that outputs the ℓ-th bit of $f(\{x_{\mathsf{idx}}\}_{\mathsf{idx}\in I})$.
2. Let g_ℓ denote the function $\mathsf{mkFHE.Dec}(\cdot, \ldots, \cdot, \mathsf{fhe.}\widetilde{\mathsf{ct}}_\ell)$. For each $\ell \in [L]$, sample $\widehat{g}_{\ell,\mathsf{id}} \leftarrow \mathsf{SMSC.FuncEnc}(\mathsf{nisc.crs}, \mathsf{id}, g_\ell, \{\mathsf{nisc.}\widehat{\mathsf{sk}}_{\mathsf{idx}}\}_{\mathsf{idx}\in I}, \mathsf{nisc.}\tau_{\mathsf{id}})$.
3. Output $\widehat{f}_{\mathsf{id}} = \{\mathsf{fhe.}\widetilde{\mathsf{ct}}_\ell, \widehat{g}_{\ell,\mathsf{id}}\}_{\ell\in[L]}$.

$\mathsf{Eval}(f, \{\widehat{f}_{\mathsf{id}}\}_{\mathsf{id}\in I}, \{\widehat{x}_{\mathsf{id}}\}_{\mathsf{id}\in I}) \to y$**.** This algorithm parses $\widehat{f}_{\mathsf{id}}$ as $\{\mathsf{fhe.}\widetilde{\mathsf{ct}}_{\ell,\mathsf{id}}, \widehat{g}_{\ell,\mathsf{id}}\}_{\ell\in[L]}$, $\widehat{x}_{\mathsf{id}}$ as $(\mathsf{fhe.pk}_{\mathsf{id}}, \mathsf{fhe.ct}_{\mathsf{id}}, \mathsf{nisc.}\widehat{\mathsf{sk}}_{\mathsf{id}})$ for each $\mathsf{id} \in I$, does the following.

1. If $\widetilde{\mathsf{fhe.ct}}_{\mathsf{id},\ell}$ are not the same for all $\mathsf{id} \in I$, abort and output $\bot$. Otherwise, let $\widetilde{\mathsf{fhe.ct}}_\ell$ be the common one for each $\ell \in [L]$. For any $\ell \in [L]$, if $\widetilde{\mathsf{fhe.ct}}_\ell \neq \mathsf{Eval}(f_\ell, \{\mathsf{fhe.pk}_{\mathsf{id}}, \mathsf{fhe.ct}_{\mathsf{id}}\}_{\mathsf{id}\in I})$, abort and output $\bot$.
2. Otherwise, for each $\ell \in [L]$, define g_ℓ to be the function $\mathsf{mkFHE.Dec}(\cdot, \ldots, \cdot, \mathsf{fhe.ct}_\ell)$. Compute $y_\ell = \mathsf{SMSC.Eval}(g_\ell, \{\widehat{g}_{\ell,\mathsf{id}}\}_{\mathsf{id}\in I}, \{\widehat{\mathsf{nisc.sk}}_{\mathsf{id}}\}_{\mathsf{id}\in I})$.
3. Output $y = (y_1, \ldots, y_L)$.

Theorem 7.14. Assuming the hardness of circularly secure LWE, there exists a semi-maliciously secure mrNISC scheme for all efficiently computable functions such that the size of function encodings of NISC is $L \cdot \mathsf{poly}(\lambda, |I|)$ where $|y| = L$ and $y = f(\{x_{\mathsf{id}}\}_{\mathsf{id}\in I})$ in the common random string model. In particular, function encoding sizes are independent of function size $|f|$ and input sizes $|x_{\mathsf{id}}|$.

Proof. We show that Construction 7.13 satisfies correctness and has compact function encoding sizes below.

Correctness. This follows from correctness of SMSC and multi-hop correctness of mkFHE. For instance, for $\ell = 1$, by correctness of SMSC, we have that $y_1 = \mathsf{mkFHE.Dec}(\{\mathsf{fhe.sk}_{\mathsf{id}}\}_{\mathsf{id}\in I}, \widetilde{\mathsf{fhe.ct}}_1)$ and $\widetilde{\mathsf{fhe.ct}}_1 = \mathsf{mkFHE.Eval}(f_1, \{\mathsf{fhe.pk}_{\mathsf{id}}, \mathsf{fhe.ct}_{\mathsf{id}}\}_{\mathsf{id}\in I})$. Then, by correctness of mkFHE, $y_1 = f_1(\{x_{\mathsf{id}}\}_{\mathsf{id}\in I})$. Thus, $y = f(\{x_{\mathsf{id}}\}_{\mathsf{id}\in I})$.

Function Encoding Size. Consider the case of $\ell = 1$, then size of function g_1 is $\mathsf{poly}(\lambda, |I|)$. Thus, size of $\widehat{g}_{1,\mathsf{id}}$ is $\mathsf{poly}(\lambda, |I|)$ for any $\mathsf{id} \in I$. Similarly $\widetilde{\mathsf{fhe.ct}}_{1,\mathsf{id}}$ encrypts one bit under $|I|$ secret keys. So, it's size is $\mathsf{poly}(\lambda, |I|)$. This is the same for any $\ell \in [L]$ where L is the output size of f. Hence, the size of function encoding for any $\mathsf{id} \in I$ is $L \cdot \mathsf{poly}(\lambda, |I|)$. □

Proof of static corruptions, semi-malicious, adaptive query security. We show that Construction 7.13 is semi-maliciously secure according to Definition 7.1. We define the following hybrids:

$\mathsf{hyb}_0^{\mathcal{A}}$ This is the honest experiment $\mathsf{Expt}_0^{\mathcal{C},\mathcal{A}}$ with semi-malicious adversary $\mathcal{A}$.

$\mathsf{hyb}_1^{\mathcal{A}}$ This is the same as $\mathsf{hyb}_0^{\mathcal{A}}$ except we will use the stateful simulator to simulate all encodings of SMSC.

$\mathsf{hyb}_{2,j}^{\mathcal{A}}$ This is the same as $\mathsf{hyb}_1^{\mathcal{A}}$ except that for $j \in [|H| + 1]$, we set $\mathsf{fhe.ct}_{\mathsf{id}}$ to be encryptions of $0^{|x_{\mathsf{id}}|}$ for the first $j - 1$ entries of the honest party set H. $\mathsf{hyb}_{2,|H|+1}^{\mathcal{A}}$ is the description of simulator $\mathcal{S}$.

Let $\mathsf{hyb}_{i(,j)}^{\mathcal{A}}(1^\lambda)$ denote the output of the experiment $\mathsf{hyb}_{i(,j)}^{\mathcal{A}}$.

Claim 7.15. If SMSC satisfies static corruptions, semi-malicious adversary, adaptive queries, then for any PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for every $\lambda \in \mathbb{N}$,

$$\left|\Pr\left[1 \leftarrow \mathsf{hyb}_0^{\mathcal{A}}(1^\lambda)\right] - \Pr\left[1 \leftarrow \mathsf{hyb}_1^{\mathcal{A}}(1^\lambda)\right]\right| \leq \mathsf{negl}(\lambda)$$

Proof. Assuming that there exists an adversary $\mathcal{A}$ that can distinguish between $\mathsf{hyb}_0^{\mathcal{A}}$ and $\mathsf{hyb}_1^{\mathcal{A}}$ such that $|\Pr\left[1 \leftarrow \mathsf{hyb}_0^{\mathcal{A}}(1^\lambda)\right] - \Pr\left[1 \leftarrow \mathsf{hyb}_1^{\mathcal{A}}(1^\lambda)\right]| = \epsilon(\lambda)$ for some non-negligible function $\epsilon(\cdot)$, we construct an adversary $\mathcal{B}_{\mathsf{SMSC}}$ that breaks the security of SMSC as follows:

1. $\mathcal{A}$ submits the number of parties n and the set of honest parties H. Send n, H to $\mathcal{C}$ to receive $\mathsf{nisc.crs}$. Sample $\mathsf{fhe.pp} \leftarrow \mathsf{mkFHE.Setup}(1^\lambda)$. Send $(\mathsf{nisc.crs}, \mathsf{fhe.pp})$ to $\mathcal{A}$.

2. When $\mathcal{A}$ queries for input encoding of $(\mathsf{id}, x_{\mathsf{id}})$ for $\mathsf{id} \in H$, do the following.

   - Compute $(\mathsf{fhe.pk}_{\mathsf{id}}, \mathsf{fhe.sk}_{\mathsf{id}}) \leftarrow \mathsf{mkFHE.KeyGen}(\mathsf{fhe.pp})$.
   - Encrypt x_{id} as $\mathsf{fhe.ct}_{\mathsf{id}} \leftarrow \mathsf{mkFHE.Enc}(\mathsf{fhe.pk}_{\mathsf{id}}, x_{\mathsf{id}})$.
   - Send $(\mathsf{id}, \mathsf{fhe.sk}_{\mathsf{id}})$ to $\mathcal{C}$ and receive $\mathsf{nisc.}\widehat{\mathsf{sk}}_{\mathsf{id}}$.

   Send $\widehat{x}_{\mathsf{id}} = (\mathsf{fhe.pk}_{\mathsf{id}}, \mathsf{fhe.ct}_{\mathsf{id}}, \mathsf{nisc.}\widehat{\mathsf{sk}}_{\mathsf{id}})$ to $\mathcal{A}$.

3. $\mathcal{A}$ submits input for $\mathsf{id} \in [n] \setminus H$ as $(\mathsf{id}, x_{\mathsf{id}}, \rho_{\mathsf{id}})$. Using ρ_{id}, sample $(\mathsf{fhe.pk}_{\mathsf{id}}, \mathsf{fhe.sk}_{\mathsf{id}})$, encrypt x_{id} as $\mathsf{fhe.ct}_{\mathsf{id}} \leftarrow \mathsf{mkFHE.Enc}(\mathsf{fhe.pk}_{\mathsf{id}}, x_{\mathsf{id}})$, and derive the randomness used in SMSC, ρ'_{id}. Send $(\mathsf{id}, \mathsf{fhe.sk}_{\mathsf{id}}, \rho'_{\mathsf{id}})$ to $\mathcal{C}$.

4. $\mathcal{A}$ queries for function encoding of (id, f, I). Sample $\mathsf{fhe.}\widetilde{\mathsf{ct}}_\ell = \mathsf{mkFHE.Eval}(f, \{\mathsf{fhe.pk}_{\mathsf{idx}}, \mathsf{fhe.ct}_{\mathsf{idx}}\}_{\mathsf{idx} \in I})$ for $\ell \in [L]$ and send $(\mathsf{id}, g_1, I), \ldots, (\mathsf{id}, g_L, I)$ to $\mathcal{C}$ to receive $\widehat{g}_{\ell,\mathsf{id}}$ for $\ell \in [L]$. Here, $g_\ell = \mathsf{mkFHE.Dec}(\cdot, \ldots, \cdot, \mathsf{fhe.}\widetilde{\mathsf{ct}}_\ell)$. Send $\{\mathsf{fhe.}\widetilde{\mathsf{ct}}_\ell, \widehat{g}_{\ell,\mathsf{id}}\}_{\ell \in [L]}$ to $\mathcal{A}$.

5. Output whatever $\mathcal{A}$ outputs.

If $\mathcal{C}$ is an honest challenger and uses $\mathsf{SMSC.InpEnc}$ and $\mathsf{SMSC.FuncEnc}$ (resp. $\mathsf{SMSC}.\mathcal{S}$) in the experiment then $\mathcal{B}_{\mathsf{SMSC}}$ simulates $\mathsf{hyb}_0^{\mathcal{A}}$ (resp. $\mathsf{hyb}_1^{\mathcal{A}}$). Thus $\mathcal{B}_{\mathsf{SMSC}}$ has advantage $\epsilon(\lambda)$ in breaking the security of SMSC. □

Claim 7.16. For every $\lambda \in \mathbb{N}$, any PPT adversary $\mathcal{A}$,

$$\Pr\left[1 \leftarrow \mathsf{hyb}_1^{\mathcal{A}}\right] = \Pr\left[1 \leftarrow \mathsf{hyb}_{2,1}^{\mathcal{A}}\right]$$

Proof. The proof of this claim is immediate from the distributions of $\mathsf{hyb}_1^{\mathcal{A}}$ and $\mathsf{hyb}_{2,1}^{\mathcal{A}}$. □

Lemma 7.17. If mkFHE satisfies semantic security, then for any PPT adversary $\mathcal{A}$, for any $j \in [|H|]$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for every $\lambda \in \mathbb{N}$:

$$\left|\Pr\left[1 \leftarrow \mathsf{hyb}_{2,j}^{\mathcal{A}}(1^\lambda)\right] - \Pr\left[1 \leftarrow \mathsf{hyb}_{2,j+1}^{\mathcal{A}}(1^\lambda)\right]\right| \leq \mathsf{negl}(\lambda)$$

Proof. W.l.o.g., assume that $j \in H$. Assuming that there exists an adversary $\mathcal{A}$ that can distinguish between $\mathsf{hyb}_{2,j}^{\mathcal{A}}$ and $\mathsf{hyb}_{2,j+1}^{\mathcal{A}}$ for some $j \in [|H|]$ such that $|\Pr\left[1 \leftarrow \mathsf{hyb}_{2,j}^{\mathcal{A}}(1^\lambda)\right] - \Pr\left[1 \leftarrow \mathsf{hyb}_{2,j+1}^{\mathcal{A}}(1^\lambda)\right]| = \epsilon(\lambda)$ for some non-negligible function $\epsilon(\cdot)$, we construct an adversary $\mathcal{B}_{\mathsf{mkFHE}}$ that breaks the semantic security of mkFHE as follows:

1. $\mathcal{A}$ submits the number of parties n and the set of honest parties H. Sample $\mathsf{nisc.crs} \leftarrow \mathsf{SMSC}.\mathcal{S}(1^\lambda, n, H)$. Receive $\mathsf{fhe.pp}, \mathsf{fhe.pk}$ from $\mathcal{C}$. Send $(\mathsf{nisc.crs}, \mathsf{fhe.pp})$ to $\mathcal{A}$.

2. When $\mathcal{A}$ queries for input encoding of $(\mathsf{id}, x_{\mathsf{id}})$ for $\mathsf{id} \in H$, do the following.

   - If id is the i-th entry in H and $i \neq j$, sample Compute $(\mathsf{fhe.pk}_{\mathsf{id}}, \mathsf{fhe.sk}_{\mathsf{id}}) \leftarrow \mathsf{mkFHE.KeyGen}(\mathsf{fhe.pp})$. Otherwise, set $\mathsf{fhe.}pk_{\mathsf{id}} := \mathsf{fhe.pk}$.
   - If $i < j$, encrypt $\mathsf{fhe.ct}_{\mathsf{id}} \leftarrow \mathsf{mkFHE.Enc}(\mathsf{fhe.pk}_{\mathsf{id}}, 0^{|x_{\mathsf{id}}|})$. Otherwise, if $i > j$, encrypt x_{id} as $\mathsf{fhe.ct}_{\mathsf{id}} \leftarrow \mathsf{mkFHE.Enc}(\mathsf{fhe.pk}_{\mathsf{id}}, x_{\mathsf{id}})$. Otherwise, send $(x_{\mathsf{id}}, 0^{|x_{\mathsf{id}}|})$ to $\mathcal{C}$ to receive $\mathsf{fhe.ct}_{\mathsf{id}}$.

- Sample $\mathsf{nisc.}\widehat{\mathsf{sk}}_{\mathsf{id}} \leftarrow \mathsf{SMSC}.\mathcal{S}(\mathsf{id})$.

Send $\widehat{x}_{\mathsf{id}} = (\mathsf{fhe.pk}_{\mathsf{id}}, \mathsf{fhe.ct}_{\mathsf{id}}, \mathsf{nisc.}\widehat{\mathsf{sk}}_{\mathsf{id}})$ to $\mathcal{A}$.

3. $\mathcal{A}$ submits input for $\mathsf{id} \in [n] \setminus H$ as $(\mathsf{id}, x_{\mathsf{id}}, \rho_{\mathsf{id}})$. Using ρ_{id}, sample $(\mathsf{fhe.pk}_{\mathsf{id}}, \mathsf{fhe.sk}_{\mathsf{id}})$, encrypt x_{id} as $\mathsf{fhe.ct}_{\mathsf{id}} \leftarrow \mathsf{mkFHE.Enc}(\mathsf{fhe.pk}_{\mathsf{id}}, x_{\mathsf{id}})$, and derive the randomness used in SMSC, ρ'_{id}. Compute $\mathsf{SMSC}.\mathcal{S}(\mathsf{id}, \mathsf{fhe.sk}_{\mathsf{id}}, \rho'_{\mathsf{id}})$.

4. $\mathcal{A}$ queries for function encoding of (id, f, I). Sample $\widetilde{\mathsf{fhe.ct}}_\ell = \mathsf{mkFHE.Eval}(f, \{\mathsf{fhe.pk}_{\mathsf{id}_\mathsf{x}}, \mathsf{fhe.ct}_{\mathsf{id}_\mathsf{x}}\}_{\mathsf{id}_\mathsf{x} \in I})$ for $\ell \in [L]$ and sample $\widehat{g}_{\ell,\mathsf{id}} \leftarrow \mathsf{SMSC}.\mathcal{S}(\mathsf{id}, g_\ell, I)$. Here, $g_\ell = \mathsf{mkFHE.Dec}(\cdot, \ldots, \cdot, \widetilde{\mathsf{fhe.ct}}_\ell)$. Send $\{\widetilde{\mathsf{fhe.ct}}_\ell, \widehat{g}_{\ell,\mathsf{id}}\}_{\ell \in [L]}$ to $\mathcal{A}$.

5. Output whatever $\mathcal{A}$ outputs.

If $\mathcal{C}$ encrypts x_j (resp. $0^{|x_j|}$) in the experiment then $\mathcal{B}_{\mathsf{mkFHE}}$ simulates $\mathsf{hyb}^{\mathcal{A}}_{2,j}$ (resp. $\mathsf{hyb}^{\mathcal{A}}_{2,j+1}$). Thus $\mathcal{B}_{\mathsf{mkFHE}}$ has advantage $\epsilon(\lambda)$ in breaking the semantic security of mkFHE. □

□

Corollary 7.18. Assuming $X \in \{\mathsf{LWE}\} \cup (\{\mathsf{SXDH}\} \times \{O(1)\text{-LIN}, \text{sub-exp-DDH}\})$, there exists a mrNISC scheme for any n parties and all efficiently computable functionalities.

We state the following corollary of Theorems 7.5, 5.2, 7.14, and 7.3.

Corollary 7.19. Assuming the hardness of circularly secure LWE, there exists a maliciously secure mrNISC scheme for any n parties (NISC) such that:

- The size of input encodings of NISC is $|x| + \mathsf{poly}(\lambda)$.
- The size of function encodings of NISC is $|y| \cdot \mathsf{poly}(\lambda, |I|)$ where $y = f(\{x_{\mathsf{id}}\}_{\mathsf{id} \in I})$.

8 Multi-Hop zkBARG with Somewhere-Extraction

Composable or multi-hop BARGs were introduced by Devadas et al. [DGKV22] who gave a construction for rate-1 multi-hop seBARG from rate-1 seBARG. Informally, a multi-hop BARG allows for the composition of multiple BARG proofs (which could themselves be composed BARG proofs) in a succinct manner. In this section, we extend the notion of a somewhere-extractable zkBARG to the *multi-hop* setting, and describe a construction based on a rate-1 seBARG and optimally efficient NIZK. At a high-level, a multi-hop somewhere-extractable zkBARG allows succinct batching of multiple zkBARGs. The number of *hops* (i.e., number of times zkBARGs can be successively batched) can be any polynomial, and the batch size in each hop can be set arbitrarily. Each hop increases the proof size by an *additive* $\mathsf{poly}(\lambda)$ factor. Our construction essentially combines the technique of [DGKV22]—compose proofs by creating a batch argument with the proofs as the witness—and appends a NIZK proof at the end of each level of composition with the seBARG proof as the witness. Naturally, the zero-knowledge property follows directly from that of the NIZK, but the key osbervation is that if the underlying seBARG scheme and the NIZK proof system are rate-1, then the composed proof is also rate-1. Further, if the seBARG scheme is somewhere-extractable, and the NIZK proof system is straight-line extractable then one can show, by the process of induction, that the resulting zkBARG scheme is also somewhere-extractable.

8.1 Definition

Formally, our syntax for multi-hop zkBARG (mzkBARG) follows that of multi-hop seBARG defined by [DGKV22]. More precisely, in addition to the algorithms of a plain zkBARG, we additionally require combining algorithm called ComposeProof.

Syntax. A multi-hop zkBARGscheme for an NP language $\mathcal{L}$ consists of the following polynomial time algorithms:

$\mathsf{Setup}(1^\lambda, d, (i_1, \ldots, i_d)) \to \mathsf{crs}$**.** This is a probabilistic setup algorithm. It takes as input the security parameter 1^λ, the maximum number of hops $d \in [2^\lambda]$ (i.e., the number of batch-compositions), and a sequence of d extraction indices[5] $I = (i_1, \ldots, i_d) \in [2^\lambda]^d$. It outputs the crs crs which consists of d strings $\mathsf{crs} = (\mathsf{crs}_1, \ldots, \mathsf{crs}_d)$.

$\mathsf{Prove}(\mathsf{crs}_1, (x_1, \cdots, x_k), (w_1, \ldots, w_k)) \to \pi$. The poly-time prover algorithm that takes as input crs_1 (which is the first string in crs), any (unbounded) number of instance-witness pairs of arbitrary size[6]. It runs in time $\mathsf{poly}\left(\lambda, (|x_i|, |w_i|)_{i\in[k]}\right)$ and outputs a proof π.

$\mathsf{ComposeProof}\left((\mathsf{crs}_i)_{i\in[d']}, \left(X^{(1)}, \ldots, X^{(\ell)}\right), \left(\pi^{(1)}, \ldots, \pi^{(\ell)}\right)\right) \to \pi$**.** This proof combiner is a poly-time algorithm that takes as input the $(\mathsf{crs})_{i\in[d']}$, a sequence of arbitrarily many *instance-trees* $X^{(1)}, \ldots, X^{(\ell)}$ (as defined below), of maximal depth $d'-1$, and corresponding mzkBARG proofs $\pi^{(1)}, \ldots, \pi^{(\ell)}$. It outputs a (combined) mzkBARG proof π.

$\mathsf{Verify}((\mathsf{crs}_i)_{i\in[d']}, X, \pi) \to \{0,1\}$**.** The poly-time verification algorithm takes as input $(\mathsf{crs}_i)_{i\in[d']}$, an instance-tree X of depth d', and a (combined) proof π, and outputs 0/1 (corresponding to reject or accept).

Definition 8.1 (Instance Tree)**.** An instance-tree X is a tree of varying arity, where each leaf node v is associated with an instance x_v, and each intermediate node u corresponds to a mzkBARG proof that certifies the validity of the sub-tree rooted at u (these mzkBARG proofs are not included in X).

Following the notation of Devadas et al. [DGKV22], a tree T is comprised of leaves, denoted by v, associated with a parameter $n_v = n_v(\lambda)$. A *poly-size tree*, is a tree T with at most $\mathsf{poly}(\lambda)$ nodes and $n_v = n_v(\lambda) \le \mathsf{poly}(\lambda)$ for every leaf v. For any tree T, we denote by $\mathsf{path}(T)$, the set of all the possible paths from the root to a leaf in T. An instance-tree X is said to be *consistent* with T if X has the exact same tree structure as T and for each leaf $v \in T$ the instance (leaf) x_v in X is of size n_v. For any instance-tree X that is consistent with some tree T and any $(i_1, \ldots, i_d) \in \mathsf{path}(T)$ we let $X_{i_1,\ldots,i_d}$ denote the instance (in the leaf) corresponding to the path $(i_1, \ldots, i_d)$. Finally, for a collecition of instance-trees $X^{(1)}, \ldots, X^{(\ell)}$ we denote by $\left(X^{(1)}, \ldots, X^{(\ell)}\right)$ the instance-tree that combines all the ℓ instance-trees $X^{(1)}, \ldots, X^{(\ell)}$ by adding a root with arity ℓ, whose i'th child is the root of $X^{(i)}$.

Definition 8.2. A rate-1 multi-hop somewhere-extractable zkBARG scheme $\mathsf{mzkBARG} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{ComposeProof}, \mathsf{Verify})$ for an NP language $\mathcal{L}$ is required to satisfy the following properties:

[5]The j'th extraction index i_j is interpreted as saying that, from an accepting zkBARG proof π created via j-compositions, we can efficiently extract the i_j'th witness, which is itself an accepting zkBARG proof π' created via $(j-1)$-compositions.

[6]Note that neither the batch size nor the instance size are fixed at setup time in the multi-hop setting

Efficiency. For every $i \in [d]$, the size of crs_i is at most $\mathsf{poly}(\lambda)$, and the size of a (combined) proof corresponding to an instance-tree X of depth d' is at most $m + d' \cdot \mathsf{poly}(\lambda, \log|X|)$, where m is the maximal witness length of all the leaf instances in X.

Completeness. For any $\lambda \in \mathbb{N}$, any $d \in [2^\lambda]$, any instance-tree X of size $\leq 2^\lambda$ and depth $d' \leq d$, and any corresponding valid witness W,

$$\Pr\left[\ \mathsf{Verify}\left((\mathsf{crs}_i)_{i\in[d']}, X, \pi\right) = 1 \quad : \quad \begin{array}{l}(\mathsf{crs}_i)_{i\in[d]} \leftarrow \mathsf{Setup}(1^\lambda, d, (i_1, \ldots, i_d)),\\ \pi \leftarrow \mathsf{Compose}\left((\mathsf{crs}_i)_{i\in[d']}, X, W\right)\end{array}\ \right] = 1$$

where $\mathsf{Compose}\left((\mathsf{crs}_i)_{i\in[d']}, X, W\right)$ is defined inductively on d' as follows:

- If $d' = 1$ then parse $X = (x_1, \ldots, x_\ell)$ and $W = (w_1 \ldots, w_\ell)$ and output
$$\mathsf{Compose}(\mathsf{crs}_1, X, W) = \mathsf{Prove}(\mathsf{crs}_1, x_1, \ldots, x_\ell, w_1 \ldots, w_\ell).$$
- If $d' > 1$ then parse $X = \left(X^{(1)}, \ldots, X^{(\ell)}\right)$ and $W = \left(W^{(1)}, \ldots, W^{(\ell)}\right)$, where $W^{(i)}$ is the witness corresponding to the sub-tree instance $X^{(i)}$. Denote by d_i the depth of $X^{(i)}$. For every $i \in [\ell]$ compute by induction
$$\pi^{(i)} = \mathsf{Compose}\left((\mathsf{crs}_j)_{j\in[d_i]}, X^{(i)}, W^{(i)}\right)$$
and output
$$\pi = \mathsf{ComposeProof}\left((\mathsf{crs}_i)_{i\in[d']}, X^{(1)}, \ldots, X^{(\ell)}, \pi^{(1)}, \ldots, \pi^{(\ell)}\right).$$

Index hiding. For any poly-size adversary $\mathcal{A}$, any polynomial $d = \mathsf{poly}(\lambda)$, any poly-size tree T of depth $d' \leq d$, and any sets of indices $I_0 = (i_{0,1}, \ldots, i_{0,d}), I_1 = (i_{1,1}, \ldots, i_{1,d}) \in \mathsf{path}(T)$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for every $\lambda \in \mathbb{N}$,

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

Somewhere argument of knowledge. There exists a stateful PPT extractor $\mathcal{E}$ such that for any poly-size adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for any polynomial $d = \mathsf{poly}(\lambda)$, any poly-size tree T of depth $d' \leq d$, and any set of indices $(i_1, \ldots, i_{d'}) \in \mathsf{path}(T)$ and $i_{d'+1} \ldots, i_d \in [2^\lambda]$, for every $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l}\mathsf{Verify}(\mathsf{crs}, X, \pi) = 1\\ \wedge\ X \text{ is consistent with } T\\ \wedge\ W^* \text{ is not a valid witness for } X_{i_1,\ldots,i_{d'}} \in \mathcal{L}\end{array} \quad : \quad \begin{array}{l}(\mathsf{crs}, \mathsf{td}) \leftarrow \mathcal{E}(1^\lambda, d, (i_1, \ldots, i_d))\\ (X, \pi) \leftarrow \mathcal{A}(\mathsf{crs})\\ W^* \leftarrow \mathcal{E}\left(\mathsf{td}, X, \pi\right)\end{array}\right] \leq \mathsf{negl}(\lambda).$$

Zero-knowledge. There exist PPT simulators $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3)$ sharing state-information for any poly-size adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for any polynomial $d = \mathsf{poly}(\lambda)$, any poly-size tree T of depth $d' \leq d$, and any set of indices $(i_1, \ldots, i_{d'}) \in \mathsf{path}(T)$ and $i_{d'+1} \ldots, i_d \in [2^\lambda]$, for every $\lambda \in \mathbb{N}$,

$$\left|\begin{array}{l}\Pr\left[\ 1 \leftarrow \mathcal{A}^{\mathsf{Prove}(\mathsf{crs},\cdot,\cdot),\mathsf{ComposeProof}(\mathsf{crs},\cdot,\cdot)}(\mathsf{crs}) \quad : \quad \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, d, (i_1, \ldots, i_d))\ \right]\\ \qquad - \Pr\left[\ 1 \leftarrow \mathcal{A}^{\mathcal{O}^{\mathcal{S}_2}(\mathsf{crs}_1,\cdot,\cdot),\mathcal{O}^{\mathcal{S}_3}(\mathsf{crs},\cdot,\cdot)}(\mathsf{crs}) \quad : \quad \mathsf{crs} \leftarrow \mathcal{S}_1(1^\lambda, d)\ \right]\end{array}\right| \leq \mathsf{negl}(\lambda)$$

where $\mathcal{O}^{\mathcal{S}_2}((x_1, \cdots, x_k), (w_1, \ldots, w_k))$ outputs $\mathcal{S}_2(x_1, \ldots, x_k)$ if for every $i \in [k]$, w_i is a valid witness for $x_i \in \mathcal{L}$ and $\perp$ otherwise; and $\mathcal{O}^{\mathcal{S}_3}\left(\left(X^{(1)}, \ldots, X^{(\ell)}\right), \left(\pi^{(1)}, \ldots, \pi^{(\ell)}\right)\right)$ outputs $\mathcal{S}_3\left(X^{(1)}, \ldots, X^{(\ell)}\right)$ if for every $i \in [k]$, $\mathsf{Verify}\left((\mathsf{crs}_i)_{i\in[d_i]}, X^{(i)}, \pi^{(i)}\right) = 1$ and $\perp$ otherwise;

8.2 Construction

We now give a construction for rate-1 multi-hop somewhere-extractable zkBARG from any rate-1 seBARG and an optimally efficient NIZK. Our multi-hop zkBARG construction, denoted mzkBARG, for some NP language $\mathcal{L}$ uses the single-hop seBARG primitive, denoted s-seBARG, for the language $\mathcal{L}'$ containing $\mathcal{L}$. Essentially, an s-seBARG is an seBARG where the number of instances and the input length are not a priori bounded. The existence of s-seBARG was shown by [DGKV22] and we recall their construction next.

Single-hop seBARG. A single-hop seBARG is a seBARG where the number of batched instances k, and the input length n can be determined when batching the arguments. The construction of s-seBARG from a regular seBARG follows from the observation made in [DGKV22] that (i) one can always pad inputs of unequal length; and more importantly, (ii) it is possible to run the setup algorithm for zkBARG, which grows only polylogarithmically in k and n, once for each $k, n \in \{2^i\}_{i\in[\lambda]}$ and allow the prover to choose the appropriate crs based on the actual k and n.

Multi-hop zkBARG. We will now use an s-seBARG for the language $\mathcal{L}'$ and NIZK for the language $\mathcal{L}''$. Here, any instance in $\mathcal{L}'$ is an instance-tree X of some depth d', along with $(\mathsf{crs}_i)_{i\in[d']}$ (if $d' = 0$ then there is no crs associated with the instance) and a valid witness is a valid aggregated proof corresponding to $(\mathsf{crs}_i)_{i\in[d']}$ (where if $d' = 0$ then a valid proof is simply a valid witness corresponding to $\mathcal{L}$); and any instance of $\mathcal{L}''$ is the input to the s-seBARG.Verify circuit, and the witness is the corresponding s-seBARG proof.

$\mathsf{Setup}(1^\lambda, d, I) \to \mathsf{crs}$. Parse $I = (i_1, \ldots, i_d)$ and for every $j \in [d]$, sample $\mathsf{barg.crs}_j \leftarrow \mathsf{s\text{-}seBARG.Setup}(1^\lambda, i_j)$ and $\mathsf{nizk.crs}_j \leftarrow \mathsf{NIZK.Setup}(1^\lambda)$. Finally, output $\mathsf{crs} = (\mathsf{nizk.crs}_j, \mathsf{barg.crs}_j)_{j\in[d]}$.

$\mathsf{Prove}(\mathsf{crs}_1, (x_1, \ldots, x_k), (w_1, \ldots, w_k)) \to \pi$. Parse crs_1 as $\mathsf{nizk.crs}_1$ and $\mathsf{barg.crs}_1$. Compute

$$\mathsf{barg}.\pi \leftarrow \mathsf{s\text{-}seBARG.Prove}(\mathsf{barg.crs}_1, (x_1, \ldots, x_k), (w_1, \ldots, w_k))$$

and output,

$$\mathsf{nizk}.\pi \leftarrow \mathsf{NIZK.Prove}(\mathsf{nizk.crs}_1, (\mathsf{barg.crs}_1, (x_1, \ldots, x_k)), \mathsf{barg}.\pi)$$

as the proof π.

$\mathsf{ComposeProof}\left(\mathsf{crs}, \left(X^{(1)}, \ldots, X^{(\ell)}\right), \left(\pi^{(1)}, \ldots, \pi^{(\ell)}\right)\right) \to \pi$. For every $i \in [\ell]$, denote by d_i the depth of $X^{(i)}$ and let $d' = \max\{d_i\} + 1$. If $d' > d$ then the abort. Otherwise, parse $\mathsf{crs} = ((\mathsf{nizk.crs}_1, \mathsf{barg.crs}_1), \ldots, (\mathsf{nizk.crs}_{d'}, \mathsf{barg.crs}_{d'}, \ldots))$ and compute

$$\mathsf{barg}.\pi \leftarrow \mathsf{s\text{-}seBARG.Prove}\left(\mathsf{barg.crs}_{d'}, \left((\mathsf{barg.crs}_j)_{j\in[d_i]}, X^{(i)}\right)_{i\in[\ell]}, \left(\pi^{(i)}\right)_{i\in[\ell]}\right)$$

where a valid witness corresponding to $\left((\mathsf{barg.crs}_j)_{j\in[d_i]}, X^{(i)}\right) \in \mathcal{L}'$ is $\pi^{(i)}$ such that

$$\mathsf{Verify}\left((\mathsf{barg.crs}_j)_{j\in[d_i]}, X^{(i)}, \pi^{(i)}\right) = 1$$

where if $d_i = 0$ then $\mathsf{Verify}\left(\bot, X^{(i)}, \pi^{(i)}\right) = 1$ if and only if $\pi^{(i)}$ is a valid witness for $X^{(i)} \in \mathcal{L}$.

Finally, output

$$\mathsf{nizk}.\pi \leftarrow \mathsf{NIZK.Prove}\left(\mathsf{nizk.crs}_{d'}, \left(\mathsf{barg.crs}_{d'}, \left((\mathsf{barg.crs}_j)_{j\in[d_i]}, X^{(i)}\right)_{i\in[\ell]}\right), \mathsf{barg}.\pi\right)$$

as the proof π.

$\mathsf{Verify}(\mathsf{crs}, X, \pi) \to 0/1$. Parsing $X = \left(X^{(1)}, \ldots, X^{(\ell)}\right)$, let $d_i \in [d]$ be the depth of the instance-tree $X^{(i)}$. If $d' \neq \max\{d_i\}_{i[\ell]} + 1$ then output 0. Otherwise parse $\mathsf{crs} = ((\mathsf{nizk.crs}_1, \mathsf{barg.crs}_1), \ldots, (\mathsf{nizk.crs}_{d'}, \mathsf{barg.crs}_{d'}, \ldots))$ and then output

$$\mathsf{NIZK.Verify}\left(\mathsf{nizk.crs}_{d'}, \left(\mathsf{barg.crs}_{d'}, \left((\mathsf{barg.crs}_j)_{j\in[d_i]}, X^{(i)}\right)_{i\in[\ell]}\right), \pi\right)$$

Theorem 8.3. If s-seBARG is a rate-1 single-hop seBARG and NIZK is an optimally efficient NIZK scheme (with straight-line extraction) then Construction 8.2 is a rate-1 multi-hop somewhere extractable zkBARG scheme.

Lemma 8.4 (Completeness)**.** If s-seBARG and NIZK are complete, then Construction 8.2 is complete.

Proof. This follows directly from the construction. □

Lemma 8.5 (Efficiency)**.** If s-seBARG and NIZK are rate-1, then Construction 8.2 is rate-1.

Proof. This follows directly from the construction. □

8.3 Security Analysis

Lemma 8.6 (Index hiding)**.** If s-seBARG is index hiding, then Construction 8.2 is index hiding.

Proof. This follows directly from the construction. □

Lemma 8.7 (Somewhere argument of knowledge)**.** If s-seBARG is somewhere argument of knowledge and NIZK is straight-line extractable, then Construction 8.2 is somewhere argument of knowledge.

Proof. Let $\mathsf{s\text{-}seBARG}.\mathcal{E}$ (resp. $\mathsf{NIZK}.\mathcal{E}$) denote the PPT extractor corresponding to the underlying s-seBARG (resp. NIZK) scheme. We define an extractor $\mathcal{E}$ that given (td, X, π) does the following:

1. Parse $\mathsf{td} = ((\mathsf{nizk.td}_1, \mathsf{barg.td}_1), \ldots, (\mathsf{nizk.td}_d, \mathsf{barg.td}_d))$ and for $j \in [d]$, let each $\mathsf{barg.td}_j$ consist of index i_j. We also assume w.l.o.g. that each $\mathsf{nizk.td}_j$ implicitly contains $\mathsf{nizk.crs}_{i<j}$ (similarly for $\mathsf{barg.td}_j$).
2. Parse $X = \left(X^{(1)}, \ldots, X^{(\ell)}\right)$, and let d' denote the depth of X.
3. For every $j \in [d']$ we denote by d_j the depth of $X^{(i_{d'}, \ldots, i_{d'-j+1})}$, the subtree of X obtained by going down from the root on the path $(i_{d'}, \ldots, i_{d'-j+1})$. Let $j^* \in [d']$ be the smallest index such that $d_j = 0$ for for every $j \geq j^*$ (i.e., j^* is the length of the path $(i_{d'}, \ldots, i_1)$ in X until we reach a leaf).

4. Run the NIZK extractor to compute

$$\mathsf{barg}.\pi^{(i_{d'})} \leftarrow \mathsf{NIZK}.\mathcal{E}\left(\mathsf{nizk.td}_{d'}, \left(X^{(1)}, \ldots, X^{(\ell)}\right), \pi\right).$$

5. Then, run the s-seBARG extractor to compute

$$\pi^{(i_{d'})} \leftarrow \mathsf{s\text{-}seBARG}.\mathcal{E}\left(\mathsf{barg.td}_{d'}, \left(X^{(1)}, \ldots, X^{(\ell)}\right), \mathsf{barg}.\pi^{(i_{d'})}\right).$$

   Intuitively, if π is a valid mzkBARG proof for X then with overwhelming probability $\pi^{(i_{d'})}$ is a valid mzkBARG proof for the instance-tree $X^{(i_{d'})}$.

6. For every $j \in [j^* - 1]$ parse $X^{(i_{d'}, \ldots, i_{d'-j+1})} = \left(X^{(1)}, \ldots, X^{(\ell_j)}\right)$, and then inductively compute (starting with $j = 1$)

$$\mathsf{barg}.\pi^{(i_{d'-j})} \leftarrow \mathsf{NIZK}.\mathcal{E}\left(\mathsf{nizk.td}_{d_j}, \left(X^{(1)}, \ldots, X^{(\ell_j)}\right), \pi^{(i_{d'}, \ldots, i_{d'-j+1})}\right).$$

   and,

$$\pi^{(i_{d'}, \ldots, i_{d'-j})} \leftarrow \mathsf{s\text{-}seBARG}.\mathcal{E}\left(\mathsf{barg.td}_{d_j}, \left(X^{(1)}, \ldots, X^{(\ell_j)}\right), \mathsf{barg}.\pi^{(i_{d'}, \ldots, i_{d'-j})}\right)$$

7. Output $W^* = \pi^{(i_{d'}, \ldots, i_{d'-j^*+1})}$.

Now, fix any PPT adversary $\mathcal{A}$ and any polynomial $d = \mathsf{poly}(\lambda)$. We will argue by induction that for every $d' \leq d$ there exists a negligible function $\mu_{d'}$, such that for any poly-size tree T of depth d', and any set of indices $(i_1, \ldots, i_{d'}) \in \mathsf{path}(T)$ and $i_{d'+1} \ldots, i_d \in [2^\lambda]$ it holds that for every $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{ll} \mathsf{Verify}((\mathsf{crs}_j)_{j\in[d']}, X, \pi) = 1 & (\mathsf{crs}, \mathsf{td}) \leftarrow \mathcal{E}(1^\lambda, d, (i_1, \ldots, i_d)) \\ \wedge\ X \text{ is consistent with } T & : \ (X, \pi) \leftarrow \mathcal{A}(\mathsf{crs}) \\ \wedge\ W^* \text{ is not a valid witness for } X_{i_1,\ldots,i_{d'}} \in \mathcal{L} & W^* \leftarrow \mathcal{E}\left(\mathsf{td}, X, \pi\right) \end{array}\right] \leq \mu_{d'}(\lambda). \tag{14}$$

Base case: $d' = 1$: Let us define the following two types of possible PPT adversaries:

Type 1. For this type of adversary, the following probability is non-negligible,

$$\Pr\left[\begin{array}{ll} \mathsf{Verify}(\mathsf{crs}_1, (x_1, \ldots, x_k), \pi) = 1 \wedge & (\mathsf{crs}, \mathsf{td}) \leftarrow \mathcal{E}(1^\lambda, d, (i_1, \ldots, i_d)) \\ \mathsf{Verify}(\mathsf{crs}_1, (x_1, \ldots, x_k), \mathsf{barg}.\pi) \neq 1 & ((x_1, \ldots, x_k), \pi) \leftarrow \mathcal{A}(\mathsf{crs}) \\ & : \ \mathsf{barg}.\pi \leftarrow \mathsf{NIZK}.\mathcal{E}\left(\mathsf{nizk.td}_1, (x_1, \ldots, x_k), \pi\right) \\ & w^* \leftarrow \mathsf{s\text{-}seBARG}.\mathcal{E}(\mathsf{td}_1, (x_1, \ldots, x_k), \mathsf{barg}.\pi) \\ & \text{parse } \mathsf{crs} = (\mathsf{crs}_1, \ldots, \mathsf{crs}_d) \end{array}\right] \tag{15}$$

Then, we give a reduction $\mathcal{B}_{\mathsf{NIZK}.\mathcal{E}}$ that breaks straight-line extraction of NIZK as follows:

1. On receiving $\overline{\mathsf{crs}}$ from the challenger, $\mathcal{B}_{\mathsf{NIZK}.\mathcal{E}}$ sets $\mathsf{nizk.crs}_1 = \overline{\mathsf{crs}}$.
2. It then generates the rest of the crs honestly and sends it to the type 1 adversary.
3. Finally it outputs whatever the adversary outputs.

It is clear that if the type 1 adversary has non-negligible probability for (15), then $\mathcal{B}_{\mathsf{NIZK}.\mathcal{E}}$ breaks straight-line extraction of NIZK with the same probability.

Type 2. For this type of adversary, the following probability is non-negligible,

$$\Pr\left[\begin{array}{l} \\ \mathsf{Verify}(\mathsf{crs}_1,(x_1,\ldots,x_k),\pi)=1 \wedge \\ \mathsf{Verify}(\mathsf{crs}_1,(x_1,\ldots,x_k),\mathsf{barg}.\pi)=1 \\ \wedge\ w^* \text{ is not a valid witness for } x_{i_1}\in\mathcal{L} \\ \\ \end{array} : \begin{array}{l} (\mathsf{crs},\mathsf{td})\leftarrow\mathcal{E}(1^\lambda,d,(i_1,\ldots,i_d)) \\ ((x_1,\ldots,x_k),\pi)\leftarrow\mathcal{A}(\mathsf{crs}) \\ \mathsf{barg}.\pi\leftarrow\mathsf{NIZK}.\mathcal{E}\left(\mathsf{nizk}.\mathsf{td}_1,(x_1,\ldots,x_k),\pi\right) \\ w^*\leftarrow\mathsf{s\text{-}seBARG}.\mathcal{E}(\mathsf{td}_1,(x_1,\ldots,x_k),\mathsf{barg}.\pi) \\ \text{parse } \mathsf{crs}=(\mathsf{crs}_1,\ldots,\mathsf{crs}_d) \end{array}\right] \tag{16}$$

Then, we give a reduction $\mathcal{B}_{\mathsf{seBARG}.\mathcal{E}}$ that breaks somewhere argument of knowledge of seBARG as follows:

1. On receiving $\overline{\mathsf{crs}}$ from the challenger, $\mathcal{B}_{\mathsf{seBARG}.\mathcal{E}}$ sets $\mathsf{barg}.\mathsf{crs}_1=\overline{\mathsf{crs}}$.
2. It then generates the rest of the crs honestly and sends it to the type 2 adversary.
3. On receiving $((x_1,\ldots,x_k),\pi)$ from the adversary, it runs the NIZK extractor, to obtain $\mathsf{barg}.\pi$.
4. It then outputs $((x_1,\ldots,x_k),\mathsf{barg}.\pi)$.

It is clear that if the type 2 adversary has non-negligible probability for (16), then $\mathcal{B}_{\mathsf{seBARG}.\mathcal{E}}$ breaks somewhere argument of knowledge of the underlying seBARG scheme with the same probability.

Intermediate step: Before proceeding with the induction step, we must first show that there exists a negligible function ν such that for every $\lambda\in\mathbb{N}$,

$$\Pr\left[\begin{array}{l} \mathsf{Verify}((\mathsf{crs}_j)_{j\in[d']},X,\pi)=1 \wedge \\ X \text{ is consistent with } T \wedge \\ \left(\begin{array}{l}\mathsf{Verify}\left((\mathsf{crs}_j)_{j\in[d_1]},X^{(i_{d'})},\mathsf{barg}.\pi^{(i_{d'})}\right)\neq 1 \\ \vee\ \mathsf{Verify}\left((\mathsf{crs}_j)_{j\in[d_1]},X^{(i_{d'})},\pi^{(i_{d'})}\right)\neq 1\end{array}\right) \end{array} : \begin{array}{l} (\mathsf{crs},\mathsf{td})\leftarrow\mathcal{E}(1^\lambda,d,(i_1,\ldots,i_d)) \\ (X,\pi)\leftarrow\mathcal{A}(\mathsf{crs}) \\ \mathsf{barg}.\pi^{(i_{d'})}\leftarrow\mathsf{NIZK}.\mathcal{E}\left(\mathsf{nizk}.\mathsf{td}_{d'},X,\pi\right) \\ \pi^{(i_{d'})}\leftarrow\mathsf{s\text{-}seBARG}.\mathcal{E}\left(\mathsf{td}_{d'},X,\mathsf{barg}.\pi^{(i_{d'})}\right) \\ \text{parse } \mathsf{crs}=(\mathsf{crs}_1,\ldots,\mathsf{crs}_d) \end{array}\right]\leq\nu(\lambda). \tag{17}$$

Let us again define two types of possible PPT adversaries:

Type 1. For this type of adversary, the following probability is non-negligible,

$$\Pr\left[\begin{array}{l} \\ \mathsf{Verify}((\mathsf{crs}_j)_{j\in[d']},X,\pi)=1 \wedge \\ X \text{ is consistent with } T \wedge \\ \mathsf{Verify}\left((\mathsf{crs}_j)_{j\in[d_1]},X^{(i_{d'})},\mathsf{barg}.\pi^{(i_{d'})}\right)\neq 1 \\ \\ \end{array} : \begin{array}{l} (\mathsf{crs},\mathsf{td})\leftarrow\mathcal{E}(1^\lambda,d,(i_1,\ldots,i_d)) \\ (X,\pi)\leftarrow\mathcal{A}(\mathsf{crs}) \\ \mathsf{barg}.\pi^{(i_{d'})}\leftarrow\mathsf{NIZK}.\mathcal{E}\left(\mathsf{nizk}.\mathsf{td}_{d'},X,\pi\right) \\ \pi^{(i_{d'})}\leftarrow\mathsf{s\text{-}seBARG}.\mathcal{E}\left(\mathsf{td}_{d'},X,\mathsf{barg}.\pi^{(i_{d'})}\right) \\ \text{parse } \mathsf{crs}=(\mathsf{crs}_1,\ldots,\mathsf{crs}_d) \end{array}\right] \tag{18}$$

Then, we give a reduction $\mathcal{B}_{\mathsf{NIZK}.\mathcal{E}}$ that breaks straight-line extraction of NIZK as follows:

1. On receiving $\overline{\mathsf{crs}}$ from the challenger, $\mathcal{B}_{\mathsf{NIZK}.\mathcal{E}}$ sets $\mathsf{nizk}.\mathsf{crs}_{d'}=\overline{\mathsf{crs}}$.

2. It then generates the rest of the crs honestly and sends it to the type 1 adversary.

3. Finally it outputs whatever the adversary outputs.

It is clear that if the type 1 adversary has non-negligible probability for (18), then $\mathcal{B}_{\mathsf{NIZK}.\mathcal{E}}$ breaks straight-line extraction of NIZK with the same probability.

Type 2. For this type of adversary, the following probability is non-negligible,

$$\Pr\left[\begin{array}{l|l} \mathsf{Verify}((\mathsf{crs}_j)_{j\in[d']}, X, \pi) = 1 \wedge & (\mathsf{crs}, \mathsf{td}) \leftarrow \mathcal{E}(1^\lambda, d, (i_1, \ldots, i_d)) \\ X \text{ is consistent with } T \wedge & (X, \pi) \leftarrow \mathcal{A}(\mathsf{crs}) \\ \mathsf{Verify}\left((\mathsf{crs}_j)_{j\in[d_1]}, X^{(i_{d'})}, \mathsf{barg}.\pi^{(i_{d'})}\right) \neq 1 & : \mathsf{barg}.\pi^{(i_{d'})} \leftarrow \mathsf{NIZK}.\mathcal{E}\left(\mathsf{nizk}.\mathsf{td}_{d'}, X, \pi\right) \\ \wedge\ \mathsf{Verify}\left((\mathsf{crs}_j)_{j\in[d_1]}, X^{(i_{d'})}, \pi^{(i_{d'})}\right) \neq 1 & \pi^{(i_{d'})} \leftarrow \mathsf{s\text{-}seBARG}.\mathcal{E}\left(\mathsf{td}_{d'}, X, \mathsf{barg}.\pi^{(i_{d'})}\right) \\ & \text{parse } \mathsf{crs} = (\mathsf{crs}_1, \ldots, \mathsf{crs}_d) \end{array}\right] \tag{19}$$

Then, we give a reduction $\mathcal{B}_{\mathsf{seBARG}.\mathcal{E}}$ that breaks somewhere argument of knowledge of seBARG as follows:

1. On receiving $\overline{\mathsf{crs}}$ from the challenger, $\mathcal{B}_{\mathsf{seBARG}.\mathcal{E}}$ sets $\mathsf{barg}.\mathsf{crs}_{d'} = \overline{\mathsf{crs}}$.

2. It then generates the rest of the crs honestly and sends it to the type 2 adversary.

3. On receiving (X, π) from the adversary, it runs the NIZK extractor, to obtain $\mathsf{barg}.\pi^{(i_{d'})}$.

4. It then outputs $\left(X, \mathsf{barg}.\pi^{(i_{d'})}\right)$.

It is clear that if the type 2 adversary has non-negligible probability for (19), then $\mathcal{B}_{\mathsf{seBARG}.\mathcal{E}}$ breaks somewhere argument of knowledge of the underlying seBARG scheme with the same probability.

Induction step: Now, suppose that Equation (14) holds for every $j < d'$ and we prove that it holds for d'. To that end, consider the poly-size adversary $\mathcal{A}'$ that given crs does the following:

1. Parse $\mathsf{crs} = ((\mathsf{nizk}.\mathsf{crs}_1, \mathsf{barg}.\mathsf{crs}_1), \ldots, (\mathsf{nizk}.\mathsf{crs}_d, \mathsf{barg}.\mathsf{crs}_d))$.

2. Generate $(\mathsf{barg}.\mathsf{crs}^*, \mathsf{barg}.\mathsf{td}^*) \leftarrow \mathsf{s\text{-}seBARG}.\mathcal{E}(1^\lambda, i_{d'})$ and $(\mathsf{nizk}.\mathsf{crs}^*, \mathsf{nizk}.\mathsf{td}^*) \leftarrow \mathsf{NIZK}.\mathcal{E}(1^\lambda)$.

3. Let $\mathsf{crs}^* = ((\mathsf{nizk}.\mathsf{crs}_1, \mathsf{barg}.\mathsf{crs}_1), \ldots, (\mathsf{nizk}.\mathsf{crs}_{d'-1}, \mathsf{barg}.\mathsf{crs}_{d'-1}), (\mathsf{nizk}.\mathsf{crs}^*, \mathsf{barg}.\mathsf{crs}^*), (\mathsf{nizk}.\mathsf{crs}_{d'+1}, \mathsf{barg}.\mathsf{crs}_{d'+1})$ $\ldots, (\mathsf{nizk}.\mathsf{crs}_d, \mathsf{barg}.\mathsf{crs}_d))$.

4. Compute $(X, \pi) \leftarrow \mathcal{A}(\mathsf{crs}^*)$.

5. Compute $\mathsf{barg}.\pi^{(i_{d'})} \leftarrow \mathsf{NIZK}.\mathcal{E}\left(\mathsf{nizk}.\mathsf{td}^*, X, \pi\right)$ and $\pi^{(i_{d'})} \leftarrow \mathsf{s\text{-}seBARG}.\mathcal{E}\left(\mathsf{td}_{d'}, X, \mathsf{barg}.\pi^{(i_{d'})}\right)$.

6. Output $\left(X^{(i_{d'})}, \pi^{(i_{d'})}\right)$.

By the induction hypothesis there exists a negligible function μ such that for every $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l} \mathsf{Verify}\left((\mathsf{crs}_j)_{j\in[d_1]}, X^{(i_{d'})}, \pi^{(i_{d'})}\right) = 1 \\ \wedge\ X^{(i_{d'})} \text{ is consistent with } T^{(i_{d'})} \\ \wedge\ \mathsf{Verify}\left((\mathsf{crs}_j)_{j\in[d_2]}, X^{(i_{d'},i_{d'-1})}, \pi^{(i_{d'},i_{d'-1})}\right) \neq 1 \end{array} : \begin{array}{l} (\mathsf{crs},\mathsf{td}) \leftarrow \mathcal{E}(1^\lambda, d, (i_1,\ldots,i_d)) \\ \left(X^{(i_{d'})}, \pi^{(i_{d'})}\right) \leftarrow \mathcal{A}'(\mathsf{crs}) \\ \pi^{(i_{d'},i_{d'-1})} \leftarrow \mathcal{E}\left(\mathsf{td}^*, X^{(i_{d'})}, \pi^{(i_{d'})}\right) \end{array}\right] \leq \mu(\lambda).$$

This, together with Equation (17), implies the induction step (i.e., Equation (14)). □

Lemma 8.8 (Zero-knowledge)**.** If NIZK satisfies zero-knowledge, then Construction 8.2 satisfies

Proof. We define the following hybrids:

$\mathsf{hyb}_0^{\mathcal{A}}$ This is the original zero-knowledge experiment described in Definition 8.2.

$\mathsf{hyb}_1^{\mathcal{A}}$ This is same as hyb_0 except that we use $\mathsf{NIZK}.\mathcal{S}$ to simulate NIZK instantiation. This is the description of the simulator $\mathcal{S}$.

Let $\mathsf{hyb}_i^{\mathcal{A}}(1^\lambda)$ denote the output of the experiment $\mathsf{hyb}_i^{\mathcal{A}}$.

Claim 8.9. If NIZK satisfies zero-knowledge, then there exists a negligible function $\mathsf{negl}(\cdot)$ such that for every $\lambda \in \mathbb{N}$:

$$\left|\Pr\left[1 \leftarrow \mathsf{hyb}_0^{\mathcal{A}}(1^\lambda)\right] - \Pr\left[1 \leftarrow \mathsf{hyb}_1^{\mathcal{A}}(1^\lambda)\right]\right| \leq \mathsf{negl}(\lambda)$$

Proof. This follows directly from the zero-knowledge property of NIZK. □

□

Corollary 8.10. Assuming $X \in \{\mathsf{LWE}, O(1)\text{-}\mathsf{LIN}, \text{sub-exp-}\mathsf{DDH}\}$, there exists a rate-1 multi-hop somewhere-extractable zkBARG scheme.

9 Publicly-Verifiable Succinct Delegation of Committed Programs

In this section, we show that reusable encrypted RAM delegation implies a stronger variant of publicly-verifiable succinct delegation of committed programs introduced by Ghosal et al. [GSW23] for fixed programs. The problem of sDel considers the scenario where a client delegates the computation of its program to a server, so that a third-party user, holding a commitment to the client's program, can later request the server for the output said program on some public input. Importantly, the third-party user need not trust the server is additionally required to provide a proof of correct computation with respect to the committed program.

As we explain below, our sDel construction satisfies a stronger notion soundness as well as straight-line extraction. The stronger soundness property is essentially the no-equivocation property for r$\mathcal{E}$Del proofs, namely that an adversary cannot furnish proofs of contradictory outputs for the same committed program. Moreover, our construction directly yields the zero-knowledge version of sDel, called zksDel where the verifier does not learn anything about the committed program.

9.1 Definition

We recall the syntax of zksDel given in [GSW23], with a mild modification. Specifically, instead of a "program authoring" algorithm $\mathsf{ProgAuth}$ that *outputs* the program and its commitment, we consider the program to be the input to an encoding algorithm that outputs the commitment and some auxiliary information. This separates the logic of the program from the algorithm generating its commitment and allows us to define a delegation scheme with respect to a program.

Syntax. A zksDel scheme with respect to a program $P \in \{0,1\}^m$ consists of the following polynomial time algorithms:

$\mathsf{Setup}(1^\lambda, n, 1^m, T) \to \mathsf{crs}$**.** This is a probabilistic setup algorithm. It takes as input the security parameter 1^λ, the input length n to the program, the size m of the program, the maximum runtime T of the program and outputs a common reference string crs.

$\mathsf{ProgEnc}(\mathsf{crs}, P) \to (c_P, \mathsf{aux})$**.** The poly-time program encoding algorithm that takes as input crs and the program $P \in \{0,1\}^m$, $m \leq 2^\lambda \in \mathbb{N}$. It outputs a commitment c_P to P and some auxiliary information aux, which includes P itself.

$\mathsf{Prove}(\mathsf{crs}, x, \mathsf{aux}) \to (y, \pi)$**.** The poly-time prover algorithm that takes as input crs, some program input $x \in \{0,1\}^n$ and some auxiliary information aux. It outputs a value $y \in \{0,1\}$ and a proof π.

$\mathsf{Verify}(\mathsf{crs}, x, y, c_P, \pi) \to \{0,1\}$**.** The poly-time verification algorithm takes as input crs, some program input x, a program output y, commitment c_P and the proof π. It outputs either 1 (accept) or 0 (reject).

Definition 9.1. A zksDel scheme $\mathsf{zksDel} = (\mathsf{Setup}, \mathsf{ProgEnc}, \mathsf{Prove}, \mathsf{Verify})$ is required to satisfy the following properties:

Completeness. For any $\lambda, n, m \in \mathbb{N}$, for all $x \in \{0,1\}^n$ and for all $P \in \{0,1\}^m$ such that $n, m < 2^\lambda$,

$$\Pr\left[P(x) = y \ \wedge \ \mathsf{Verify}\left(\mathsf{crs}, x, y, c_P, \pi\right) = 1 \quad : \quad \begin{array}{l} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, n, 1^m, T), \\ (c_P, \mathsf{aux}) \leftarrow \mathsf{ProgEnc}(\mathsf{crs}, P), \\ (y, \pi) \leftarrow \mathsf{Prove}(\mathsf{crs}, x, \mathsf{aux}) \end{array} \right] = 1$$

Efficiency. For the completeness experiment above,

- Setup's running time and the size of the crs are $\mathsf{poly}(\lambda, \log T, \log n, m)$.
- The size of c_P is $\mathsf{poly}(\lambda, m)$.
- Prover's running time is $\mathsf{poly}(\lambda, T, n, m)$.
- The size of π is $\mathsf{poly}(\lambda, \log T, \log n, \log m)$.
- The Verifier's running time is $\mathsf{poly}(\lambda, \log T, \log n, m)$.

$\mathsf{Expt}_0^{\mathcal{C},\mathcal{A}}(1^\lambda, n, 1^m, T)$. This is the Real experiment parameterized by an honest challenger $\mathcal{C}$. $\mathcal{A}$ receives $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, n, 1^m, T)$ from $\mathcal{C}$. $\mathcal{A}$ makes the following queries in an adaptive manner. After this, $\mathcal{A}$ outputs guess b'. Output b'.

- PROGRAM ENCODING: $\mathcal{A}$ sends program P of length m and runtime T, and receives (c_P, i) where $(c_P, \mathsf{aux}_i) \leftarrow \mathsf{ProgEnc}(\mathsf{crs}, P)$ and i is an index that after each query of this type is incremented.
- PROVE: $\mathcal{A}$ sends (x, i) and receives (y, π) where $(y, \pi) \leftarrow \mathsf{Prove}(\mathsf{crs}, x, \mathsf{aux}_i)$ and $y = P(x)$ (for P corresponding to i).

$\mathsf{Expt}_1^{\mathcal{S},\mathcal{A}}(1^\lambda, n, 1^m, T)$. This is the Ideal experiment parameterized by a stateful simulator $\mathcal{S}$. $\mathcal{A}$ receives $\mathsf{crs} \leftarrow \mathcal{S}(1^\lambda, n, 1^m, T)$ from $\mathcal{C}$. $\mathcal{A}$ makes the following queries in an adaptive manner. After this, $\mathcal{A}$ outputs guess b'. Output b'.

- PROGRAM ENCODING: $\mathcal{A}$ sends program P of length m and runtime T, and receives (c_P, i) where $c_P \leftarrow \mathcal{S}(\mathsf{crs})$ and i is an index that after each query of this type is incremented.
- PROVE: $\mathcal{A}$ sends (x, i), and receives (y, π) where $(y, \pi) \leftarrow \mathcal{S}(\mathsf{crs}, x)$ and $y = P(x)$ (for P corresponding to i), otherwise, receives $\perp$.

Figure 6: Real and Ideal experiments for the zero-knowledge property of the succinct delegation of scheme for committed programs.

Zero-knowledge. There exists a poly-time stateful simulator $\mathcal{S}$ such that for any poly-time adversary $\mathcal{A}$, there is a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda, n, m \in \mathbb{N}$ with $n, m \leq 2^\lambda$:

$$\left|\Pr\left[1 \leftarrow \mathsf{Expt}_0^{\mathcal{C},\mathcal{A}}(1^\lambda, 1^n, 1^m)\right] - \Pr\left[1 \leftarrow \mathsf{Expt}_1^{\mathcal{S},\mathcal{A}}(1^\lambda, 1^n, 1^m)\right]\right| \leq \mathsf{negl}(\lambda)$$

where definitions of $\mathsf{Expt}_0^{\mathcal{C},\mathcal{A}}$ and $\mathsf{Expt}_1^{\mathcal{S},\mathcal{A}}$ are provided in Figure 6.

Straight-line Extraction. There exists a PPT extractor $\mathcal{E}$ such that for any stateful poly-size adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for $\lambda \in \mathbb{N}$

$$\Pr\left[\begin{array}{l} P^*(x) \neq y \ \wedge \ \mathsf{Verify}\left(\mathsf{crs}, x, y, c_P, \pi\right) = 1 \end{array} \ : \ \begin{array}{l} (n, m, T) \leftarrow \mathcal{A}(1^\lambda) \\ \mathsf{crs} \leftarrow \mathcal{E}(1^\lambda, n, 1^m, T) \\ (c_P, x, y, \pi) \leftarrow \mathcal{A}(\mathsf{crs}) \\ P^* \leftarrow \mathcal{E}(c_P) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

such that $\mathsf{crs}_1 \leftarrow \mathcal{E}(1^\lambda, n, 1^m, T)$ and $\mathsf{crs}_2 \leftarrow \mathsf{Setup}(1^\lambda, n, 1^m, T)$ are indistinguishable.

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

$$\Pr\left[\begin{array}{l} \mathsf{Verify}(\mathsf{crs}, x, 0, c_P, \pi^{(0)}) = 1 \\ \wedge\ \mathsf{Verify}(\mathsf{crs}, x, 1, c_P, \pi^{(1)}) = 1 \end{array} \ : \ \begin{array}{l} (n, m, T) \leftarrow \mathcal{A}(1^\lambda), \\ \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, n_{\mathsf{pb}}, 1^{n_{\mathsf{pr}}}, S, T) \\ (c_P, x, \pi^{(0)}, \pi^{(1)}) \leftarrow \mathcal{A}(\mathsf{crs}) \end{array}\right] \leq \mathsf{negl}(\lambda) \quad .$$

We note that the aforementioned straight-line extraction property implies the following weaker property given in [GSW23]:

Soundness. For any stateful poly-size adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for $\lambda, n, m \in \mathbb{N}$ with $n = n(\lambda)$ and $m = m(\lambda)$, and every $P \in \{0,1\}^m$,

$$\Pr\left[\begin{array}{l} P(x) \neq y \ \wedge \ \mathsf{Verify}\left(\mathsf{crs}, x, y, c_P, \pi\right) = 1 \\ \end{array} \ : \ \begin{array}{l} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda) \\ (c_P, \mathsf{aux}) \leftarrow \mathsf{ProgEnc}(\mathsf{crs}, P) \\ (x, y, \pi) \leftarrow \mathcal{A}(\mathsf{crs}, c_P, \mathsf{aux}) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

9.2 Construction

We now give a construction for zksDel for program P from $\mathsf{r}\mathcal{E}\mathsf{Del}$ for the RAM machine $\mathcal{M}_P$ defined as below, and associated with a hash tree HT with hash key $\mathsf{ht.hk}$.

RAM Machine $\mathcal{M}_P$

Public Input: $x \in \{0,1\}^n$
Private Input: $P \in \{0,1\}^m$
Output: $y \in \{0,1\}$ such that $y = P(x)$.

Figure 7: RAM Machine for program P

$\mathsf{Setup}(1^\lambda, n, 1^m, T) \to \mathsf{crs}$. Sample $\mathsf{crs} \leftarrow \mathsf{r}\mathcal{E}\mathsf{Del.Setup}(1^\lambda, n, 1^m, T)$ and output it.

$\mathsf{ProgEnc}(\mathsf{crs}, P) \to (c_P, \mathsf{aux})$. Generate $(\mathsf{eh}_P, \pi_P, \mathsf{aux}) \leftarrow \mathsf{r}\mathcal{E}\mathsf{Del.PrivEnc}(\mathsf{crs}, P)$, where $\mathsf{aux} = (P, r)$, and output $c_P = (\mathsf{eh}_P, \pi_P)$ and aux.

$\mathsf{Prove}(\mathsf{crs}, x, \mathsf{aux}) \to (y, \pi)$. Compute $(y, \pi) \leftarrow \mathsf{r}\mathcal{E}\mathsf{Del.Prove}(\mathsf{crs}, x, \mathsf{aux})$ and output (y, π).

$\mathsf{Verify}(\mathsf{crs}, x, y, c_P, \pi) \to \{0,1\}$. Parse c_P as (eh_P, π_P), compute the hash $h \leftarrow \mathsf{HT.Hash}(\mathsf{ht.hk}, x)$ and output

$$\mathsf{r}\mathcal{E}\mathsf{Del.VerifyEnc}(\mathsf{crs}, \mathsf{eh}_P, \pi_P) \ \wedge \ \mathsf{r}\mathcal{E}\mathsf{Del.Verify}(\mathsf{crs}, h, \mathsf{eh}_P, y, \pi) \ .$$

Theorem 9.2. If $\mathsf{r}\mathcal{E}\mathsf{Del}$ for $\mathcal{M}_P$ is secure resuable encrypted RAM delegation scheme then Construction 9.2 is a secure publicly-verifiable zero-knowledge succinct delegation of committed programs scheme for the program P.

Proof. We show that Construction 9.2 satisfies all desired properties.

Completeness. This follows directly from completeness of $\mathsf{r}\mathcal{E}\mathsf{Del}$ for $\mathcal{M}_P$.

Efficiency. This follows directly from the efficiency of $\mathsf{r}\mathcal{E}\mathsf{Del}$ for $\mathcal{M}_P$.

Zero-knowledge. It follows by construction that if $\mathsf{r}\mathcal{E}\mathsf{Del}$ is input-hiding, then Construction 9.2 is zero-knowledge. □

Straight-line Extraction. It follows by construction that if $\mathsf{r}\mathcal{E}\mathsf{Del}$ is an straight-line extractable, then Construction 9.2 is also straight-line extractable.

Strong Soundness. It follows by construction that if $\mathsf{r}\mathcal{E}\mathsf{Del}$ is srongly sound, then Construction 9.2 is also strongly sound.

References

[AB09] Shweta Agrawal and Dan Boneh. Homomorphic macs: Mac-based integrity for network coding. In *Applied Cryptography and Network Security: 7th International Conference, ACNS 2009, Paris-Rocquencourt, France, June 2-5, 2009. Proceedings 7*, pages 292–305. Springer, 2009.

[ABC+07] Giuseppe Ateniese, Randal C. Burns, Reza Curtmola, Joseph Herring, Lea Kissner, Zachary N. J. Peterson, and Dawn Song. Provable data possession at untrusted stores. In Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson, editors, *ACM CCS 2007: 14th Conference on Computer and Communications Security*, pages 598–609, Alexandria, Virginia, USA, October 28–31, 2007. ACM Press.

[ACG26] Abtin Afshar, Jiaqi Cheng, and Rishab Goyal. Multi-hop multi-key homomorphic signatures with context hiding from standard assumptions. Cryptology ePrint Archive, Paper 2024/931, To appear in PKC 2026, 2026.

[ACJ17] Prabhanjan Ananth, Aloni Cohen, and Abhishek Jain. Cryptography with updates. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017, Part II*, volume 10211 of *Lecture Notes in Computer Science*, pages 445–472, Paris, France, April 30 – May 4, 2017. Springer, Cham, Switzerland.

[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.

[AG26] Abtin Afshar and Rishab Goyal. Fully-succinct multi-key fhe & rate-1 simulatable threshold decryption from lwe. To appear at Crypto 2026, 2026.

[AGM18] Shashank Agrawal, Chaya Ganesh, and Payman Mohassel. Non-interactive zero-knowledge proofs for composite statements. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part III*, volume 10993 of *Lecture Notes in Computer Science*, pages 643–673, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Cham, Switzerland.

[AGP14] Prabhanjan Ananth, Vipul Goyal, and Omkant Pandey. Interactive proofs under continual memory leakage. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014, Part II*, volume 8617 of *Lecture Notes in Computer Science*, pages 164–182, Santa Barbara, CA, USA, August 17–21, 2014. Springer Berlin Heidelberg, Germany.

[AJJM21] Prabhanjan Ananth, Abhishek Jain, Zhengzhong Jin, and Giulio Malavolta. Unbounded multi-party computation from learning with errors. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 754–781. Springer, 2021.

[AJL+19] Prabhanjan Ananth, Aayush Jain, Huijia Lin, Christian Matt, and Amit Sahai. Indistinguishability obfuscation without multilinear maps: New paradigms via low degree weak pseudorandomness and security amplification. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 284–332, August 2019.

[AJLA+12] Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. Multiparty computation with low communication, computation and interaction via threshold fhe. In *Advances in Cryptology–EUROCRYPT 2012: 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings 31*, pages 483–501. Springer, 2012.

[AN11] Tolga Acar and Lan Nguyen. Homomorphic proofs and applications, 2011.

[BDS25] Pedro Branco, Nico Döttling, and Akshayaram Srinivasan. Rate-1 statistical noninteractive zero-knowledge. In *Annual International Cryptology Conference*, pages 514–547. Springer, 2025.

[BDSMP91] Manuel Blum, Alfredo De Santis, Silvio Micali, and Giuseppe Persiano. Noninteractive zero-knowledge. *SIAM Journal on Computing*, 20(6):1084–1118, 1991.

[BF11] Dan Boneh and David Mandell Freeman. Homomorphic signatures for polynomial functions. In *Advances in Cryptology–EUROCRYPT 2011: 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings 30*, pages 149–168. Springer, 2011.

[BFKW09] Dan Boneh, David Freeman, Jonathan Katz, and Brent Waters. Signing a linear subspace: Signature schemes for network coding. In *Public Key Cryptography–PKC 2009: 12th International Conference on Practice and Theory in Public Key Cryptography, Irvine, CA, USA, March 18-20, 2009. Proceedings 12*, pages 68–87. Springer, 2009.

[BFM88] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, page 103–112, New York, NY, USA, 1988. Association for Computing Machinery.

[BGI15] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015, Part II*, volume 9057 of *Lecture Notes in Computer Science*, pages 337–367, Sofia, Bulgaria, April 26–30, 2015. Springer Berlin Heidelberg, Germany.

[BGI16] Elette Boyle, Niv Gilboa, and Yuval Ishai. Breaking the circuit size barrier for secure computation under DDH. In Matthew Robshaw and Jonathan Katz, editors, *Advances*

in Cryptology – CRYPTO 2016, Part I, volume 9814 of *Lecture Notes in Computer Science*, pages 509–539, Santa Barbara, CA, USA, August 14–18, 2016. Springer Berlin Heidelberg, Germany.

[BGLS03] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In *Advances in Cryptology—EUROCRYPT 2003: International Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4–8, 2003 Proceedings 22*, pages 416–432. Springer, 2003.

[BHK17] Zvika Brakerski, Justin Holmgren, and Yael Kalai. Non-interactive delegation and batch np verification from standard computational assumptions. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 474–482, 2017.

[BJKL21] Fabrice Benhamouda, Aayush Jain, Ilan Komargodski, and Huijia Lin. Multiparty reusable non-interactive secure computation from LWE. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 724–753. Springer, 2021.

[BKP+24] Nir Bitansky, Chethan Kamath, Omer Paneth, Ron D. Rothblum, and Prashant Nalini Vasudevan. Batch proofs are statistically hiding. In Bojan Mohar, Igor Shinkar, and Ryan O'Donnell, editors, *56th Annual ACM Symposium on Theory of Computing*, pages 435–443, Vancouver, BC, Canada, June 24–28, 2024. ACM Press.

[BL20] Fabrice Benhamouda and Huijia Lin. Mr NISC: Multiparty reusable non-interactive secure computation. In Rafael Pass and Krzysztof Pietrzak, editors, *Theory of Cryptography*, pages 349–378, Cham, 2020. Springer International Publishing.

[BSW11] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In Yuval Ishai, editor, *TCC 2011*, volume 6597 of *LNCS*, pages 253–273, March 2011.

[BWW24] Eli Bradley, Brent Waters, and David J. Wu. Batch arguments to NIZKs from one-way functions. In Elette Boyle and Mohammad Mahmoody, editors, *TCC 2024: 22nd Theory of Cryptography Conference, Part II*, volume 15365 of *Lecture Notes in Computer Science*, pages 431–463, Milan, Italy, December 2–6, 2024. Springer, Cham, Switzerland.

[CG25] Jiaqi Cheng and Rishab Goyal. Boosting SNARKs and rate-1 barrier in arguments of knowledge. In Keren Censor-Hillel, Fabrizio Grandoni, Joël Ouaknine, and Gabriele Puppis, editors, *ICALP 2025: 52nd International Colloquium on Automata, Languages and Programming*, volume 334 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 56:1–56:20, Aarhus, Denmark, July 8–11, 2025. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik.

[CHL+25] Suvradip Chakraborty, Dennis Hofheinz, Roman Langrehr, Jesper Buus Nielsen, Christoph Striecks, and Daniele Venturi. Malleable SNARKs and their applications.

In Serge Fehr and Pierre-Alain Fouque, editors, *Advances in Cryptology – EUROCRYPT 2025, Part IV*, volume 15604 of *Lecture Notes in Computer Science*, pages 184–213, Madrid, Spain, May 4–8, 2025. Springer, Cham, Switzerland.

[CJJ21] Arka Rai Choudhuri, Abhishek Jain, and Zhengzhong Jin. Non-interactive batch arguments for np from standard assumptions. In *Annual International Cryptology Conference*, pages 394–423. Springer, 2021.

[CJJ22] Arka Rai Choudhuri, Abhishek Jain, and Zhengzhong Jin. SNARGs for $\mathcal{P}$ from LWE. In *62nd Annual Symposium on Foundations of Computer Science*, pages 68–79, Denver, CO, USA, February 7–10, 2022. IEEE Computer Society Press.

[CKLM12] Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Sarah Meiklejohn. Malleable proof systems and applications. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 281–300, Cambridge, UK, April 15–19, 2012. Springer Berlin Heidelberg, Germany.

[CKLM13a] Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Sarah Meiklejohn. Malleable signatures: Complex unary transformations and delegatable anonymous credentials. Cryptology ePrint Archive, Report 2013/179, 2013.

[CKLM13b] Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Sarah Meiklejohn. Succinct malleable NIZKs and an application to compact shuffles. In Amit Sahai, editor, *TCC 2013: 10th Theory of Cryptography Conference*, volume 7785 of *Lecture Notes in Computer Science*, pages 100–119, Tokyo, Japan, March 3–6, 2013. Springer Berlin Heidelberg, Germany.

[CM15] Michael Clear and Ciaran McGoldrick. Multi-identity and multi-key leveled fhe from learning with errors. In *Advances in Cryptology–CRYPTO 2015: 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II 35*, pages 630–656. Springer, 2015.

[CW23] Jeffrey Champion and David J. Wu. Non-interactive zero-knowledge from non-interactive batch arguments. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023, Part II*, volume 14082 of *Lecture Notes in Computer Science*, pages 38–71, Santa Barbara, CA, USA, August 20–24, 2023. Springer, Cham, Switzerland.

[DCB25] Trisha Datta, Binyi Chen, and Dan Boneh. VerITAS: Verifying image transformations at scale. In Marina Blanton, William Enck, and Cristina Nita-Rotaru, editors, *2025 IEEE Symposium on Security and Privacy*, pages 4606–4623, San Francisco, CA, USA, May 12–15, 2025. IEEE Computer Society Press.

[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.

[DGMR22] Nico Döttling, Vipul Goyal, Giulio Malavolta, and Justin Raizes. Interaction-preserving compilers for secure computation. In Mark Braverman, editor, *ITCS 2022: 13th Innovations in Theoretical Computer Science Conference*, volume 215, pages 57:1–57:18, Berkeley, CA, USA, January 31 – February 3, 2022. Leibniz International Proceedings in Informatics (LIPIcs).

[DHRW16] Yevgeniy Dodis, Shai Halevi, Ron D. Rothblum, and Daniel Wichs. Spooky encryption and its applications. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part III*, volume 9816 of *Lecture Notes in Computer Science*, pages 93–122, Santa Barbara, CA, USA, August 14–18, 2016. Springer Berlin Heidelberg, Germany.

[FMNP16] Dario Fiore, Aikaterini Mitrokotsa, Luca Nizzardo, and Elena Pagnin. Multi-key homomorphic authenticators. In *International conference on the theory and application of cryptology and information security*, pages 499–530. Springer, 2016.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 169–178, 2009.

[GGI+15] Craig Gentry, Jens Groth, Yuval Ishai, Chris Peikert, Amit Sahai, and Adam Smith. Using fully homomorphic hybrid encryption to minimize non-interative zero-knowledge proofs. *Journal of Cryptology*, 28(4):820–843, 2015.

[GMR85] S Goldwasser, S Micali, and C Rackoff. The knowledge complexity of interactive proof-systems. In *Proceedings of the seventeenth annual ACM symposium on Theory of computing*, pages 291–304, 1985.

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, STOC '87, page 218–229, New York, NY, USA, 1987. Association for Computing Machinery.

[GOS06] Jens Groth, Rafail Ostrovsky, and Amit Sahai. Non-interactive zaps and new techniques for nizk. In *Advances in Cryptology-CRYPTO 2006: 26th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2006. Proceedings 26*, pages 97–111. Springer, 2006.

[GSW23] Riddhi Ghosal, Amit Sahai, and Brent Waters. Non-interactive publicly-verifiable delegation of committed programs. In Alexandra Boldyreva and Vladimir Kolesnikov, editors, *PKC 2023, Part II*, volume 13941 of *LNCS*, pages 575–605, May 2023.

[GVW15a] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Predicate encryption for circuits from LWE. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 503–523, August 2015.

[GVW15b] Sergey Gorbunov, Vinod Vaikuntanathan, and Daniel Wichs. Leveled fully homomorphic signatures from standard lattices. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th Annual ACM Symposium on Theory of Computing*, pages 469–477, Portland, OR, USA, June 14–17, 2015. ACM Press.

[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.

[HW15] Pavel Hubacek and Daniel Wichs. On the communication complexity of secure function evaluation with long output. In Tim Roughgarden, editor, *ITCS 2015*, pages 163–172. ACM, January 2015.

[JMSW02] Robert Johnson, David Molnar, Dawn Song, and David Wagner. Homomorphic signature schemes. In *Cryptographers' track at the RSA conference*, pages 244–262. Springer, 2002.

[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.

[KR09] Yael Tauman Kalai and Ran Raz. Probabilistically checkable arguments. In *Annual International Cryptology Conference*, pages 143–159. Springer, 2009.

[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.

[LATV12] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pages 1219–1234, 2012.

[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.

[Mic94] Silvio Micali. A secure and efficient digital signature algorithm. Technical Memo MIT/LCS/TM-501b, Massachusetts Institute of Technology, Laboratory for Computer Science, April 1994.

[MW16] Pratyay Mukherjee and Daniel Wichs. Two round multiparty computation via multikey fhe. In *Advances in Cryptology–EUROCRYPT 2016: 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II 35*, pages 735–763. Springer, 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, 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.

[PR17] Omer Paneth and Guy N Rothblum. On zero-testable homomorphic encryption and publicly verifiable non-interactive arguments. In *Theory of Cryptography: 15th International Conference, TCC 2017, Baltimore, MD, USA, November 12-15, 2017, Proceedings, Part II 15*, pages 283–315. Springer, 2017.

[RAD+78] Ronald L Rivest, Len Adleman, Michael L Dertouzos, et al. On data banks and privacy homomorphisms. *Foundations of secure computation*, 4(11):169–180, 1978.

[Shi22] Sina Shiehian. mrnisc from lwe with polynomial modulus. In *International Conference on Security and Cryptography for Networks*, pages 481–493. Springer, 2022.

[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.

[Yao82] Andrew C Yao. Protocols for secure computations. In *23rd annual symposium on foundations of computer science (sfcs 1982)*, pages 160–164. IEEE, 1982.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *27th annual symposium on foundations of computer science (Sfcs 1986)*, pages 162–167. IEEE, 1986.

A Additional Definitions

In this section we provide additional pre-requisite definitions used in this work.

A.1 Somewhere Extractable Hash Families

In what follows we recall the definition of a somewhere extractable (SEH) hash family based on prior works [HW15, OPWW15].

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

$\mathsf{Gen}(1^\lambda, N, I) \to (\mathsf{hk}, \mathsf{td})$**.** This is a probabilistic setup algorithm that takes as input a security parameter 1^λ in unary, a message length N, and a subset $I \subseteq [N]$. It outputs a hash key hk.

$\mathsf{Hash}(\mathsf{hk}, x) \to \mathsf{v}$**.** This is a deterministic algorithm that takes as input a hash key hk generated by $\mathsf{Gen}(1^\lambda, N, I)$ and an input $x \in \{0,1\}^N$, and outputs a hash value $\mathsf{v} \in \{0,1\}^{|I| \cdot \mathsf{poly}(\lambda)}$.

$\mathsf{Open}(\mathsf{hk}, x, S) \to ((b_j)_{j \in S}, \rho)$**.** This is a deterministic algorithm that takes as input a hash key hk generated by $\mathsf{Gen}(1^\lambda, N, I)$, an input $x \in \{0,1\}^N$ and set of indices $S \subseteq [N]$, outputs bits b_j for $j \in S$ and an opening $\rho \in \{0,1\}^{\leq |I| \cdot |S| \cdot \mathsf{poly}(\lambda, \log N)}$.

$\mathsf{Verify}(\mathsf{hk}, \mathsf{v}, S, (b_j)_{j \in S}, \rho) \to 0/1$**.** This is a deterministic algorithm that takes as input a hash key hk, a hash value $\mathsf{v} \in \{0,1\}^{|I| \cdot \mathsf{poly}(\lambda, \log N)}$, set of indices $S \subseteq [N]$, a sequence of bits b_j for $j \in S$ and an opening $\rho, |\rho| \leq |I| \cdot |S| \cdot \mathsf{poly}(\lambda, \log N)$, and outputs 0 or 1.

$\mathsf{Extract}(\mathsf{td}, \mathsf{v}) \to (b^*_j)_{j \in I}$**.** This is a deterministic extraction algorithm that takes as input the trapdoor td generated by $\mathsf{Gen}(1^\lambda, N, I)$, a hash value $\mathsf{v} \in \{0,1\}^{|I| \cdot \mathsf{poly}(\lambda, \log N)}$, and outputs extracted sequence of bits $(b_j)_{j \in I}$.

Definition A.1 (SEH)**.** A somewhere extractable hash family $\mathsf{SEH} = (\mathsf{Gen}, \mathsf{Hash}, \mathsf{Open}, \mathsf{Verify}, \mathsf{Extract})$ is required to satisfy the following properties:

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

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 $\lambda \in \mathbb{N}$,

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

Opening Completeness. For any $\lambda \in \mathbb{N}$, any $N \leq 2^\lambda$, any subset $I \subseteq [N]$, any index $j \in [N]$, and any $x \in \{0,1\}^N$,

$$\Pr\left[\begin{array}{l} \forall\ j \in S, b_j = x_j \\ \wedge\ \mathsf{Verify}(\mathsf{hk}, \mathsf{v}, S, (b_j)_{j \in S}, \rho) = 1 \end{array} : \begin{array}{l} (\mathsf{hk}, \mathsf{td}) \leftarrow \mathsf{Gen}(1^\lambda, N, I), \\ \mathsf{v} = \mathsf{Hash}(\mathsf{hk}, x), \\ ((b_j)_{j \in S}, \rho) = \mathsf{Open}(\mathsf{hk}, x, S), \end{array}\right] = 1.$$

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

$$\Pr\left[\begin{array}{ll} \mathsf{Verify}(\mathsf{hk}, \mathsf{v}, I, (b_j)_{j\in I}, \rho) = 1 & (\mathsf{hk}, \mathsf{td}) \leftarrow \mathsf{Gen}(1^\lambda, N, I), \\ \wedge\ b_j \neq b_j^*, \text{ for any } j \in I & : (\mathsf{v}, (b_j)_{j\in I}, \rho) \leftarrow \mathcal{A}(\mathsf{hk}), \\ & (b_j^*)_{j\in I} = \mathsf{Extract}(\mathsf{td}, \mathsf{v}) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

Remark A.2 ([KLVW23, HW15, OPWW15])**.** Assuming $X \in \{\mathsf{QR}, \mathsf{DCR}, \mathsf{DDH}, \mathsf{LWE}, O(1)\text{-}\mathsf{LIN}$, there exists an SEH family.

A.2 Multi-Key Fully Homomorphic Encryption

We provide the and definitions for a multi-key fully homomorphic encryption (mkFHE) for message space $\mathcal{M} \in \{0,1\}^*$ used in our mrNISC constructions.

Syntax. A multi-key fully homomorphic encryption scheme (mkFHE) consists of the following polynomial-time algorithms.

$\mathsf{Setup}(1^\lambda) \to \mathsf{PP}$**.** The probabilistic setup algorithm takes as input the security parameter λ and outputs the public parameters PP. The following algorithms take PP as an implicit parameters unless mentioned.

$\mathsf{KeyGen}(\mathsf{PP}) \to (\mathsf{pk}, \mathsf{sk})$**.** The probabilistic key generation algorithm takes as input the public parameters PP 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, (multi-bit) message $m \in \{0,1\}^*$, and outputs the ciphertext ct.

$\mathsf{Eval}(f, \{\mathsf{pk}_{\mathsf{id}}, \mathsf{ct}_{\mathsf{id}}\}_{\mathsf{id}\in[n]}) \to \widetilde{\mathsf{ct}}$**.** This possibly randomized algorithm takes as input the description of the function f, the public keys and ciphertexts for n parties $\{\mathsf{pk}_{\mathsf{id}}, \mathsf{ct}_{\mathsf{id}}\}_{\mathsf{id}\in[n]}$, and outputs the evaluated ciphertext $\widetilde{\mathsf{ct}}$.

$\mathsf{Dec}(\mathsf{sk}_1, \ldots, \mathsf{sk}_n, \widetilde{\mathsf{ct}}) \to y$**.** The deterministic decryption algorithm takes as input secret keys for n parties $\{\mathsf{sk}_{\mathsf{id}}\}_{\mathsf{id}\in[n]}$, an evaluated ciphertext ct, and outputs the value y.

Definition A.3 (mkFHE)**.** An mkFHE scheme $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ for message space $\{0,1\}^*$, is required to satisfy the following properties:

Correctness. For any $\lambda \in \mathbb{N}$, $n = n(\lambda)$, $x, x_1, \ldots, x_n \in \{0,1\}^*$, and n-ary $\mathsf{P/Poly}$ circuit f:

$$\Pr\left[x = \mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) \ : \begin{array}{l} \mathsf{PP} \leftarrow \mathsf{Setup}(1^\lambda), \\ (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{PP}), \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, x) \end{array}\right] = 1$$

$$\Pr\left[\begin{array}{ll} f(x_1, \ldots, x_n) = & \mathsf{PP} \leftarrow \mathsf{Setup}(1^\lambda), \\ \mathsf{Dec}(\mathsf{sk}_1, \ldots, \mathsf{sk}_n, \widetilde{\mathsf{ct}}) & : \forall\ \mathsf{id} \in [n], (\mathsf{pk}_{\mathsf{id}}, \mathsf{sk}_{\mathsf{id}}) \leftarrow \mathsf{KeyGen}(\mathsf{PP}), \\ & \forall\ \mathsf{id} \in [n], \mathsf{ct}_{\mathsf{id}} \leftarrow \mathsf{Enc}(\mathsf{pk}_{\mathsf{id}}, x_{\mathsf{id}}), \\ & \widetilde{\mathsf{ct}} = \mathsf{Eval}(f, \{\mathsf{pk}_{\mathsf{id}}, \mathsf{ct}_{\mathsf{id}}\}_{\mathsf{id}\in[n]}) \end{array}\right] = 1$$

Compactness. For any $\lambda \in N$, $n = n(\lambda)$, $x_1, \ldots, x_n \in \{0,1\}^*$, and n-ary $\mathsf{P/Poly}$ circuit f, there exists a fixed polynomial $\mathsf{poly}(\cdot,\cdot,\cdot)$ such that $|\widetilde{\mathsf{ct}}| = \mathsf{poly}(\lambda, n, |f(x_1, \ldots, x_n)|)$ where $\mathsf{PP} \leftarrow \mathsf{Setup}(1^\lambda), \forall\, \mathsf{id} \in [n], (\mathsf{pk_{id}}, \mathsf{sk_{id}}) \leftarrow \mathsf{KeyGen}(\mathsf{PP}), \mathsf{ct_{id}} \leftarrow \mathsf{Enc}(\mathsf{pk_{id}}, x_{\mathsf{id}}), \widetilde{\mathsf{ct}} = \mathsf{Eval}(f, \{\mathsf{pk_{id}}, \mathsf{ct_{id}}\}_{\mathsf{id}\in[n]})$.

Semantic Security. For any stateful PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that $\forall\ \lambda \in \mathbb{N}$,

$$\left| \Pr \left[b \leftarrow \mathcal{A}(\mathsf{ct}_b) \quad : \quad \begin{array}{l} b \leftarrow \{0,1\}, \mathsf{PP} \leftarrow \mathsf{Setup}(1^\lambda), \\ (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{PP}), \\ (m_0, m_1) \leftarrow \mathcal{A}(1^\lambda, \mathsf{PP}, \mathsf{pk}), \\ \mathsf{ct}_b \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b) \end{array} \right] - \frac{1}{2} \right| \leq \mathsf{negl}(\lambda)$$

Theorem A.4 ([LATV12, CM15, MW16])**.** Assuming the hardness of LWE with sub-exponential modulus-to-noise ratio and circular security, there exists an mkFHE scheme for all $\mathsf{P/Poly}$ circuits and $\mathcal{M} = \{0,1\}^*$.

B Split Memory RAM Delegation

In this section, we state and provide a short proof of a property of RAM Delegation that [CJJ22, KLVW23] satisfies implicitly. Note that in this section we use 'memory' and 'configuration' interchangeably and we let both of them include the input as well the work tape. We show that the RAM delegation of [CJJ22, KLVW23] can be done when the memory is hashed into multiple hash values and can be verified using these multiple hash values. One advantage of this approach as we demonstrate is that the running time of the prover of RAM delegation scheme can be dependent only on the portion of the memory it uses as work tape (which of polynomial-size computations, is polynomial).

We require an additional property from the collision resistant hash family as building block: for $\mathsf{HT.Hash}$ $(\mathsf{ht.hk}, 0^*)$, the output value $\mathsf{h} = 0^{\lambda+1}$. This can be achieved using any collision resistant hash tree HT. Using HT we can define a "punctured" hashing algorithm as follows:

$\mathsf{Hash}(\mathsf{ht.hk}, x) \to \mathsf{h}$**.** If x is an all-zero string, output $\mathsf{h} = 0^{\lambda+1}$. Otherwise, output $\mathsf{h} = 1 \,\|\, \mathsf{HT.Hash}(\mathsf{ht.hk}, x)$.

As remarked in [CG25], completeness and collision resistance follow from HT. One useful implication of this "punctured" hash evaluation is that the RAM machine need not worry about the part of memory which is not needed. Hence, when we start with an all zero configuration, we need not provide the hash values for all the configurations as they are a string of zeros anyway. Thus, the running time of RAM machine only grows polynomially in the size of the work tape it uses depending on the number partitions specified and hash values changed. We provide a RAM delegation construction that obeys Definition 3.17 which splits the memory into n parts below.

B.1 Split Memory RAM Delegation Construction

We provide the construction of split memory RAM delegation in accordance with Definition 3.17 as follows. We remark that $\widetilde{\mathcal{L}}$ (specified below) is the language for Batch Index Circuit-SAT and $C^{\mathcal{M}}$ doesn't require the entire description of $\mathcal{M}$ hardwired. As shown by Choudhuri et al. [CJJ22], a state transition circuit which ensures that transition between st_{i-1} and st_i are consistent

with $\mathcal{M}$'s description is sufficient. For the sake of brevity, we ignore this technical subtlety in the construction. In addition, since we are using batch arguments for index languages and the statements are clear from the context, we avoid repeating the index statements as input to the batch argument's algorithms. However, we will add the language circuit to the inputs of the batch argument's algorithms as it will help with the analysis.

Construction B.1 (Split Memory RAM Delegation)**.** Let $\mathsf{SEH} = (\mathsf{Gen}, \mathsf{Hash}, \mathsf{Open}, \mathsf{Verify}, \mathsf{Extract})$ be a somewhere extractable hash family (Definition A.1), $\mathsf{HT} = (\mathsf{Gen}, \mathsf{Hash}, \mathsf{Open}, \mathsf{Write}, \mathsf{VerifyRead}, \mathsf{VerifyWrite})$ be a hash tree, $\mathsf{seBARG} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ be a (publicly-verifiable, non-interactive) batch argument scheme for the language $\widetilde{\mathcal{L}} = \{C^{\mathcal{M}} : \exists\ w_i \text{ such that } C^{\mathcal{M}}(i, w_i) = 1,\ \forall\ i \in [k]\}$ where $C^{\mathcal{M}}$ is defined in Figure 8. We provide the construction of a RAM delegation scheme for machine $\mathcal{M}$ with strong soundness below.

Description of circuit $C^{\mathcal{M}}$

Hardwired: $T, S, \mathsf{seh.hk}^0, \mathsf{seh.hk}^1, \mathsf{ht.hk}, \mathsf{h}_0^*, \mathsf{v}^0, \mathsf{v}^1, \mathsf{out}$
Instance: i
Witness: $w_i = (\widehat{w}_{i-1}, \rho_{i-1}^0, \rho_{i-1}^1, \widehat{w}_i, \rho_i^0, \rho_i^1, b_{\mathsf{rd,i}}, \ell_{\mathsf{rd,i}}, j_i, u_{\mathsf{rd,i}}, \mathsf{op}_{\mathsf{rd},i}, b_{\mathsf{wr,i}}, \ell_{\mathsf{wr,i}}, j_i', u_{\mathsf{wr,i}}, \mathsf{op}_{\mathsf{wr},i})$
Output: Output 1 if and only if all of these checks pass.

1. Parse $\widehat{w}_{i-1} = (\mathsf{st}_{i-1}, \mathsf{h}_{i-1})$, and $\widehat{w}_i = (\mathsf{st}_i, \mathsf{h}_i)$.
2. If $i = 1$, check that st_0 is the initial state. Also, check $\mathsf{h}_0 = \mathsf{h}_0^*$.
3. If $i = T$, check that st_T is accepting/rejecting state based on $\mathsf{out} = 1/0$ respectively.
4. $\mathsf{SEH.Verify}\left(\mathsf{seh.hk}^0, \mathsf{v}^0, \mathbb{I}_{i-1}, \widehat{w}_{i-1}, \rho_{i-1}^0\right) \stackrel{?}{=} 1$ and $\mathsf{SEH.Verify}\left(\mathsf{seh.hk}^1, \mathsf{v}^1, \mathbb{I}_{i-1}, \widehat{w}_{i-1}, \rho_{i-1}^1\right) \stackrel{?}{=} 1$.
5. $\mathsf{SEH.Verify}\left(\mathsf{seh.hk}^0, \mathsf{v}^0, \mathbb{I}_i, \widehat{w}_i, \rho_i^0\right) \stackrel{?}{=} 1$ and $\mathsf{SEH.Verify}\left(\mathsf{seh.hk}^1, \mathsf{v}^1, \mathbb{I}_i, \widehat{w}_i, \rho_i^1\right) \stackrel{?}{=} 1$.
6. $\mathcal{M}$ in st_{i-1} should read from $\ell_{\mathsf{rd,i}}$ and write $b_{\mathsf{wr,i}}$ to $\ell_{\mathsf{wr,i}}$ and transitions to st_i. In addition, $j_i, j_i' \in [n]$ are partitions that accommodate indices $\ell_{\mathsf{rd,i}}$ and $\ell_{\mathsf{wr,i}}$ respectively.
7. $u_{\mathsf{rd,i}} \stackrel{?}{=} \ell_{\mathsf{rd,i}} - \left(S^{(1)} + \ldots + S^{(j_i - 1)}\right)$ and $u_{\mathsf{wr,i}} \stackrel{?}{=} \ell_{\mathsf{wr,i}} - \left(S^{(1)} + \ldots + S^{(j_i' - 1)}\right)$.
8. $\mathsf{HT.VerifyRead}(\mathsf{ht.hk}, \mathsf{h}_{i-1}^{(j_i)}, u_{\mathsf{rd,i}}, b_{\mathsf{rd,i}}, \mathsf{op}_{\mathsf{rd},i}) \stackrel{?}{=} 1$.
9. $\mathsf{HT.VerifyWrite}(\mathsf{ht.hk}, \mathsf{h}_{i-1}^{(j_i')}, u_{\mathsf{wr,i}}, b_{\mathsf{wr,i}}, \mathsf{h}_i^{(j_i')}, \mathsf{op}_{\mathsf{wr},i}) \stackrel{?}{=} 1$.

Figure 8: Circuit for verifying RAM memory transition.

$\mathsf{Setup}(1^\lambda, T, S) \to \mathsf{crs}$**.** The probabilistic setup algorithm takes as input the security parameter, a vector of split sizes $S = \left(S^{(1)}, \ldots, S^{(n)}\right)$, running time T for machine $\mathcal{M}$, and does the following.

1. Sample two SEH hash keys, $\mathsf{seh.hk}^0, \mathsf{seh.hk}^1 \leftarrow \mathsf{SEH.Gen}(1^\lambda, m \cdot (T+1), \mathbb{I}_1)$ where $m = |\widehat{w}_i|$ (later specified in Prove) and $\mathbb{I}_i = \{(i-1) \cdot m + 1, \ldots, i \cdot m\}$ for $i \in [T]$.
2. Sample crs for seBARG, $\mathsf{barg.crs} \leftarrow \mathsf{seBARG.Setup}\left(1^\lambda, T, 1^{|C^{\mathcal{M}}|}, 1\right)$.
3. Sample a HT hash key, $\mathsf{ht.hk} \leftarrow \mathsf{HT.Gen}(1^\lambda)$.

Output $\mathsf{crs} = \left(T, S, \mathsf{seh.hk}^0, \mathsf{seh.hk}^1, \mathsf{barg.crs}, \mathsf{ht.hk}\right)$.

$\mathsf{Digest}(\mathsf{crs}, x) \to \mathsf{h}$. The deterministic digest algorithm parses crs, uses $\mathsf{ht.hk}$, parses x as $\left(x^{(1)}, \ldots, x^{(n)}\right)$, such that $\left|x^{(j)}\right| = S^{(j)}$ for each $j \in [n]$, computes $\mathsf{h}^{(j)} = \mathsf{HT.Hash}\left(\mathsf{ht.hk}, x^{(j)}\right)$, and outputs $\mathsf{h} = \left(\mathsf{h}^{(1)}, \ldots, \mathsf{h}^{(n)}\right)$.

$\mathsf{Prove}(\mathsf{crs}, \mathsf{cf}_0) \to (\mathsf{out}, \pi)$. The probabilistic prover algorithm parses crs as $(T, S, \mathsf{seh.hk}^0, \mathsf{seh.hk}^1, \mathsf{barg.crs}, \mathsf{ht.hk})$ and does the following.

1. Let out denote the output, $((\mathsf{st}_1, \mathsf{cf}_1), \ldots, (\mathsf{st}_T, \mathsf{cf}_T))$ denote the states and configurations resultant from running $\mathcal{M}(\mathsf{cf}_0, T)$, where cf_i for $i \in [0, T]$ includes the input and work tape.
2. For each $i \in [0, T]$, let $\mathsf{h}_i = \mathsf{Digest}(\mathsf{crs}, \mathsf{cf}_i)$.
3. For each $i \in [T]$:
   - Let $\ell_{\mathsf{rd,i}}, \ell_{\mathsf{wr,i}}$ denote the location that $\mathcal{M}$ reads from, writes to memory when it is in state st_{i-1}. Let $\ell_{\mathsf{rd,i}}, \ell_{\mathsf{wr,i}}$ belong to some partitions indexed by $j_i, j'_i \in [n]$ respectively.
   - Let $u_{\mathsf{rd,i}}$ denote the index of $\ell_{\mathsf{rd,i}}$ in the partition j_i. That is $u_{\mathsf{rd,i}} = \ell_{\mathsf{rd,i}} - \left(S^{(1)} + \ldots + S^{(j_i - 1)}\right)$. Similarly, calculate $u_{\mathsf{wr,i}}$.
   - Compute $(b_{\mathsf{rd,i}}, \mathsf{op}_{\mathsf{rd},i}) = \mathsf{HT.Open}\left(\mathsf{ht.hk}, x_{i-1}^{(j_i)}, u_{\mathsf{rd,i}}\right)$ and $\left(\widetilde{\mathsf{h}}_i^{(j'_i)}, \mathsf{op}_{\mathsf{wr},i}\right) = \mathsf{HT.Write}\left(\mathsf{ht.hk}, x_{i-1}^{(j'_i)}, u_{\mathsf{wr,i}}, b_{\mathsf{wr,i}}\right)$ where $x_{i-1}^{(j_i)}$ and $x_{i-1}^{(j'_i)}$ are the j_i-th and j'_i-th partitions of cf_{i-1}.
4. For $i \in [0, T]$, set $\widehat{w}_i := (\mathsf{st}_i, \mathsf{h}_i)$.
5. Let $\widehat{\mathbf{w}} = (\widehat{w}_i)_{i \in [0,T]}$, compute $\mathsf{v}^0 = \mathsf{SEH.Hash}(\mathsf{seh.hk}^0, \widehat{\mathbf{w}})$, $\mathsf{v}^1 = \mathsf{SEH.Hash}(\mathsf{seh.hk}^1, \widehat{\mathbf{w}})$.
6. For $i \in [0, T]$, $(\widehat{w}_i, \rho_i^0) = \mathsf{SEH.Open}(\mathsf{seh.hk}^0, \widehat{\mathbf{w}}, \mathbb{I}_i)$, $(\widehat{w}_i, \rho_i^1) = \mathsf{SEH.Open}(\mathsf{seh.hk}^1, \widehat{\mathbf{w}}, \mathbb{I}_i)$.
7. For $i \in [T]$, let $w_i := (\widehat{w}_{i-1}, \rho_{i-1}^0, \rho_{i-1}^1, \widehat{w}_i, \rho_i^0, \rho_i^1, b_{\mathsf{rd,i}}, \ell_{\mathsf{rd,i}}, j_i, u_{\mathsf{rd,i}}, \mathsf{op}_{\mathsf{rd},i}, b_{\mathsf{wr,i}}, \ell_{\mathsf{wr,i}}, j'_i, u_{\mathsf{wr,i}}, \mathsf{op}_{\mathsf{wr},i})$.
8. Let $\mathsf{barg}.\pi \leftarrow \mathsf{seBARG.Prove}\left(\mathsf{barg.crs}, C^{\mathcal{M}}, (w_i)_{i \in [T]}\right)$.
9. Set $\pi := (\mathsf{v}^0, \mathsf{v}^1, \mathsf{barg}.\pi)$ and output (out, π).

$\mathsf{Verify}(\mathsf{crs}, \mathsf{h}_0^*, \mathsf{out}, \pi) \to 0/1$. The deterministic verification algorithm parses crs as $(T, S, \mathsf{seh.hk}^0, \mathsf{seh.hk}^1, \mathsf{barg.crs}, \mathsf{ht.hk})$, parses π as $(\mathsf{v}^0, \mathsf{v}^1, \mathsf{barg}.\pi)$, and outputs $\mathsf{seBARG.Verify}\left(\mathsf{barg.crs}, C^{\mathcal{M}}, \mathsf{barg}.\pi\right)$.

Theorem B.2. If SEH is a secure somewhere extractable hash family (Definition A.1), HT is a sound hash tree (Definition 3.6), and seBARG is a secure somewhere extractable batch argument scheme for language $\widetilde{\mathcal{L}}$ (Definition 3.1), then Construction B.1 is a RAM delegation scheme for machine $\mathcal{M}$ w.r.to HT that satisfies strong soundness.

Proof. We show that Construction B.1 is correct, efficient, and collision resistant as follows:

Completeness. Completeness of RAM delegation scheme follows from the completeness of HT, seBARG, and opening completeness of SEH.

Efficiency. Assuming that $\mathsf{HT}, \mathsf{SEH}, \mathsf{seBARG}$ are efficient, the RAM delegation scheme satisfies the following properties:

1. Sizes of the crs and proof π are $\mathsf{poly}(\lambda, n, \log T)$.
2. Running time of Setup is $\mathsf{poly}(\lambda, n, \log T)$.
3. Running time of Digest is $\mathsf{poly}(n, \lambda)$ and the size of digest is $\mathsf{poly}(n, \lambda)$.
4. Running time of Prove is $\mathsf{poly}(\lambda, T, n)$.
5. Running time of Verify is $\mathsf{poly}(\lambda, n, \log T)$.

Collision Resistance. It is easy to see that if HT is collision resistant, then so is the RAM delegation scheme. If there exists an adversary that can produce two configurations $\mathsf{cf}, \mathsf{cf}'$ that break collision resistance with non-negligible probability ϵ, then note that there is at least one partition indexed by $j \in [n]$ such that the j-th partition of cf is not equal to the j-th partition of cf'. We can reduce this adversary to an adversary that breaks the collision resistance of HT by using this $x^{(j)}$ with ϵ probability. □

B.2 Security Analysis

Here, we show that Construction B.1 is secure against PPT adversaries. We provide a sketch of the proof and it is easy to see that full proof follows from [CJJ22, KLVW23].

Proof of strong soundness. We show that if HT satisfies reading and writing soundness, SEH satisfies index-hiding, straight-line extraction, and seBARG satisfies index-hiding and somewhere extraction, then Construction B.1 satisfies strong soundness. Assume towards a contradiction that there exists a PPT adversary $\mathcal{A}$ that can break the strong soundness property with some non-negligible probability $\epsilon(\lambda)$ as defined in Definition 3.17 for $T = T(\lambda), n = n(\lambda), S(\lambda) = S = (S^{(1)}, \ldots, S^{(n)})$. In other words,

$$\Pr\left[\begin{array}{l|ll} \forall\, b \in \{0,1\}, \mathsf{seBARG.Verify}(\mathsf{barg.crs}, & : & \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, T, S), \\ C_b, \mathsf{barg}.\pi_b) = 1 & & (\mathsf{h}_0, \pi_0, \pi_1) \leftarrow \mathcal{A}(\mathsf{crs}) \end{array}\right] = \epsilon(\lambda)$$

where C_b is $C^{\mathcal{M}}$ (Figure 8) with $\mathsf{out} = b$. In what follows, we will assume that Setup_k is the same as Setup except that we use $k, \mathbb{I}_k, \mathbb{I}_{k+1}$ for $k \in [T]$ in $\mathsf{seBARG.Setup}$, β-th $\mathsf{SEH.Gen}$, and $(1-\beta)$-th $\mathsf{SEH.Gen}$ respectively where $\beta = (k-1) \mod 2$. By index-hiding property of $\mathsf{SEH}, \mathsf{seBARG}$, there exists a negligible function $\mu(\cdot)$ such that for any PPT adversary $\mathcal{A}$, $\lambda \in \mathbb{N}$ and $k \in [T]$,

$$\Pr\left[\begin{array}{l|ll} \forall\, b \in \{0,1\}, & & \mathsf{crs} \leftarrow \mathsf{Setup}_k(1^\lambda, T, S), \\ \mathsf{seBARG.Verify}(\mathsf{barg.crs}, C_b, & : & (\mathsf{h}_0, \pi_0, \pi_1) \leftarrow \mathcal{A}(\mathsf{crs}) \\ \mathsf{barg}.\pi_b) = 1 & & \end{array}\right] \geq \epsilon(\lambda) - \mu(\lambda)$$

Now, we proceed to inductively argue for k that there exists a negligible function $\zeta(\cdot)$ such that for any PPT adversary $\mathcal{A}$, $k \in [T], \lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{ll} \forall\ b \in \{0,1\}, & \mathsf{crs} \leftarrow \mathsf{Setup}_k(1^\lambda, T, S), \\ \mathsf{seBARG.Verify} & (\mathsf{h}_0, \pi_0, \pi_1) \leftarrow \mathcal{A}(\mathsf{crs}) \\ \ (\mathsf{barg.crs}, C_b, \mathsf{barg}.\pi_b) = 1 \wedge & \forall\ b \in \{0,1\}, \widehat{w}_{b,k} \leftarrow \\ \widehat{w}_{0,k} = \widehat{w}_{1,k} & \mathsf{SEH.Extract}(\mathsf{seh.td}^\beta, v_b^\beta) \end{array}\right] \geq \epsilon(\lambda) - k \cdot \zeta(\lambda) \qquad (20)$$

where $\beta \equiv (k-1) \mod 2$. When $k = T$, Equation 20 claims that an accepting state and rejecting configurations are identical with non-negligible probability which is a contradiction. Hence, we will prove by induction on k that above equation holds.

Base Case. We need to start from $\widehat{w}_{b,0}$, however, since the Setup_k is only defined for $k \geq 1$, for the base case we just consider $k = 1$ for setup and extract $\widehat{w}_{0,0}$ and $\widehat{w}_{1,0}$ from SEH. This holds as there is a unique starting hash value h_0 and starting state st_0.

Induction. Assume that Equation 20 holds for k. Now, we will switch to using $\mathsf{seBARG}.\mathcal{E}$ instead of normal instantiation. By somewhere extraction property of seBARG, there exists a negligible function $\nu(\cdot)$ such that for any PPT adversary $\mathcal{A}$, $k \in [T], \lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{ll} \widehat{w}_{0,k} = \widehat{w}_{1,k} \wedge & \mathsf{crs} \leftarrow \mathsf{Setup}_k(1^\lambda, T, S), \\ \forall\ b \in \{0,1\}, & (\mathsf{h}_0, \pi_0, \pi_1) \leftarrow \mathcal{A}(\mathsf{crs}) \\ \mathsf{seBARG.Verify} & \forall\ b \in \{0,1\}, \\ \ (\mathsf{barg.crs}, C_b, \mathsf{barg}.\pi_b) = 1 & w_{b,k} \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{barg.td}, C_b, \mathsf{barg}.\pi_b), \\ \wedge\ C_b(k, w_{b,k}) = 1 & \widehat{w}_{b,k} \leftarrow \mathsf{SEH.Extract}(\mathsf{seh.td}^\beta, v_b^\beta) \end{array}\right]$$
$$\geq \epsilon(\lambda) - k \cdot \zeta(\lambda) - \nu(\lambda)$$

Parse $w_{b,k}$ to find $\mathsf{st}_{b,k-1}$, $\mathsf{st}_{b,k}$, $\mathsf{h}_{b,k-1}$, $\mathsf{h}_{b,k}$. Next we additionally extract from $(1-\beta)$-th instantiation SEH and by assuming the somewhere statistical binding property of SEH, there exists a negligible function $\nu_1(\cdot)$ such that for any $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{ll} \widehat{w}_{0,k} = \widehat{w}_{1,k} \wedge & \mathsf{crs} \leftarrow \mathsf{Setup}_k(1^\lambda, T, S), \\ \forall\ b \in \{0,1\}, & (\mathsf{h}_0, \pi_0, \pi_1) \leftarrow \mathcal{A}(\mathsf{crs}) \\ \mathsf{seBARG.Verify} & \forall\ b \in \{0,1\}, \\ \ (\mathsf{barg.crs}, C_b, \mathsf{barg}.\pi_b) = 1 & w_{b,k} \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{barg.td}, C_b, \mathsf{barg}.\pi_b), \\ \wedge\ C_b(k, w_{b,k}) = 1 \wedge & \widehat{w}_{b,k} \leftarrow \mathsf{SEH.Extract}(\mathsf{seh.td}^\beta, v_b^\beta), \\ \widehat{w}_{b,k} = (\mathsf{st}_{b,k}, \mathsf{h}_{b,k}) \wedge & \\ \widehat{w}_{b,k+1} = (\mathsf{st}_{b,k+1}, \mathsf{h}_{b,k+1}) & \widehat{w}_{b,k+1} \leftarrow \mathsf{SEH.Extract}(\mathsf{seh.td}^{1-\beta}, v_b^{1-\beta}) \end{array}\right]$$
$$\geq \epsilon(\lambda) - k \cdot \zeta(\lambda) - \nu(\lambda) - \nu_1(\lambda)$$

By relying on the reading soundness and writing soundness of HT, we can see that there exists a negligible function $\nu_2(\cdot)$ such that for any PPT adversary $\mathcal{A}$, $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l} \\ \widehat{w}_{0,k+1} = \widehat{w}_{1,k+1} \wedge \\ \forall\ b \in \{0,1\}, \\ \mathsf{seBARG.Verify} \\ \ (\mathsf{barg.crs}, C_b, \mathsf{barg}.\pi_b) = 1 \\ \\ \end{array} : \begin{array}{l} \mathsf{crs} \leftarrow \mathsf{Setup}_k(1^\lambda, T, S), \\ (\mathsf{h}_0, \pi_0, \pi_1) \leftarrow \mathcal{A}(\mathsf{crs}) \\ \forall\ b \in \{0,1\}, \\ w_{b,k} \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{barg.td}, C_b, \mathsf{barg}.\pi_b), \\ \widehat{w}_{b,k} \leftarrow \mathsf{SEH.Extract}(\mathsf{seh.td}^\beta, v_b^\beta), \\ \widehat{w}_{b,k+1} \leftarrow \mathsf{SEH.Extract}(\mathsf{seh.td}^{1-\beta}, v_b^{1-\beta}) \end{array}\right]$$
$$\geq \epsilon(\lambda) - k \cdot \zeta(\lambda) - \nu(\lambda) - \nu_1(\lambda) - \nu_2(\lambda)$$

Now, we will stop extracting using seBARG and β-th SEH and instead extract them on $k+1$ and $\mathbb{I}_{k+2}$ respectively. By the index hiding properties of seBARG and SEH, there exists a negligible function $\nu_3(\cdot)$ such th for any PPT $\mathcal{A}, \lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l} \widehat{w}_{0,k+1} = \widehat{w}_{1,k+1} \wedge \\ \forall\ b \in \{0,1\}, \\ \mathsf{seBARG.Verify} \\ \ (\mathsf{barg.crs}, C_b, \mathsf{barg}.\pi_b) = 1 \end{array} : \begin{array}{l} \mathsf{crs} \leftarrow \mathsf{Setup}_{k+1}(1^\lambda, T, S), \\ (\mathsf{h}_0, \pi_0, \pi_1) \leftarrow \mathcal{A}(\mathsf{crs}) \\ \forall\ b \in \{0,1\}, \\ \widehat{w}_{b,k+1} \leftarrow \mathsf{SEH.Extract}(\mathsf{seh.td}^{1-\beta}, v_b^{1-\beta}) \end{array}\right]$$
$$\geq \epsilon(\lambda) - k \cdot \zeta(\lambda) - \nu(\lambda) - \nu_1(\lambda) - \nu_2(\lambda) - \nu_3(\lambda)$$

Note that $(1-\beta) \equiv ((k+1)-1) \mod 2$. By setting $\zeta(\lambda) = \nu(\lambda) + \sum_{l=1}^{3} \nu_l(\lambda)$, we prove the induction step as desired. $\square$