Encrypted RAM Delegation: Applications to Rate-1 Extractable Arguments, Homomorphic NIZKs, MPC, and more

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 GMU ayadav5@gmu.edu[†] Saikumar Yadugiri UW-Madison saikumar@cs.wisc.edu

Abstract

In this paper we introduce the notion of encrypted RAM delegation. In an encrypted RAM delegation scheme, the prover creates a succinct proof for a group of two input strings x_{pb} and x_{pr} , where x_{pb} corresponds to a large *public* input and x_{pr} is a *private* input. A verifier can check correctness of computation of \mathcal{M} on (x_{pb}, x_{pr}) , given only the proof π and x_{pb} .

We design encrypted RAM delegation schemes from a variety of standard assumptions such as DDH, or LWE, or k-linear. We prove strong knowledge soundness guarantee for our scheme as well as a special input hiding property to ensure that π does not leak anything about x_{pr} .

We follow this by describing multiple applications of encrypted RAM delegation. First, we show how to design a rate-1 non-interactive zero-knowledge (NIZK) argument system with a straight-line extractor. Despite over 30+ years of research, the only known construction in the literature for rate-1 NIZKs from standard assumptions relied on fully homomorphic encryption. Thus, we provide the first rate-1 NIZK scheme based purely on DDH or k-linear assumptions.

Next, we also design fully-homomorphic NIZKs from encrypted RAM delegation. The only prior solution crucially relied on algebraic properties of pairing-based NIZKs, thus was only known from the decision linear assumption. We provide the first fully-homomorphic NIZK system from LWE (thus post-quantum security) and from DDH-hard groups.

We also provide a communication-complexity-preserving compiler for a wide class of semimalicious multiparty computation (MPC) protocols to obtain fully malicious MPC protocols. This gives the first such compiler for a wide class of MPC protocols as any comparable compiler provided in prior works relied on strong non-falsifiable assumptions such as zero-knowledge succinct non-interactive arguments of knowledge (zkSNARKs). Moreover, we also show many other applications to composable zero-knowledge batch arguments, succinct delegation of committed programs, and fully context-hiding multi-key multi-hop homomorphic signatures.

^{*}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.

Contents

1 Introduction		
	1.1 Overview of our applications	3
	1.2 Overview of our Encrypted RAM construction	9
	1.3 Related Work	11
9	Decliminanies	10
4	2.1 Pate 1 Semewhere Extractable (Zero Knowledge) Pateh Argumenta	12 19
	2.1 Rate-1 Somewhere Extractable (Zero-Knowledge) Datch Arguments	14
	2.2 RAM Flograms	14
	2.4 Non Internative Zero Knowledge Arguments (NIZK)	16
	2.4 Non-Interactive Zero-Knowledge Arguments (NIZIX)	17
	2.6 Public Key Encountion System	18
	2.7 BAM Delegation Scheme	10
	2.7 RAM Delegation Scheme	19
3	Defining (Reusable) Encrypted RAM Delegation	20
	3.1 Encrypted RAM Delegation	20
	3.2 Reusable Encrypted RAM Delegation	22
4	Constructing (Rouseble) Engrupted RAM Delegation	າ⊿
-	4.1 Encrypted RAM Delegation	24 24
	4.1 1 Security Analysis	24 26
	4.2 Reusable Encrypted RAM Delegation	20
	4.2.1 Security Analysis.	31
5	Rate-1 NIZK	34
	5.1 Construction	35
	5.2 Security Analysis	36
6	Homomorphic NIZK	39
-	6.1 Definition	39
	6.2 Boolean Circuit Decomposition	40
	6.3 Construction	41
	6.4 Security Analysis	43
_		
7	Rate Preserving Maliciously Secure mrNISC	46
	7.1 Definition	40
	7.2 Construction \dots	47
	7.3 Security Analysis	50
8	Multi-Hop zkBARG with Somewhere-Extraction	55
	8.1 Definition	55
	8.2 Construction	57
	8.3 Security Analysis	59

9	Publicly-Verifiable Succinct Delegation of Committed Programs		
	9.1 Definition	64	
	9.2 Construction	66	
Α	Additional Definitions	75	
	A.1 Somewhere Extractable Hash Families	75	
	A.2 Rate-1 Multi-Key Fully Homomorphic Encryption	76	
В	RAM Delegation with Split Configuration	77	
	B.1 RAM Delegation Construction	78	
	B.2 Security Analysis	80	

1 Introduction

Can we delegate a long computation, $y = \mathcal{M}(x)$, yet verify its validity using a short certificate π ? This question is one of the most fundamental problems in theoretical computer science. Delegation of computation has immense practical applications, in part due to rapid rise in popularity of blockchains and cloud services. While constructing non-interactive succinct proofs for delegation of computation is believed to be impossible information theoretically, this task is known to be feasible under computational hardness as long as soundness of the system is desired only against polynomial time adversaries [BCC88, Kil92, Mic94].

Succinct non-interactive arguments (SNARGs) [Mic94] are powerful proof systems that solve the problem of *publicly verifiable delegation*. Over the last several years, numerous works [KR09, KRR14, BHK17, PR17, BKK⁺18, CCH⁺19, KPY19, JKKZ21, CJJ21, CJJ22a, KVZ21, WW22, BBK⁺23, NWW23, JKLV24] have constructed SNARGs for various subclasses of **NP**. All these are useful for publicly verifiable delegation of different types of *deterministic computation*. To go beyond delegation of deterministic computation, we need SNARGs for general *non-deterministic* computations. While the seminal work of Micali [Mic94] designed SNARGs for **NP** in the random oracle model [BR93], designing SNARGs for **NP** in the "plain model" under standard falsifiable cryptographic assumptions remains a grand challenge, and requires overcoming strong barriers [GW11].

In this work, we approach the problem of designing publicly verifiable delegation for *non-deterministic computations* from a different direction. Our philosophy is to anchor ourselves to the plain model and standard cryptographic assumptions, and explore what applications and forms of delegation for non-deterministic computations are achievable.

We propose and define a new notion of delegation system that we call *encrypted RAM* delegation. We show several applications of encrypted RAM delegation that were previously known only from SNARGs for **NP**, or select algebraic assumptions. We provide a new construction for encrypted RAM delegation that is provably secure under a wide variety of standard cryptographic assumptions such as DDH, LWE, or k-linear assumption.

We begin by informally introducing the notion of encrypted RAM delegation. An encrypted RAM delegation scheme for a machine \mathcal{M} consists of a prover and verifier algorithm¹. The prover algorithm, Prove, takes two input strings x_{pb} and x_{pr} , where x_{pb} corresponds to a large *public* input and x_{pr} is a *private* input. It generates a short proof π that is used by the verifier algorithm, Verify, to check validity of computation of \mathcal{M} , while given only x_{pb} as an additional input. In words, consider $y = \mathcal{M}(x_{pb}, x_{pr})$, then encrypted RAM delegation allows proving validity of this computation to a verifier that only gets x_{pb} and y, but never knows x_{pr} .

Intuitively, for soundness we require that if a proof π gets verified for some input x_{pb} and output y, then there must exist a private input x_{pr} such that $y = \mathcal{M}(x_{pb}, x_{pr})$. Even further, we consider extraction soundness, where such an x_{pr} should be efficiently extractable from an accepting proof π . To formally capture an appropriate notion of privacy for x_{pr} , we consider a zero-knowledge style guarantee for delegated proofs, called input hiding. Intuitively, it states that an honestly computed proof π for $y = \mathcal{M}(x_{pb}, x_{pr})$ should be efficiently simulatable, given only x_{pb} and y.

At this point, the reader might be wondering:

- 1. How does this capture delegation for (a non-trivial class of) non-deterministic computations?
- 2. Why do we refer to it as "encrypted" RAM delegation?

¹Similar to vanilla RAM delegation, we also consider a setup algorithm to sample a CRS, but ignore it for simplicity of exposition.

3. Why does this not face similar strong barriers [GW11] as SNARGs for NP?

Let us answer these questions one at a time. Informally, we view x_{pr} as the non-deterministic portion of the computation. That is, x_{pr} is only available to the prover, and not the verifier. Moreover, we require strong hiding guarantees for x_{pr} from a malicious verifier. To further illustrate this, consider the following application.

RAM delegation over encrypted data. Consider a public cloud server that stores a large encrypted database D. For example, this could be a corpus of facial images collected by airport authorities, or patient health data collected by NIH. Suppose an authorized user/entity that has the decryption key, dk, wants to run a long computation \mathcal{M} on the database D. This could be FBI running a facial recognition software to identify potential threats, or a research study group training a huge ML model to develop a predictor for heart diseases. Consider an auditor that wants to validate the result of the computation, but can only access the encrypted database D, and not the decryption key dk. For example, this could be the DOJ (Department of Justice) vetting the credibility of FBI's reports, or the OIG (Office of Inspector General) auditing the research study.

The authorized user/entity can perform this computation given dk and moreover, using basic RAM delegation (for deterministic computation) they can create a short certificate π to prove validity of computation. Unfortunately, π can **not** be verified by anyone that does not possess dk! Thus, any third party that wants to validate the correctness of the computation needs to be able to read the entire database in the clear.

Encrypted RAM delegation gets around the above limitations, and provides a solution to generate public verifiable proofs for delegated computations over "encrypted data". Quite simply, using encrypted RAM delegation, the authorized user/entity can generate a proof π by treating $x_{pb} := D$ and $x_{pr} := dk$ such that π can be verified given only encrypted database D^2 . Because of the above perspective, we refer to our concept as encrypted RAM delegation, and the non-determinism aspect is highlighted by the fact that the verifier does not know x_{pr} .

While the above answers the first two questions, it does not fully address the feasibility question surrounding encrypted RAM delegation. To answer that, we start with our main observation about useful desiderata for encrypted RAM delegation. Looking from the perspective of applications, we observe (as well as demonstrate later) that the private input x_{pr} is usually quite small in many applications. For instance, x_{pr} is simply a fixed size decryption key dk in the above auditing application. Thus, if the proof size is not fully succinct w.r.t. the length of private input, then it suffices for many applications. Importantly, this would ensure that known proof barriers [GW11] do not prohibit the design of encrypted RAM delegation. In words, our plan is to require the proof size to be as large as the size of *private* input, but *not* scale with the size of *public* input. Thus, $|\pi| = \text{poly}(\lambda, \log |x_{pb}|, |x_{pr}|)$. As we show later, this has the advantage that we can design encrypted RAM delegation relying only on polynomial hardness of many standard cryptographic assumptions as well as it suffices to get best possible solutions for many new applications.

We also consider a strengthening of encrypted RAM delegation to support fully succinct proofs, as in we design proofs whose size does not grow even with $|x_{pr}|$. We call them *reusable* encrypted RAM delegation. Our intuition is that if the same private input, x_{pr} , is being reused to generate multiple proofs π_1, π_2, \ldots for either different x_{pb} or different machines \mathcal{M} , then we can reduce the overall proof size by amortization. Basically, we will generate a pre-processed proof $\tilde{\pi}$ using x_{pr} such that each proof π_i (w.r.t. x_{pr}) can be thought of as two sub-proofs—a reusable portion $\tilde{\pi}$, and

²Technically, a verifier only needs to read the digest of D

a non-reusable portion π'_i . Here only π'_i depends on x_{pb} and \mathcal{M} . Moreover, the size of each π'_i does not grow with $|x_{pr}|$, i.e. $|\pi'| = \operatorname{poly}(\lambda, \log |x_{pb}|, \log |x_{pr}|)$. The security properties for such resuable encrypted RAM delegation systems can be appropriately generalized.

1.1 Overview of our applications

We will start with our applications, and defer the explanation of our construction and proof for (reusable) encrypted RAM delegation to later.

Boosting Non-Interactive Zero-Knowledge Arguments to Rate-1. A non-interactive zeroknowledge argument system (NIZK) [BDSMP91, BFM88] for an NP language \mathcal{L} is a standard two-party proof system defined in the CRS model, where a prover wants to convince a verifier that an instance $x \in \mathcal{L}$, without the revealing any information about the corresponding witness ω . All parties have access to a public crs. Soundness states that any polytime cheating prover should not be able to convince the verifier of a false assertion, while zero-knowledge requires that the proof string π should be simulatable given just a valid instance x. A stronger notion of knowledge soundness requires the system to additionally support efficient extraction of a valid witness ω from any accepting proof π for an instance x. NIZKs are a fantastic and fundamental cryptographic tool with numerous applications across cryptography.

Almost all NIZK systems built from standard tools³ result in proofs and CRS of large size. That is, $|\pi|$, $|crs| = poly(\lambda, |x|, |\omega|)$. In particular, the proof size is much larger than the witness size, $|\omega|$. One notable exception is the work of Gentry et al. [GGI⁺15] that designed rate-1 NIZKs. Here rate-1 refers to the fact that $|\pi| = |\omega| + poly(\lambda)$. Their work, however, crucially relies on fully-homomorphic encryption (FHE) [RAD⁺78, Gen09] to perform hybrid encryption to reduce the proof size. Thus, despite the fact that the community has developed multiple approaches for designing NIZKs from a variety of standard tools and assumptions [BFM88, GO94, FLS99, SW14, PS19, JJ21, Wat24] over the last 30+ years, the only known construction we have for rate-1 NIZKs relies on learning with errors [Reg09].

In this paper, we show how to use encrypted RAM delegation to generically design a rate-1 NIZK scheme with knowledge soundness. In a very recent work, Cheng and Goyal [CG24] proved that rate-1 NIZKs with knowledge soundness are essentially optimal, and going beyond rate-1 faces the same barriers as SNARGs for **NP** [GW11]. Thus, our transformation essentially designs NIZKs with knowledge soundness and optimal efficiency from nearly all standard assumptions. Moreover, our rate-1 NIZK construction also enjoys many useful properties such as straight-line extraction as well as it is a universal proof system, which means the CRS is not tied to a single **NP** language, but can be reused for any **NP** language.

The technique for designing rate-1 NIZKs is fairly simple. To generate a NIZK proof for instance x with witness ω , simply start by encrypting ω using a one-time pad key K, where K could be computed as the output of a pseudorandom generator on a random, but fixed size, seed sd. Now the NIZK proof can be generated by running the encrypted RAM delegation with $x_{pb} := (x, \omega \oplus K)$ and $x_{pr} := sd$. This results in a proof π that can be verified given just x_{pb} . That is, a verifier does not need x_{pr} . Therefore, the NIZK proof can be set as π and the one-time pad encryption of ω , i.e.

 $^{^{3}}$ For the purposes of this overview, we roughly consider direct cosntructions from standard falsifiable cryptographic assumptions to be standard tools, and those based on multilinear maps, indistinguishability obfuscation, random oracles, or other non-falsifiable assumptions not to be.

 $\omega \oplus K$. Clearly, the verifier can run the encrypted RAM delegation verified to check the validity of the proof.

The fact that the resulting NIZK proof is rate-1 follows immediately from the fact that one-time pad is rate-1 and the proof size in encrypted RAM delegation does not grow with x_{pb} , but only x_{pr} which is of fixed size (say λ). For zero-knowledge, we can use a simple hybrid argument to first rely on the input hiding property of encrypted RAM delegation proof, and then use a one-time pad encryption argument to remove any remaining information about ω . Lastly, for knowledge soundness, we can rely on the extraction soundness property for encrypted RAM delegation to recover the seed sd and use it to extract the witness ω . At its core, our encrypted RAM delegation scheme uses a poor-rate NIZK scheme to guarantee the input hiding property, thus if the underlying zero-knowledge property is statistical, then so is the zero-knowledge guarantee of our rate-1 NIZKs.

The above construction works for any scheme with an apriori bounded size witness. We can instead handle unbounded length witnesses quite easily by rather leveraging a pseudorandom function instead of a pseudorandom generator. The details of our transformation are given in Section 5.

Fully-Homomorphic NIZKs. The concept of homomorphism in NIZKs was introduced by Ananth et al. [ADKL19]. The intuition behind fully-homomorphic NIZK was to enable homomorphic computations over independently generated NIZK proofs. To avoid trivial impossibilities, the standard approach to formalize homomorphism in NIZK systems was to consider an **NP** language, where each instance contains a pair of circuit and boolean output (C, b) and a witness ω is an input string such that $C(\omega) = b$. In addition to the standard prover/verifier algorithms, homomorphic NIZK systems have a special proof evaluation evaluation, **Eval**. It takes k tuples of the form $\{(C_i, b_i, \pi_i)\}_{i \in [k]}$ and a boolean circuit C that takes k bits as input. 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.

Unlike FHE, there are no special compactness requirements from evaluated proofs⁴. Despite that, there does not exist any canonical approach to design homomorphic NIZK. The reason is that the naive approach of simply setting the evaluated proof as π_1, \ldots, π_k along with circuit Chas several issues. First, the evaluated proof does not necessarily satisfy zero-knowledge property, as the evaluated proof leaks intermediate wire values in the circuit computation. For example, if C is just a tree of OR gates and b' = 1, then (ideally) the evaluated proof should hide which of the input bits from b_1 to b_k are 1. However, the naive approach leaks that in the clear. Second, the evaluated proof does not resemble an 'honestly generated proof' for the instance (C', b'). Thus, a verifier can clearly distinguish whether the NIZK proof is an evaluated proof or it is a fresh proof.

To address these issues and enable new applications, Ananth et al. [ADKL19] introduced a new unlinkability property for homomorphic NIZKs. It 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.

Prior to this work, the only other construction for homomorphic NIZKs [ADKL19] relied on the seminal pairing-based NIZK construction by Groth-Ostrovsky-Sahai (GOS) [GOS06]. At a

⁴Actually requiring compactness of evaluated signatures would necessitate the use of SNARGs with adaptive knowledge soundness, which face even stronger barriers [GW11, CGKS23].

very high level, during each homomorphic evaluation, they followed a two-step approach: (1) use a commitment to hide the output wires of each input proof, and (2) generate a fresh NIZK proof proving that it knows openings to the output wire commitments such that thet satisfy the circuit being evaluated. To ensure unlinkability, the evaluator also had to randomize all wire commitments as well as re-randomize existing NIZK proofs to match these randomized commitments. In a few words, their core idea was to use the fact that GOS NIZKs were malleable, i.e. one can rerandomize a NIZK proof for certain special relations. For more details, readers are encouraged to consult [GOS06].

While it might appear that we always need to cleverly exploit algebraic properties of a NIZK proof system to enable (unlinkable) homomorphism, our main insight is that this is not necessary! We show that by carefully composing NIZK proofs we can design fully-homomorphic proofs from a variety of cryptographic assumptions beyond pairings, such as DDH and LWE. For instance, this gives the first post-quantum homomorphic NIZK to the best of our knowledge.

Our main idea is rather straightforward. For simplicity, let us start with a rate-1 NIZK system with knowledge soundness as a building block, rather than encrypted RAM delegation. Although we really need some special features from a rate-1 NIZK which are somewhat strong, we observe that our encrypted RAM delegation can be easily generalized to obtain all such desired features. Thus, we stick to rate-1 NIZKs for ease of exposition.

Suppose we are given a sequence of instances $\{(C_i, b_i)\}_{i \in [k]}$ and proofs $\{\pi_i\}_{i \in [k]}$, and now we want to homomorphically evaluate a circuit C on it. Our plan is to simply generate a new NIZK proof for the statement (C', b'), where $C' = \text{Compose}(C, C_1, \ldots, C_k)$ is the composed circuit and $b' = C(b_1, \ldots, b_k)$ is the evaluated output. That is, we generate a new NIZK proof for following:

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

The witness for this proof consists of (b_1, \ldots, b_k) and (π_1, \ldots, π_k) . It turns out the above design for homomorphic NIZKs guarantees completeness, soundness, zero-knowledge, as well as efficiency. Because we use rate-1 NIZKs, thus the proof size only grows additively with each homomorphic evaluation, and is therefore efficient. Moreover, by using straight-line knowledge soundness property, we can prove (knowledge) soundness of our homomorphic NIZKs. And, similarly, completeness and zero-knowledge follows.

Unfortunately, this does not yet ensure unlinkability. The problem is that, just by looking at the size of the evaluated proof, one can distinguish an evaluated proof from a freshly generated proof. While this might seem a big barrier, we notice by relying on a simple algorithmic trick, we can bypass this issue. Basically, our idea is essentially to compute a fresh NIZK proof by running the evaluation algorithm. This means we will simply "decompose" each circuit C into non-overlapping groups of "atomic circuits". And, to generate a NIZK proof, a prover would first decompose each circuit into its atomic decompositions, and then use homomorphic evaluation to create a proof. The important part is each homomorphic operation only evaluates an atomic circuit. Therefore, the size of a freshly generated NIZK proof will be the same as any arbitrarily evaluted NIZK proof for the same instance. This relies on the fact that the "atomic circuit decomposition" technique that we rely on generates deterministic encodings.

Later in the main body, we provide our algorithms for such deterministic atomic circuit decomposition. We remark that the core approach behind our decomposition is to view each circuit as a directed acyclic graph, and identify the largest set of vertices that can be safely 'cut' to partition a single directed acyclic graph into directed acyclic graphs that each could be restored as a circuit with at least one logic gate. We refer the reader to Section 6 for a full formal definition and our construction. We remark that our homomorphic NIZK construction supports unbounded homomorphism, and is truly *fully* homomorphic. That is, we do not specify any a-priori upper bound on the size or number of circuits that can be homomorphically evaluated.

Rate-Preserving Semi-Malicious to Malicious MPC Compilers. Multiparty Computation (MPC) [Yao86, GMW87] protocols are a staple in modern cryptography as they allow mutually distrusting parties to jointly compute a function f over distributed inputs, with the guarantee that none of the honest parties' inputs will get compromised. There are multiple approaches in the literature categorize different flavors of security for MPC protocols. Two such popular formulations are of *semi-malicious* and *malicious* security. Briefly, semi-malicious security states that the protocol must remain secure as long an attacker honestly runs the algorithms, for every corruped party, as defined in the protocol description. But, it can arbitrarily choose the randomness distribution to maliciously influence its messages in the protocol. Fully malicious security states that security must hold even against attackers that can arbitrarily deviate from the protocol description.

A very popular approach in the design of MPC protocols is to design an MPC protocol that achieves only semi-malicious security, and later compile it to a maliciously secure protocol generically. Such a compiler was first presented by Asharov et al. [AJLA⁺12] who designed a compiler to transform semi-malicious MPC protocols into maliciously secure protocols. Their compiler relied on NIZK systems, and preserved the underlying function class \mathcal{F} . In a few words, their approach was to require each party to additionally provide a NIZK proof along with each message, where the NIZK proof proves that it correctly followed the protocol for the corresponding round. By a careful utilization of NIZKs, one could prove such a compiler amplifies security to full malicious security.

However, using vanilla NIZKs is very inefficient. This adversely affects the efficiency of the original (semi-malicious) MPC protocol. While it is well known that by using general-purpose zero-knowledge SNARGs with knowledge soundness (zk-SNARKs), one can significantly improve the efficiency of the compiler. (Refer to [DGMR21] for details.) By significantly we mean that the total communication complexity of the compiled protocol is asymptotically nearly identical to that of the original protocol. But this clearly relies on a very strong non-falsifiable assumption in the form of zk-SNARKs.

In this work, we show that reusable encrypted RAM delegation is a great tool to build (optimal) rate-preserving compilers from semi-malicious MPC to malicious MPC. By rate-preserving, we again mean that the communication complexity does not degrade with the size of the MPC functionality, inputs, or the number of parties. To the best of our knowledge, this gives the first rate-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 noninteractive secure computation (mrNISC) protocols [BL20]. A mrNISC protocol is a powerful MPC protocol that gives fantastic solutions to round-optimal MPC protocols. In a mrNISC protocol, each party (say P_i) encodes its private inputs (say x_i) using a special **Com** algorithm. Such an encoding, \hat{x}_i , is thought to be either posted on a public bulletin board, or broadcasted widely through the network. 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 a special evaluated encoding \hat{f}_i using the FuncEnc algorithm. This only requires the input encodings $\{\hat{x}_i\}_{i\in S}$ and the *decommitment* information corresponding to \hat{x}_i . All these encodings together can be used to learn $f(x_1, \ldots, x_n)$, but nothing more. The point behind mrNISC is that by making the first round of the protocol to be independent of the function f and the set S of parties, we can amortize the cost of computing k functions in just k + 1 rounds.

Over the last few years, mrNISC has received tremendous attention from the community. Today we have multiple approaches to semi-maliciously secure protocols [BL20, BJKL21, AJJM21, Shi22] from standard assumptions. Despite great progress, we do not have any approaches to compile them into maliciously secure protocols without significantly degrading their efficiency under standard falsifiable assumptions. We show that a reusable encrypted RAM delegation system is a great fit for designing an optimal rate-preserving compiler for malicious security in mrNISC.

The main idea is as follows. Along with every input encoding (commitments \hat{x}_i to private inputs x_i), we will generate a pre-processed proof $\hat{\pi}_i$ using x_i and its decommitment information as the *private* input for our encrypted RAM delegation. Together $\hat{\pi}_i$ and \hat{x}_i correspond to the input encoding for the compiled protocol. Next, while generating function encodings, each party first generates the function encoding \hat{f}_i as is for semi-malicious protocol. This can be generated by using $\{\hat{x}_j\}_{j\in S}$ and the *decommitment* information for \hat{x}_i . Next, it generates the actual encrypted RAM delegation proof, by setting \hat{f}_i and all the encodings $\{\hat{x}_j\}_{j\in S}$ as the *public* input for our encrypted RAM delegation. Given this, each party outputs the semi-malicious function encoding as well as the encrypted RAM proof as the final function encoding. An evaluator simply checks that all the proofs verify, and then it uses the semi-malicious security of the underlying mrNISC protocol, and knowledge soundness and input-hiding property of our reusable encrypted RAM delegation, we can provide malicious security of our compiled protocol.

Our above compiler preserves the function class for the underlying mrNISC protocol, and gives nearly rate-preserving compilation. The reason it is nearly rate-preserving because the decommitment information relies on the length of input x_i . Thus, even when our pre-processed proof is also rate-1, the resulting input encoding is going to be rate- $\frac{1}{2}$. However, we show that this is not a major hurdle, and we can design fully rate-preserving compiler by slightly altering our compiler.

To make the input encoding fully succinct, our approach is to perform a very simple additional level of input delegation. By this we mean, rather than encoding the actual input x_i using the semimalicious mrNISC protocol, we will create an input encoding for a short random seed sd_i , and use this to one-time pad the input x_i (similar to what we did in our rate-1 NIZK construction). With this, we will set the input encoding to contain the one-time pad encryption of x_i , input encoding for sd_i , and a corresponding pre-processed proof $\hat{\pi}_i$. Clearly, the resulting input encoding is truly rate-1 as only the one-time pad encryption depends upon the input x_i .

However, to ensure that we can still compute the function f, we need to non-trivially alter the function encoding procedure. Specifically, we have to generate the function encodings not for f, but for a new function g_f that does the following:

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

The security of this new protocol can also be argued similar to the previous protocol by additionally relying on pseduorandomness of PRG. We highlight that although g_f is a simple extension of f, it is not the same as f. Thus, this suggests either we can get optimal-rate by altering the function class slightly, or we can get a very mildly worse rate while preserving the function class. For further discussion on definitions of mrNISC, construction, efficiency, and security, please refer to Section 7. More applications: composable zkBARGs, delegation of committed programs, etc. Beyond the aforementioned applications, we show that reusable encrypted RAM delegation enables many more interesting applications that we briefly discuss below.

Composable zkBARGs. Batch arguments (BARGs) allow a prover to generate a short proof for a 'batch' statement that $x_1 \in \mathcal{L} \land \ldots \land x_k \in \mathcal{L}$. Soundness states that an attacker cannot create an accepting proof for a batch of instances containing at least one instance $x_i \notin \mathcal{L}$. Somewhere extractable BARGs (seBARGs) [CJJ21, CJJ22a] are a mild strengthening, that enable witness extraction for a single statement at some hidden trapdoor index i^* from any accepting proof. Composable or multi-hop batch arguments (BARGs) were recently introduced by Devadas et al. [DGKV22] as a new tool to enable many new applications such as aggregate signatures with unbounded aggregation [BGLS03]. They constructed composable seBARG from any rate-1 seBARG. Informally, a multi-hop (se)BARG allows for succinct composition of multiple (se)BARG proofs (which could themselves be composed proofs).

In this work, we provide a simple construction for rate-1 (somewhere-extractable) zero-knowledge BARGs (zkBARGs). Our construction essentially combines the techniques of [DGKV22]. The idea is create a NIZK proof for each BARG proof before composing it further using BARGs. Naturally, the zero-knowledge property follows directly from that of the NIZK. And, since both the underlying seBARGs and NIZKs are rate-1, thus the resulting 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 somewhere extractability for zkBARGs too. This immediately gives us privacypreserving aggregate signatures supporting unbounded aggregation, where by privacy preserving we mean an aggregated signature completely hides the original signature. We provide more details in Section 8.

Succinct Delegation of Committed Programs. In a recent work, Ghosal, Sahai, and Waters [GSW23] proposed a new mechanism to assist users that wish to delegate the responsibility of hosting a program to a server. The desired functionality was that any third-party client can be convinced that they are indeed receiving the correct output of the program on some public input, without the knowledge of program beyond a commitment and without trusting the server. Ghosal, Sahai, and Waters designed such publicly verifiable succinct delegation protocols by carefully exploiting many intricate properties of the vanilla RAM delegation construction of Choudhuri, Jain, and Jin [CJJ21]. To that end, they defined a new notion for SNARGs that they called semi-trusted SNARGs.

Here we show that reusable encrypted RAM delegation immediately gives such succinct delegation protocols. Moreover, our construction guarantees a much stronger zero-knowledge property. In particular, we give a generic way to build zero-knowledge delegation of committed program from reusable encrypted RAM delegation as well as a stronger definition of soundness than was considered in [GSW23]. We provide more details in Section 9.

Context-Hiding Multi-Hop/Multi-Key Homomorphic Signatures. Homomorphic signatures [JMSW02, AB09, BFKW09, BF11, GVW15b] enable computations on secretly signed data. Given a circuit C, they enable derivation of an evaluated signature $\sigma_{C,y}$ from a signature σ_x for data x. In words, $\sigma_{C,y}$ is an unforgeable token validating possession of a signature σ_x on some data x such that C(x) = y. A highly desirable property in homomorphic signatures is to support homomorphic evaluation on evaluated signatures, while ensuring that evaluated signatures do not reveal non-trivial information about the original data x. These two properties are commonly regarded as multi-hop homomorphic evaluation and context hiding [GVW15b].

In a very recent work, Afshar, Cheng, and Goyal [ACG24] designed the first homomorphic signature scheme that satisfied general multi-hop homomorphic evaluation and context hiding from standard falsifiable assumptions. All other prior works suffered from one or more limitations as explained in detail in [ACG24]. However, to support context hiding, their construction relied on rate-1 NIZKs which were previously known only under LWE assumption. Thus, their construction could not be generalized to other cryptographic assumptions. By combining our rate-1 NIZK schemes, we obtain the first non-lattice-based homomorphic signature scheme satisfying general multi-hop homomorphic evaluation and context hiding, from sub-exponential hardness of DDH/klinear assumption, as a simple corollary. Moreover, this result also directly generalizes to multikey setting [GVW15b, FMNP16] as well. Prior to our work, context hiding multi-hop multi-key homomorphic signatures were not known from falsifiable non-lattice assumptions. We refer the reader to [ACG24] for a detailed discussion on the current state of the art.

We believe there are many more applications of encrypted RAM delegation that will be discovered in the future. For instance, we suspect that encrypted RAM delegation would lead to new results in the domain of attribute-based signatures [MPR11], ciphertext-rate-preserving CPA-to-CCA transformations for encryption systems [NY90], etc.

1.2 Overview of our Encrypted RAM construction

We start by recalling the problem of the RAM delegation where a prover generates a proof π to convince a verifier that a RAM machine \mathcal{M} accepts an input x within T steps such that both the proof size and the verifier's running time only grow with $\operatorname{polylog}(T, |x|)$. In this work, we consider the problem where the input is partially private, namely, $x = (x_{pb}, x_{pr})$ for some private (resp. public) input x_{pr} (resp. x_{pb}). More specifically we consider an *encrypted* RAM delegation scheme as a prover that, given (x_{pb}, x_{pr}) , generates a proof π , and a verifier that, given a short digest h_{pb} of x_{pb} , verifies π . Let $\mathcal{L}_{\mathcal{M},T}$ describe the language of the machine \mathcal{M} consisting of all (x_{pb}, x_{pr}) that are accepted by \mathcal{M} within T steps. We require our scheme to satisfy the following properties

- 1. Soundness: no PPT adversary can find an accepting (x_{pb}, π) s.t. for all $x_{pr}, (x_{pb}, x_{pr}) \notin \mathcal{L}_{\mathcal{M},T}$
- 2. Straight-line extraction: there is an extractor \mathcal{E} s.t. for any accepting (x_{pb}, π) , it holds that $x^*_{\mathsf{pr}} \leftarrow \mathcal{E}(x_{\mathsf{pb}}, \pi)$ and $(x_{\mathsf{pb}}, x^*_{\mathsf{pr}}) \in \mathcal{L}_{\mathcal{M},T}$,
- 3. Input hiding: there is a simulator that generates indistinguishable proofs without knowing x_{pr} ,
- 4. Efficiency: the proof size and verification time grow with $poly(\log T, \log |x_{pb}|, |x_{pr}|)$.⁶

We further consider an additional *reusability* notion for the encrypted RAM delegation. This property allows one to generate an encoding eh_{pr} of the private input x_{pr} , with a long proof of the encoding π_{enc} , and then reuse the encoding to generate short proofs $\pi_{edel}^{(i)}$ for the computation on different public inputs $x_{pb}^{(i)}$. Thus, we have $|\pi_{enc}|$ grow with $poly(|x_{pr}|)$, but $\pi_{edel}^{(i)}$ only grow with $polylog(|x_{pr}|)$.

⁵We actually prove stronger soundness in the main body, where a cheating prover need not even output the full x_{pb} . ⁶Note that if the proof does not grow with $poly(|x_{pr}|)$ then it will imply SNARGs for **NP**.

From BARGs to (vanilla) RAM delegation. Before describing our construction we recall the canonical construction of the RAM delegation from BARGs and why it falls short in satisfying our required properties. For the computation of the RAM machine \mathcal{M} , let cf_0 be the initial configuration (including the input), cf_i be the *i*-th configuration of the machine, and $nxt-cnfg(cf_{i-1}) = cf_i$ be the step function that computes the next configuration of the machine. The idea for RAM delegation is to break down the global consistency of the configurations, to local consistency of any two consecutive configurations cf_{i-1} and cf_i . Namely, in the canonical construction the prover first generates a succinct commitment c of (cf_0, \ldots, cf_T) , then generate a BARG proof where the *i*-th instance, given the witness $(cf_{i-1}, cf_i, op_{i-1}, op_i)$ checks if $cf_i = nxt-cnfg(cf_{i-1})$, and whether op_{i-1} and op_i are valid openings of cf_{i-1} and cf_i w.r.t. c. To instantiate the commitment above a hash tree with local opening is used.

First note that in the canonical construction the proof size grows with the configuration size which is not ideal. The works of [CJJ22a, KLVW23] show how to construct efficient RAM delegation schemes that have the desired polylogarithmic dependency on the configuration size (by additionally hashing each configuration). However, the more important issue, that is the leakage of information about the input remains unsolved. By analyzing the canonical construction we realize that both the commitment and the BARG proof leak information about the inputs. Thus even if we use succinct hiding commitments with local openings, the above construction fails in hiding any information about the inputs. The main reason is that the configurations do not hide information about the inputs, and the BARG proof does not hide the configurations.

Achieving input hiding through encrypt and prove trick. We start with a RAM delegation scheme and show how to generically transform it to an encrypted RAM delegation. Here we consider RAM delegation as a prover that given an initial configuration cf_0 generates a proof π and a verifier that given a digest of cf_0 verifies the proof. The idea is to generate an encoding of the memory, then prove the well-formedness of the encoding, then generate a RAM delegation proof and use the encoded memory to generate a short proof, proving the validity of the delegation. For this approach to work, the encoding should be hiding and succinct, and the well-formedness proof as well as the proof of the validity of the delegation should both hide the information about the witnesses. Thus, we hash the memory and then encrypt the hash value to generate the encoding, then we use NIZK to generate the rest of the proofs. While this idea seems directly applicable, one issue that arises is that the RAM delegation proof corresponds to a digest of the entire initial configuration, however the proof size grows polynomially with the size of the encoded part. Therefore, instead of encoding the entire memory we will have to only encode the private input, which means that memory has to be split to different parts and digest each part separately. It is known to be a well-believed folklore fact that existing designs for RAM delegation can be very easily extended to handle such split memories. For completeness, we formally provide the design of such a splittable RAM delegation scheme and its proof in Appendix B. Given all the ideas above we present our encrypted RAM delegation scheme as follows:

- Let $cf_0 = (x_{pb}, x_{pr}, \Sigma)$. Compute digest of the initial configuration, i.e. $(h_{pb}, h_{pr}, h_{\Sigma})$ using a hash tree.
- Compute an encoding of the private input eh_{pr} by encrypting h_{pr} under uniform randomness r.
- Compute NIZK proof π_{enc} proving knowledge of x_{pr} corresponding to eh_{pr} using (x_{pr}, h_{pr}, r)

as witness.

- Compute the a RAM delegation proof $del.\pi$ of the computation.
- Compute NIZK proof π_{edel} of the validity of the del. π as well as the correctness of encryption where the statement is $(h_{pb}, eh_{pr}, h_{\Sigma})$ and the witness is $(h_{pr}, r, del.\pi)$.
- Let $\pi = (\mathsf{eh}_{\mathsf{pr}}, \pi_{\mathsf{enc}}, \pi_{\mathsf{edel}}).$

Here we analyze our construction.

- **Input hiding.** Note that the proof hides all the information about x_{pr} by the security of the encrypted system and the NIZK proof, so the scheme is input hiding.
- Straight-line extraction. Note that if the underlying NIZKs have straight-line extraction, then, we can extract some x_{pr}^* , check it is consistent with eh_{pr} , then extract the witness from π_{edel} , use the correctness of the encryption to argue the consistency of the extracted h_{pr}^* (from π_{edel}) and x_{pr}^* , and finally rely on the security of the RAM delegation scheme to argue that $(x_{pb}, x_{pr}^*) \in \mathcal{L}_{\mathcal{M},T}$.
- Efficiency. Additionally, the size of all the hash values is λ bits, thus $|\mathsf{eh}_{\mathsf{pr}}| = \mathsf{poly}(\lambda)$, and $|\pi_{\mathsf{enc}}| = \mathsf{poly}(\lambda, |x_{\mathsf{pr}}|)$. Additionally, by the efficiency of the RAM delegation $|\mathsf{del}.\pi| = \mathsf{poly}(\lambda, \log T, \log |x_{\mathsf{pb}}|, \log |x_{\mathsf{pr}}|)$, thus π_{edel} is of the same size. The efficiency of the verification time follows the same argument.
- **Reusability** We notice that our construction of the encrypted RAM delegation, already satisfies reusability. Namely, eh_{pr} and π_{enc} are generated independent of the x_{pb} and machine \mathcal{M} , thus once they are generated, a prover can use $aux = (x_{pr}, r)$ in addition to $x_{pb}^{(i)}$ to generate $\pi_{edel}^{(i)}$.

The above construction highlights the ease with which we can generalize the core ideas behind RAM delegation construction to support delegation of restricted forms of non-deterministic computations such as computation over encrypted data. In the main body, we provide an even simpler design for encrypted RAM delegation. Regardless, we believe the above interpretation of core ideas beneath vanilla RAM delegation might be useful to consider further strengthenings. Thus, we provide the above construction as part of our overview. We conclude our overview by stating that we view identifying reusable encrypted RAM delegation as a new abstraction and developing different approaches to use it enable new applications as our core contribution.

1.3 Related Work

Partially hiding functional encryption. Gorbunov, Vaikuntanathan, and Wee [GVW15a] introduced the notion of *partially hiding* predicate encryption as an amalgamation of standard attribute-based encryption [SW05, GPSW06] and predicate encryption [BW07, KSW08]. The goal was to split up the attribute into two parts (public and private). Their scheme allowed "heavyweight" computation over public attributes and only a "lightweight" private processing for secret attributes. Later on, [AJL+19] further generalized the concept to partially hiding functional encryption [BSW11], which allows evaluation of function f(x, y) such that only output and x is revealed.

One could view encrypted RAM delegation as a similar generalization of RAM delegation, where the goal is to be able to generate a proof for the validity of RAM computation $\mathcal{M}(x_{\mathsf{pb}}, x_{\mathsf{pr}}) = y$, while keeping x_{pr} private from the verifier.

NIZK Arguments. NIZK arguments [BDSMP91, BFM88] are computationally sound noninteractive counterpart of zero-knowledge proofs [GMR85]. Two very popular approaches to design NIZK are the hidden-bits paradigm [FLS99, QRW19] and the Fiat-Shamir heuristic [FS86] by using correlation-intractable hash function [CGH04]. Using these two approaches and more, we have seen numerous designs for NIZK for a variety of assumptions such as DLIN [GOS06, GOS12], sub-exponential DDH [JJ21], CDH [CHK03, CJJQ23], LPN [BKM20, DJJ24, CJJQ23], LWE [PS19, Wat24]. Moreover, there are generic construction known from objects such as weakly succinct SNARGs [KMY23], vector trapdoor hash [BCD⁺24], and batch arguments [BKP⁺23, CW23, BWW23].

Reusable non-interactive MPC. Reusable non-interactive MPC [IKO⁺11, BGI⁺14] is a specialization of the general concept of MPC [Yao82, Yao86, GMW87], and it has received a lot of attention recently as removes the need for constant interaction between users. For a detailed discussion, we refer the reader to [BL20] who formally introduced the concept of mrNISC. They constructed a mrNISC protocol in semi-malicious model using a round collapsing MPC argument [GS17, GS22, BL18] from witness encryption and NIZK schemes for a specific commitment language assuming SXDH. Subsequently, post-quantum secure versions of mrNISC was constructed in two concurrent works [AJJM21, BJKL21] from LWE and functional oblivious transfer. An additional advantage of these constructions is that the reusable first round messages need not depend on the number of parties n. Shiehian [Shi22] later used a bootstrapping argument to construct mrNISC using plain LWE.

Concurrent Work. In a recent concurrent work, Branco, Döttling, and Srinivasan [BDS24] also studied the problem of designing optimal-rate NIZK arguments with knowledge soundness. The focus of their work was only purely designing rate-1 NIZK arguments of knowledge. On the other hand, the focus of our work is the new concept of encrypted RAM delegation, and we show multiple new applications of it, where rate-1 NIZK with (straight-line) knowledge extractor are one of our many applications. In terms of comparing our rate-1 NIZK schemes, their focus is on statistical zero-knowledge (which leads to slightly poor rate of 1 + o(1)), while we only prove computational zero-knowledge. In terms of comparing our technical approach, we provide a simple design from RAM delegation, while they need to carefully combine many different building blocks in a non-black-box way.

2 Preliminaries

Notation. We denote the security parameter by λ . By PPT, we denote a probabilistic polynomialtime. All polynomials denoted by $poly(\cdot)$ are positive polynomials. For any finite set $S, x \leftarrow S$ denotes a uniformly random element $x \in S$. 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 $negl(\lambda)$, we define negligible functions. A function $negl : \mathbb{N} \to \mathbb{R}$ is a negligible function if for every $c \in \mathbb{N}$ and for large enough λ , $negl(\lambda) < \lambda^{-c}$.

2.1 Rate-1 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:

- Setup $(1^{\lambda}, k, n, i^*) \to \text{crs.}$ 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 $\text{poly}(\lambda, n, \log k)$ and outputs a common reference string crs.
- Prove(crs, $x_1, \ldots, x_k, w_1, \ldots, w_k$) $\rightarrow \pi$. 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 .

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

Definition 2.1 (seBARG). A somewhere-extractable batch argument scheme seBARG = (Setup, Prove, Verify) for \mathcal{L} is required to satisfy the following properties:

- **Efficiency.** The size of the CRS and the proof is at most $poly(\lambda, \log k, n, m)$, 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 $i^* \in [k]$,

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

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 $\lambda \in \mathbb{N}$,

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

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

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

Remark 2.2. 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 $\lambda \in \mathbb{N}$,

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

Definition 2.3 (zkBARG). A somewhere-extractable zero-knowledge batch argument scheme zkBARG = (Setup, Prove, Verify) for \mathcal{L} is required to satisfy the following property in addition to all properties under Definition 2.1:

Zero-knowledge. There exists a PPT simulator S such that for any PPT adversary A, there exists a negligible function $\operatorname{negl}(\cdot)$ such that for any polynomials $k = k(\lambda)$ and $n = n(\lambda)$, and any index $i^* \in [k]$, for every $\lambda \in \mathbb{N}$,

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

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

Definition 2.4 (Rate-1 seBARG). An seBARG scheme (Setup, Prove, Verify) is said to be <u>rate-1</u> if the proof generated by $Prove(crs, x_1, \ldots, x_k, w_1, \ldots, w_k)$ is of length $m + O(m/\lambda) + poly(\lambda)$.

Remark 2.5 ([DGKV22, PP22, KLVW23]). Assuming either LWE/DLIN/sub-exponential DDH, there exists rate-1 seBARG.

2.2 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., $x = (M_1, \ldots, M_n)$), and the rest of the memory (i.e., $(M_{n+1}, \ldots, M_{n+S})$) 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:

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

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:

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

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:

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

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

2.3 Hash Tree

- Syntax. A hash tree consists of the following polynomial time algorithms:
- $Gen(1^{\lambda}) \rightarrow hk$. 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 $\mathsf{op} \in \{0, 1\}^{\leq \mathsf{poly}(\lambda)}$.
- Write(hk, x, i, b) \rightarrow (h', 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 h' $\in \{0, 1\}^{\operatorname{poly}(\lambda)}$ an opening $\operatorname{op'} \in \{0, 1\}^{\leq \operatorname{poly}(\lambda)}$.
- VerifyRead(hk, h, i, b, op) $\rightarrow 0/1$. This is a read-verification algorithm that takes as input the hash key hk, a hash value $h \in \{0, 1\}^{\mathsf{poly}(\lambda)}$, an index $i \in [|x|]$, a bit $b \in \{0, 1\}$ and an opening op $\in \{0, 1\}^{\leq \mathsf{poly}(\lambda)}$, and outputs 0 (reject) or 1 (accept).
- VerifyWrite(hk, h, i, b, h', op') $\rightarrow 0/1$. This is a write-verification algorithm that takes as input the hash key hk, a hash value $h \in \{0, 1\}^{\mathsf{poly}(\lambda)}$, an index $i \in [|x|]$, a bit $b \in \{0, 1\}$, a hash value $h' \in \{0, 1\}^{\mathsf{poly}(\lambda)}$ and an opening $\mathsf{op}' \in \{0, 1\}^{\leq \mathsf{poly}(\lambda)}$, and outputs 0 (reject) or 1 (accept).

Definition 2.6 (Hash Tree). A hash tree HT = (Gen, Hash, Open, Write, VerifyRead, VerifyWrite) is required to satisfy the following properties:

- Efficiency. The size of hash key hk and hash value h is at most $poly(\lambda)$ and the size of openings is at most $poly(\lambda, \log n)$ 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 $\lambda \in \mathbb{N}$,

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

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

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

Remark 2.7. Reading soundness implies collision resistance of the hash tree.

Remark 2.8 ([Mer87]). Hash trees can be constructed using any family $\mathcal{F} : \{0,1\}^{2\lambda} \to \{0,1\}^{\lambda}$ of collision-resistance hash functions.

2.4 Non-Interactive Zero-Knowledge Arguments (NIZK)

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

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

- Setup $(1^{\lambda}, 1^{n_x}, 1^{n_w}) \to \text{crs.}$ 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.
- Prove(crs, x, w) $\rightarrow \pi$. The prover algorithm takes as input a common reference string crs, an instance x, and a witness w and outputs a proof π .
- Verify(crs, x, π) $\rightarrow 0/1$. The verifier algorithm takes as input a common reference string crs, an instance x, and a proof π . It outputs 0 (reject) or 1 (accept).

Definition 2.9 (NIZK). A non-interactive zero-knowledge proof (Setup, Prove, 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 $\lambda, n_x \in \mathbb{N}$:

$$\Pr[\mathsf{Verify}(\mathsf{crs}, x, \pi) = 1 \land x \notin \mathcal{L} : \mathsf{crs} \leftarrow \mathsf{Setup}(1^{\lambda}, 1^{n_x}, 1^{n_w}), (x, \pi) \leftarrow \mathcal{A}(\mathsf{crs}), |x| = n_x] \le \mathsf{negl}(\lambda)$$

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

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

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

$$\Pr\left[\begin{array}{ccc} (\overline{\operatorname{crs}}, \operatorname{td}) \leftarrow \mathcal{E}(1^{\lambda}, 1^{n_{x}}, 1^{n_{w}}), \\ (\mathcal{R}(x, w) = 0 \ \lor \ |w| > n_{w}) \end{array} : \begin{array}{c} (\overline{\operatorname{crs}}, \operatorname{td}) \leftarrow \mathcal{E}(1^{\lambda}, 1^{n_{x}}, 1^{n_{w}}), \\ (x, \pi) \leftarrow \mathcal{A}(\overline{\operatorname{crs}}), \\ |x| = n_{x}, \\ w \leftarrow \mathcal{E}(\operatorname{td}, x, \pi) \end{array}\right] \le \operatorname{negl}(\lambda)$$

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

Remark 2.10 ([CW23, BWW23, BKP⁺23]). Assuming seBARGs there exists NIZKs.

Definition 2.11 (NIZK with strong extraction). A NIZK scheme (Setup, Prove, Verify) for language \mathcal{L} is said to be a NIZK scheme with strong extraction if it satisfies Definition 2.9 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 $negl(\cdot)$ such that for all $\lambda, n_x, n_w \in \mathbb{N}$:

$$\Pr\left[\begin{array}{ccc} (\overline{\operatorname{crs}}, \operatorname{td}) \leftarrow \mathcal{E}(1^{\lambda}, 1^{n_{x}}, 1^{n_{w}}), \\ (\mathcal{R}(x, w) = 0 \ \lor \ |w| > n_{w}) \end{array} : \begin{array}{c} (\overline{\operatorname{crs}}, \operatorname{td}) \leftarrow \mathcal{E}(1^{\lambda}, 1^{n_{x}}, 1^{n_{w}}), \\ (x, \pi) \leftarrow \mathcal{A}(\overline{\operatorname{crs}}, \operatorname{td}), \\ |x| = n_{x}, \\ w \leftarrow \mathcal{E}(\operatorname{td}, x, \pi) \end{array}\right] \le \operatorname{negl}(\lambda).$$

Note that this property implies straight-line extraction property from Definition 2.9.

Remark 2.12 (NIZK with strong extraction). Strong extraction is an additional property which can be achieved using a PKE scheme (Definition 2.16) in conjunction with soundness of NIZK scheme.

Definition 2.13 (Rate-1 NIZK). A NIZK scheme (Setup, Prove, Verify) for language \mathcal{L} is said to be a Rate-1 NIZK if it satisfies Definition 2.9 and the size of the proof π is $|w| + \text{poly}(\lambda, \log |x|, \log |w|)$.

2.5 Rate-1 Message Encoding

A rate-1 message encoding of a string $x \in \{0,1\}^*$ is an encoding ρ of the same length as x and secret information sk. However, for any PPT adversary without the sk, the encoding should look indistinguishable from a random string. In particular, we define a rate-1 message encoding scheme as follows:

Syntax. A rate-1 message encoding scheme (r1Enc) for any *x* consists of the following polynomial time algorithms:

- $\mathsf{Encode}(1^{\lambda}, x) \to (\rho, \mathsf{sk})$. The probabilistic encoding algorithm takes as input a security parameter λ , a string x, outputs the encoding ρ , and the secret information sk .
- $\mathsf{Decode}(1^{\lambda}, \rho, \mathsf{sk}) \to y$. The decoding algorithm takes as input the security parameter λ , encoded string ρ , the secret information sk , and outputs a string y.

Definition 2.14 (r1Enc). A rate-1 message encoding scheme (Encode, Decode) is required to satisfy the following properties:

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

Efficiency. The output size of $\mathsf{Encode}(\cdot, \cdot)$ is $|x| + \mathsf{poly}(\lambda)$ and the running time is $\mathsf{poly}(\lambda, |x|)$.

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

$$\left| \Pr \left[\begin{array}{ccc} 1 \leftarrow \mathcal{A}(\rho) & : & x \leftarrow \mathcal{A}(1^{\lambda}), \\ (\rho, \mathsf{sk}) \leftarrow \mathsf{Encode}(1^{\lambda}, x) \end{array} \right] - \Pr \left[\begin{array}{ccc} 1 \leftarrow \mathcal{A}(\rho) & : & x \leftarrow \mathcal{A}(1^{\lambda}), \\ \rho \leftarrow \{0, 1\}^{|x|} \end{array} \right] \right| \leq \mathsf{negl}(\lambda)$$

Remark 2.15. 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.

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

Definition 2.16 (PKE). A public-key encryption system (Setup, Enc, 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 $\operatorname{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$:

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

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

2.7 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 .
- $Digest(crs, cf) \rightarrow h$: 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 .
- Verify(crs, h_0, b, π) $\rightarrow \{0, 1\}$: 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 2.17 (RAM Delegation). A publicly verifiable non-interactive RAM delegation scheme (Setup, Digest, Prove, 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

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

Efficiency. In the completeness experiment above,

- Setup runs in time T_S .
- Digest on input cf runs in time $|cf| \cdot poly(\lambda)$ and outputs a digest of length λ .
- Prove runs in time $poly(\lambda, T, |cf|)$ and output a proof of length ℓ_{π} .
- Verify runs in time $\mathcal{O}(\ell_{\pi}) + \text{poly}(\lambda)$.
- Collision Resistance. For every PPT adversary \mathcal{A} and pair of polynomials $T = T(\lambda)$ there exists a negligible function $negl(\cdot)$ such that for every $\lambda \in \mathbb{N}$,

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

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

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

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

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

Remark 2.18 ([CJJ22b]). Assuming seBARGs and SEH there exists RAM delegation.

3 Defining (Reusable) Encrypted RAM Delegation

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

Recall that in RAM delegation a prover generates a proof π to convince a verifier that a RAM machine \mathcal{M} accepts an input x within T steps such that both the proof size and the verifier's running time only grow with $\operatorname{polylog}(T, |x|)$. In encrypted RAM delegation the input is partially private, namely, $x = (x_{pb}, x_{pr})$ for some private (resp. public) input x_{pr} (resp. x_{pb}). More specifically, consider RAM machine \mathcal{M} , and an input (x_{pb}, x_{pr}) where x_{pb} is a publicly known input, and x_{pr} is a private input. Then an encrypted RAM delegation comprises of a setup algorithm that generates a CRS (which is given to all the algorithms), a prover that given (x_{pb}, x_{pr}) , generates a proof π , and a verifier that, given only a short digest h_{pb} (of public input x_{pb}), verifies π .

Let $\mathcal{L}_{\mathcal{M},T}$ describe the language of the machine \mathcal{M} consisting of all $(x_{\mathsf{pb}}, x_{\mathsf{pr}})$ that are accepted by \mathcal{M} within T steps. We require our scheme to satisfy the following properties – (1) soundness: no PPT adversary can find an accepting (x_{pb}, π) s.t. for all $x_{\mathsf{pr}}, (x_{\mathsf{pb}}, x_{\mathsf{pr}}) \notin \mathcal{L}_{\mathcal{M},T}$, (2) straight-line extraction: there is an extractor \mathcal{E} such that for any accepting (x_{pb}, π) , it holds that $x_{\mathsf{pr}}^* \leftarrow \mathcal{E}(\mathsf{td}, \pi)$ (for some trapdoor td associated with crs) and $(x_{\mathsf{pb}}, x_{\mathsf{pr}}^*) \in \mathcal{L}_{\mathcal{M},T}$, (3) input hiding: there is a simulator that without knowing x_{pr} generates proofs that are indistinguishable from the output of the prover, and(4) efficiency: the proof size and verification time grow with $\mathsf{poly}(\log T, \log |x_{\mathsf{pb}}|, |x_{\mathsf{pr}}|)$. Note that if the proof doesn't grow with $\mathsf{poly}(|x_{\mathsf{pr}}|)$ then it will imply SNARGs for NP breaking the [GW11] barrier.

We further consider an additional *reusability* notion for the encrypted RAM delegation. This property allows one to generate an encoding eh_{pr} of the private input x_{pr} , with a long proof of the encoding π_{enc} , and then reuse the encoding to generate short proofs $\pi_{edel}^{(i)}$ for the computation on different public inputs $x_{pb}^{(i)}$. Note that the only reason that the proof size and the verification time grow polynomially with $|x_{pr}|$ is to prove the well-formedness of the private encoding eh_{pr} . Thus we let $|\pi_{enc}|$ to grow with $poly(|x_{pr}|)$, and then have $\pi_{edel}^{(i)}$ only grow with $polylog(|x_{pr}|)$. Moreover, we consider a stronger soundness notion for reusable encrypted RAM delegation where the adversary cannot generate accepting proofs $\pi_{edel}^{(0)}$ and $\pi_{edel}^{(1)}$ for a unique tuple (h_{pb}, eh_{pr}) such that $\pi_{edel}^{(b)}$ is a proof for $\mathcal{M}(x_{pb}, x_{pr}) = b$.

3.1 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.

- Prove(crs, x_{pb}, x_{pr}) $\rightarrow (b, \pi)$. 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 .
- Verify(crs, h_{pb}, b, π) $\rightarrow 0/1$. 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 3.1 (Encrypted RAM Delegation). An encrypted RAM delegation scheme (Setup, Prove, Verify) for \mathcal{M} associated with a hash tree HT with hash key ht.hk has to satisfy the following property:

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

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

- Efficiency. The crs and proof size, and the setup and the verifier running time are $poly(\lambda, \log T, \log n_{pb}, n_{pr})$, and the prover's running time is $poly(\lambda, T, n_{pb}, n_{pr})$.
- **Input Hiding.** There exists a stateful PPT simulator S such that for any PPT adversary A, there is a negligible function $negl(\cdot)$ such that for all $\lambda, n_{pb}, n_{pr}, S, T \in \mathbb{N}$:

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

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 $\lambda \in \mathbb{N}$:

$$\Pr\left[\begin{array}{ccc} (n_{\mathsf{pb}}, 1^{n_{\mathsf{pr}}}, S, T) \leftarrow \mathcal{A}(1^{\lambda}), \\ (n_{\mathsf{pb}}, 1^{n_{\mathsf{pr}}}, S, T) \leftarrow \mathcal{A}(1^{\lambda}), \\ (n_{\mathsf{pb}}, 1^{n_{\mathsf{pr}}}, S, T) \leftarrow \mathcal{A}(1^{\lambda}), \\ (n_{\mathsf{pb}}, 1^{n_{\mathsf{pr}}}, S, T), \\ (n_{\mathsf{pb}}, 1^{n_{\mathsf{pr}}}, S, T) \leftarrow \mathcal{A}(1^{\lambda}), \\ (n_{\mathsf{pb}}, 1^{n_{\mathsf{pr}}}, 1^{n_{\mathsf{pr}}}, S, T) \leftarrow \mathcal{A}(1^{\lambda}), \\ (n_{\mathsf{pb}}, 1^{n_{\mathsf{pr}}}, 1^{n_{\mathsf{pr}}}, 1^{n_{\mathsf{pr}}}, S, T) \leftarrow \mathcal{A}(1^{\lambda}), \\ (n_{\mathsf{pb}}, 1^{n_{\mathsf{pr}}}, 1^{n_{\mathsf{pr}}}, 1^{n_{\mathsf{pr}}}, 1^{n_{\mathsf{pr}}}, 1^{n_{\mathsf{pr}}}, S, T) \leftarrow \mathcal{A}(1^{\lambda}), \\ (n_{\mathsf{pb}}, 1^{n_{\mathsf{pr}}}, 1^{n_{\mathsf{pr}}}}, 1^{n_{\mathsf{pr}}}, 1^{n_{\mathsf{pr}}}, 1^{n_$$

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 $\lambda \in \mathbb{N}$:

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

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

3.2 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 .
- Prove(crs, x_{pb} , aux) $\rightarrow (b, \pi_{edel})$. 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} .
- VerifyEnc(crs, eh_{pr}, π_{enc}) $\rightarrow 0/1$. The encoding verification algorithm takes as input the common reference string crs, an encrypted digest eh_{pr} and a proof π_{enc} . It outputs 0 (reject) or 1 (accept).
- Verify(crs, h_{pb} , eh_{pr} , b, π_{edel}) $\rightarrow 0/1$. The verification algorithm takes as input the common reference string crs, a digest h_{pb} of the public input, an encrypted digest eh_{pr} of the private input, a bit b, and a proof π_{edel} . It outputs 0 (reject) or 1 (accept).

Definition 3.2 (Reusable Encrypted RAM Delegation). A reusable encrypted RAM delegation scheme (Setup, PrivEnc, Prove, VerifyEnc, Verify) for \mathcal{M} associated with a hash tree HT with hash key ht.hk has to satisfy the following property:

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

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

Efficiency. For the completeness experiment above,

- The size of the crs and setup running time are $poly(\lambda, \log T, \log n_{pb}, n_{pr})$.
- The size of π_{enc} , the private encoding and its verification time are $poly(\lambda, n_{pr})$.
- Prover's running time is $poly(\lambda, T, n_{pb}, n_{pr})$.
- The size of π_{edel} and the verification time are $\text{poly}(\lambda, \log T, \log n_{\text{pb}}, \log n_{\text{pr}})$.
- **Input Hiding.** There exists a stateful PPT simulator S such that for any PPT adversary A, there is a negligible function $negl(\cdot)$ such that for all $\lambda, n_{pb}, n_{pr}, S, T \in \mathbb{N}$:

$$\left|\Pr\left[1 \leftarrow \mathsf{Expt}_{0}^{\mathcal{C},\mathcal{A}}(1^{\lambda}, n_{\mathsf{pb}}, 1^{n_{\mathsf{pr}}}, S, T)\right] - \Pr\left[1 \leftarrow \mathsf{Expt}_{1}^{\mathcal{S},\mathcal{A}}(1^{\lambda}, n_{\mathsf{pb}}, 1^{n_{\mathsf{pr}}}, S, T)\right]\right| \le \mathsf{negl}(\lambda)$$

where definitions of $\mathsf{Expt}_0^{\mathcal{C},\mathcal{A}}$ and $\mathsf{Expt}_1^{\mathcal{S},\mathcal{A}}$ are provided in Figure 1.

 $\mathsf{Expt}_0^{\mathcal{C},\mathcal{A}}(1^\lambda, n_{\mathsf{pb}}, 1^{n_{\mathsf{pr}}}, S, T)$. 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: A sends x_{pr} and receives (eh_{pr}, π_{enc}, i) where (eh_{pr}, π_{enc}, aux_i) ← PrivEnc(crs, x_{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 $(b, \pi_{edel}) \leftarrow \mathsf{Prove}(\mathsf{crs}, x_{pb}, \mathsf{aux}_i)$.

 $\mathsf{Expt}_1^{\mathcal{S},\mathcal{A}}(1^\lambda, n_{\mathsf{pb}}, 1^{n_{\mathsf{pr}}}, S, T)$. 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 $(\mathsf{eh}_{pr}, \pi_{\mathsf{enc}}, i)$ where $(\mathsf{eh}_{pr}, \pi_{\mathsf{enc}}) \leftarrow \mathcal{S}(1^{|x_{pr}|}, i)$ and i is an index that after each query of this type is incremented.
- PROVE: \mathcal{A} sends (x_{pb}, i) , and if $(x_{pb}, x_{pr}) \in \mathcal{L}_{\mathcal{M},T}$ (for x_{pr} corresponding to *i*) receives $\pi_{edel} \leftarrow \mathcal{S}(x_{pb}, i)$, otherwise, receives \perp .

Figure 1: Real and ideal experiments for Input Hiding property of reusable encrypted RAM delegation.

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

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

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

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

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

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 $\lambda, n_x \in \mathbb{N}$:

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

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

4 Constructing (Reusable) Encrypted RAM Delegation

In this section we construct encrypted RAM delegation and reusable encrypted RAM delegation. Our construction in a nutshell works as follows – First compute a short digest h_{pr} of x_{pr} , encrypt h_{pr} using randomness r to get the private encoding eh_{pr} , and generate a NIZK proof of the validity of eh_{pr} using (x_{pr}, h_{pr}, r) as witness. Then let $cf_0 = (x_{pb}, x_{pr}, 0^S)$ be the initial configuration of \mathcal{M} and compute its partial digests h_{pb} (of x_{pb}) and h_{Σ} (of 0^S). Finally compute a delegation proof del. π for $\mathcal{M}(cf_0) = b$ and generate a NIZK proof for the verification of the delegation for a partially hiding digest $(h_{pb}, eh_{pr}, h_{\Sigma})$ (of cf_0) using $(h_{pr}, r, del.\pi)$ as witness. Finally let the final proof be $\pi = (eh_{pr}, zk.\pi_{enc}, zk.\pi_{edel})$ where one can verify $(eh_{pr}, zk.\pi_{enc})$ once and reuse eh_{pr} across different delegation proofs $zk.\pi_{edel}$. In what follows we present our formal constructions and their analysis.

4.1 Encrypted RAM Delegation

Language \mathcal{L}_{enc}	Language \mathcal{L}_{edel}
Hardwired: ht.hk, pke.pk	Hardwired: ht.hk, pke.pk
Instance: $x_{enc} = eh_{pr}$	Instance: $x_{edel} = (h_{pb}, eh_{pr}, h_{\Sigma}, b)$
Witness: $w_{enc} = (x_{pr}, h_{pr}, r)$	Witness: $w_{edel} = (h_{pr}, r, del.\pi)$
Output: $(x_{enc}, w_{enc}) \in \mathcal{L}_{enc}$ if the following hold:	Output: $(x_{edel}, w_{edel}) \in \mathcal{L}_{edel}$ if the following hold:
$\begin{array}{ll} - & h_{pr} = HT.Hash(ht.hk, x_{pr}). \\ - & eh_{pr} = PKE.Enc(pke.pk, h_{pr}; r) \end{array}$	$ \begin{array}{ll} - & eh_{pr} = PKE.Enc(pke.pk,h_{pr};r). \\ - & Del.Verify(del.crs,(h_{pb},h_{pr},h_{\Sigma}),b,del.\pi) = 1 \end{array} $

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

Construction 4.1. [Encrypted RAM Delegation] Let PKE = (PKE.Gen, PKE.Enc, PKE.Dec) be a public-key encryption scheme, NIZK = (NIZK.Setup, NIZK.Prove, NIZK.Verify) be a non-interactive zero-knowledge scheme, and Del = (Del.Setup, Del.Digest, Del.Prove, Del.Verify) be a RAM delegation scheme associated with a hash tree HT with hash key ht.hk. We construct our encrypted RAM delegation scheme $\mathcal{E}Del = (Setup, Prove, Verify)$ as follows:

 $\begin{aligned} \mathsf{Setup}(1^{\lambda}, n_{\mathsf{pb}}, 1^{n_{\mathsf{pr}}}, S, T) &\to \mathsf{crs.} \text{ Sample a RAM delegation scheme del.crs} \leftarrow \mathsf{Del.Setup}(1^{\lambda}, T, n_{\mathsf{pb}}, n_{\mathsf{pr}}, S), \\ & a \, \mathsf{PKE} \, \mathsf{scheme} \, (\mathsf{pke.pk}, \mathsf{pke.sk}) \leftarrow \mathsf{PKE.Gen}(1^{\lambda}), \, a \, \mathsf{NIZK} \, \mathsf{scheme} \, \mathsf{for} \, \mathcal{L}_{\mathsf{enc}} \, \mathsf{as} \, (\mathsf{zk.crs}_{\mathsf{enc}}, \mathsf{zk.td}_{\mathsf{enc}}) \leftarrow \\ & \mathsf{NIZK.Setup}(1^{\lambda}, 1^{n_{\mathsf{enc}}}, 1^{n'_{\mathsf{enc}}}), \, \mathsf{and} \, \mathsf{finally} \, \mathsf{another} \, \mathsf{NIZK} \, \mathsf{scheme} \, \mathsf{for} \, \mathcal{L}_{\mathsf{edel}} \, \mathsf{as} \, (\mathsf{zk.crs}_{\mathsf{edel}}, \mathsf{zk.td}_{\mathsf{edel}}) \leftarrow \\ & \mathsf{NIZK.Setup}(1^{\lambda}, 1^{n_{\mathsf{edel}}}, 1^{n'_{\mathsf{edel}}}). \, \, \mathsf{Let} \, \, \mathsf{crs} = (\mathsf{del.crs}, \mathsf{pke.pk}, \mathsf{zk.crs}_{\mathsf{enc}}, \mathsf{zk.crs}_{\mathsf{edel}}, S). \end{aligned}$

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

- 1. Parse $crs = (del.crs, pke.pk, zk.crs_{enc}, zk.crs_{edel}, S)$.
- 2. Compute a hash digest of the private input $h_{pr} \leftarrow HT.Hash(ht.hk, x_{pr})$, sample a randomness r, and compute an encryption of h_{pr} as $eh_{pr} \leftarrow PKE.Enc(pke.pk, h_{pr}; r)$.
- 3. Compute a NIZK proof for validity of eh_{pr} as $zk.\pi_{enc} \leftarrow NIZK.Prove(zk.crs_{enc}, eh_{pr}, (x_{pr}, h_{pr}, r))$.
- 4. Let $cf_0 = (x_{pb}, x_{pr}, 0^S)$ be the initial configuration of the RAM machine and compute its partial digests $h_{pb} \leftarrow HT.Hash(ht.hk, x_{pb})$ and $h_{\Sigma} \leftarrow HT.Hash(ht.hk, 0^S)$.
- 5. Compute a delegation proof $(b, del.\pi) \leftarrow Del.Prove(del.crs, cf_0)$.
- 6. Compute a NIZK proof $\mathsf{zk}.\pi_{\mathsf{edel}} \leftarrow \mathsf{NIZK}.\mathsf{Prove}(\mathsf{zk}.\mathsf{crs}_{\mathsf{edel}},(\mathsf{h}_{\mathsf{pb}},\mathsf{eh}_{\mathsf{pr}},\mathsf{h}_{\Sigma},b),(\mathsf{h}_{\mathsf{pr}},r,\mathsf{del}.\pi)).$
- 7. Output $(b, \pi = (\mathsf{eh}_{\mathsf{pr}}, \mathsf{zk}.\pi_{\mathsf{enc}}, \mathsf{zk}.\pi_{\mathsf{edel}})).$
- Verify(crs, h_{pb}, b, π) $\rightarrow 0/1$. First parse crs = (del.crs, pke.pk, zk.crs_{enc}, zk.crs_{edel}, S), and then compute $h_{\Sigma} \leftarrow HT.Hash(ht.hk, 0^S)$. Output 1 if NIZK.Verify(zk.crs_{enc}, eh_{pr}, zk. π_{enc}) = 1 and NIZK.Verify(zk.crs_{enc}, (h_{pb}, eh_{pr}, h_{\Sigma}, b), zk. π_{edel}) = 1; otherwise Output 0.

Theorem 4.2. Assuming RAM delegation, PKEs, and NIZKs, the Construction 4.1 is an encrypted RAM delegation (Definition 3.1) with strong soundness, straight-line extractor, and input hiding.

Corollary 4.3. Assuming either LWE, k-LIN over pairing groups for any constant $k \in \mathbb{N}$, or subexponential DDH over pairing-free groups, the Construction 4.1 is an encrypted RAM delegation (Definition 3.1) with straight-line extractor, and input hiding. Additionally, if we plug-in our rate-1 NIZK from Section 5, the crs size and the setup running-time will only grow with log n_{pr} .

Remark 4.4. 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.

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

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

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

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 $poly(\lambda, \log T, \log n_{pb}, \log n_{pr}) + poly(\lambda, n_{pr})$ and the generation time is $poly(\lambda, T, n_{pb}, n_{pr})$

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

4.1.1 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 (crs, td) proceed the same as normal setup except that use NIZK. \mathcal{E}_1 to sample (zk.crs_{enc}, zk.td_{enc}) and (zk.crs_{edel}, zk.td_{edel}), then construct crs similar to setup and let td = (pke.sk, zk.td_{enc}, zk.td_{edel}), (2) upon receiving $\pi = (eh_{pr}, zk.\pi_{enc}, zk.\pi_{edel})$ extract $(x_{pr}^*, h_{pr,1}^*, r_1^*) = NIZK.\mathcal{E}_1(zk.td_{enc}, eh_{pr}, zk.\pi_{enc})$ 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 $\operatorname{Exp}_{\mathcal{E}\mathsf{Del}}$ 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 (crs,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 $\mathsf{h}_{\mathsf{pb}} = \mathsf{HT}.\mathsf{Hash}(\mathsf{ht.hk}, x_{\mathsf{pb}})$, and $\mathsf{h}_{\Sigma} \leftarrow \mathsf{HT}.\mathsf{Hash}(\mathsf{ht.hk}, 0^S)$. 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:

$$\mathsf{Exp}_{\mathcal{E}\mathsf{Del}} = \begin{bmatrix} (n_{\mathsf{pb}}, 1^{n_{\mathsf{pr}}}, S, T) \leftarrow \mathcal{A}(1^{\lambda}), \\ (\mathsf{crs}, \mathsf{td}) \leftarrow \mathcal{E}(1^{\lambda}, n_{\mathsf{pb}}, 1^{n_{\mathsf{pr}}}, S, T), \\ (x_{\mathsf{pb}}, b, \pi) \leftarrow \mathcal{A}(\mathsf{crs}), \\ \mathsf{h}_{\mathsf{pb}} = \mathsf{HT}.\mathsf{Hash}(\mathsf{ht.hk}, x_{\mathsf{pb}}), \\ \mathsf{h}_{\Sigma} \leftarrow \mathsf{HT}.\mathsf{Hash}(\mathsf{ht.hk}, 0^{S}), \\ \mathsf{Let} \ \pi = (\mathsf{eh}_{\mathsf{pr}}, \mathsf{zk}.\pi_{\mathsf{enc}}, \mathsf{zk}.\pi_{\mathsf{edel}}), \\ (x_{\mathsf{pr}}^*, \mathsf{h}_{\mathsf{pr},1}^*, r_1^*) \leftarrow \mathsf{NIZK}.\mathcal{E}_1(\mathsf{zk.td}_{\mathsf{enc}}, \mathsf{eh}_{\mathsf{pr}}, \mathsf{zk}.\pi_{\mathsf{enc}}), \\ (\mathsf{h}_{\mathsf{pr},2}^*, r_2^*, \mathsf{del}.\pi^*) = \mathsf{NIZK}.\mathcal{E}_2(\mathsf{zk.td}_{\mathsf{edel}}, (\mathsf{h}_{\mathsf{pb}}, \mathsf{eh}_{\mathsf{pr}}, \mathsf{h}_{\Sigma}, b), \mathsf{zk}.\pi_{\mathsf{edel}}) \end{bmatrix}$$

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 NIZK.Verify(zk.crs_{enc}, eh_{pr}, zk. π_{enc}) = 1 and NIZK.Verify(zk.crs_{edel}, (h_{pb}, eh_{pr}, h_{Σ}, b), zk. π_{edel}) = 1 and $(x_{pb}, x_{pr}^*) \notin \mathcal{L}_{\mathcal{M},T}$. Namely \mathcal{A} wins if:

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

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

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

$$\epsilon_{2} = \Pr\left[\begin{array}{cc} (x_{\mathsf{pb}}, x_{\mathsf{pr}}^{*}) \notin \mathcal{L}_{\mathcal{M},T} \land (x_{\mathsf{enc}}, w_{\mathsf{enc}}) \in \mathcal{L}_{\mathsf{enc}} \land \\ \mathsf{NIZK.Verify}(\mathsf{zk.crs}_{\mathsf{edel}}, x_{\mathsf{edel}}, \mathsf{zk}, \pi_{\mathsf{edel}}) = 1 \end{array} : \mathsf{Exp}_{\mathcal{E}\mathsf{Del}} \right] - \\ \Pr\left[\begin{array}{cc} (x_{\mathsf{pb}}, x_{\mathsf{pr}}^{*}) \notin \mathcal{L}_{\mathcal{M},T} \land (x_{\mathsf{enc}}, w_{\mathsf{enc}}) \in \mathcal{L}_{\mathsf{enc}} \land (x_{\mathsf{edel}}, w_{\mathsf{edel}}) \in \mathcal{L}_{\mathsf{edel}} \end{array} : \mathsf{Exp}_{\mathcal{E}\mathsf{Del}} \right].$$
(4)

$$\epsilon_{3} = \Pr\left[\begin{array}{ccc} (x_{\mathsf{pb}}, x_{\mathsf{pr}}^{*}) \notin \mathcal{L}_{\mathcal{M},T} \land (x_{\mathsf{enc}}, w_{\mathsf{enc}}) \in \mathcal{L}_{\mathsf{enc}} \land (x_{\mathsf{edel}}, w_{\mathsf{edel}}) \in \mathcal{L}_{\mathsf{edel}} & : \quad \mathsf{Exp}_{\mathcal{E}\mathsf{Del}} \end{array}\right] - \\ \Pr\left[\begin{array}{ccc} (x_{\mathsf{pb}}, x_{\mathsf{pr}}^{*}) \notin \mathcal{L}_{\mathcal{M},T} \land \mathsf{h}_{\mathsf{pr},2}^{*} = \mathsf{HT}.\mathsf{Hash}(\mathsf{ht.hk}, x_{\mathsf{pr}}^{*}) \\ \mathsf{Del}.\mathsf{Verify}(\mathsf{del.crs}, (\mathsf{h}_{\mathsf{pb}}, \mathsf{h}_{\mathsf{pr},2}^{*}, \mathsf{h}_{\Sigma}), b, \mathsf{del}.\pi) = 1 \end{array} \right] \cdot \left[\begin{array}{c} \mathsf{Exp}_{\mathcal{E}\mathsf{Del}} \end{array}\right]. \tag{5}$$

$$\epsilon_{4} = \Pr\left[\begin{array}{cc} (x_{\mathsf{pb}}, x_{\mathsf{pr}}^{*}) \notin \mathcal{L}_{\mathcal{M},T} \land h_{\mathsf{pr},2}^{*} = \mathsf{HT}.\mathsf{Hash}(\mathsf{ht.hk}, x_{\mathsf{pr}}^{*}) \\ \mathsf{Del}.\mathsf{Verify}(\mathsf{del.crs}, (\mathsf{h}_{\mathsf{pb}}, \mathsf{h}_{\mathsf{pr},2}^{*}, \mathsf{h}_{\Sigma}), b, \mathsf{del}.\pi) = 1 \end{array} : \mathsf{Exp}_{\mathcal{E}\mathsf{Del}} \right].$$
(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:

$$\Pr[\mathsf{NIZK}.\mathsf{Verify}(\mathsf{zk}.\mathsf{crs}_{\mathsf{enc}},\mathsf{eh}_{\mathsf{pr}},\mathsf{zk}.\pi_{\mathsf{enc}}) = 1 \land (x_{\mathsf{enc}},w_{\mathsf{enc}}) \notin \mathcal{L}_{\mathsf{enc}}] \ge \epsilon_1$$

Now we construct adversary $\mathcal{B}_{zk}^{\mathcal{E}}$ 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 $\text{Exp}_{\mathcal{E}\text{Del}}$ where $zk.crs_{enc} = zk.crs_{enc}^*$, and send it to \mathcal{A} .
- 4. Receive (x_{pb}, b, π) from \mathcal{A} .
- 5. Parse $\pi = (\mathsf{eh}_{\mathsf{pr}}, \mathsf{zk}.\pi_{\mathsf{enc}}, \mathsf{zk}.\pi_{\mathsf{edel}}).$
- 6. Send $(eh_{pr}, zk.\pi_{enc})$ to C.

Note that $\mathcal{B}_{\mathsf{zk}}^{\mathcal{E}}$ 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:

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

Now we construct adversary $\mathcal{B}_{zk}^{\mathcal{E}}$ 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 $\mathsf{zk}.\mathsf{crs}_{\mathsf{edel}} = \mathsf{zk}.\mathsf{crs}_{\mathsf{edel}}^*$, and send it to \mathcal{A} .
- 4. Receive $(x_{\mathsf{pb}}, b, \pi)$ from \mathcal{A} .
- 5. Parse $\pi = (eh_{pr}, zk.\pi_{enc}, zk.\pi_{edel})$.
- 6. Compute $h_{pb} = HT.Hash(ht.hk, x_{pb})$ and $h_{\Sigma} \leftarrow HT.Hash(ht.hk, 0^S)$.
- 7. Send $((\mathsf{h}_{\mathsf{pb}},\mathsf{eh}_{\mathsf{pr}},\mathsf{h}_{\Sigma},b),\mathsf{zk}.\pi_{\mathsf{edel}})$ to \mathcal{C} .

Note that $\mathcal{B}_{zk}^{\mathcal{E}}$ 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 $(x_{enc}, w_{enc}) \in \mathcal{L}_{enc}$ is equivalent with:

$$\mathsf{eh}_{\mathsf{pr}} = \mathsf{PKE}.\mathsf{Enc}(\mathsf{pke}.\mathsf{pk},\mathsf{h}^*_{\mathsf{pr},1};r_1^*) \land \mathsf{h}^*_{\mathsf{pr},1} = \mathsf{HT}.\mathsf{Hash}(\mathsf{ht}.\mathsf{hk},x^*_{\mathsf{pr}})$$

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

$$\mathsf{eh}_{\mathsf{pr}} = \mathsf{PKE}.\mathsf{Enc}(\mathsf{pke}.\mathsf{pk},\mathsf{h}^*_{\mathsf{pr},2};r_2^*) \land \mathsf{Del}.\mathsf{Verify}(\mathsf{del}.\mathsf{crs},(\mathsf{h}_{\mathsf{pb}},\mathsf{h}^*_{\mathsf{pr},2},\mathsf{h}_\Sigma),b,\mathsf{del}.\pi) = 1$$

then by Eq. (5) it holds that

$$\Pr \begin{bmatrix} \mathsf{eh}_{\mathsf{pr}} = \mathsf{PKE}.\mathsf{Enc}(\mathsf{pke}.\mathsf{pk},\mathsf{h}_{\mathsf{pr},1}^*;r_1^*) \land \mathsf{h}_{\mathsf{pr},1}^* = \mathsf{HT}.\mathsf{Hash}(\mathsf{ht}.\mathsf{hk},x_{\mathsf{pr}}^*) \land \\ \mathsf{eh}_{\mathsf{pr}} = \mathsf{PKE}.\mathsf{Enc}(\mathsf{pke}.\mathsf{pk},\mathsf{h}_{\mathsf{pr},2}^*;r_2^*) \land \mathsf{h}_{\mathsf{pr},2}^* \neq \mathsf{HT}.\mathsf{Hash}(\mathsf{ht}.\mathsf{hk},x_{\mathsf{pr}}^*) & : \quad \mathsf{Exp}_{\mathcal{E}\mathsf{Del}} \end{bmatrix} \geq \epsilon_3$$

Note that this implies that $h_{pr,1}^* \neq h_{pr,2}^*$, however, PKE.Enc(pke.pk, $h_{pr,1}^*; r_1^*$) = PKE.Enc(pke.pk, $h_{pr,2}^*; r_2^*$). 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 $\mathcal{B}_{del}^{sound}$ 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 del.crs^{*} from \mathcal{C} .
- 3. Construct crs similar to experiment $\mathsf{Exp}_{\mathcal{E}\mathsf{Del}}$ where del.crs = del.crs^{*}, and send it to \mathcal{A} .
- 4. Receive (x_{pb}, b, π) from \mathcal{A} .
- 5. Parse $\pi = (\mathsf{eh}_{\mathsf{pr}}, \mathsf{zk}.\pi_{\mathsf{enc}}, \mathsf{zk}.\pi_{\mathsf{edel}}).$
- 6. Compute $h_{pb} = HT.Hash(ht.hk, x_{pb})$ and $h_{\Sigma} \leftarrow HT.Hash(ht.hk, 0^S)$.
- 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 $(cf_0 = (x_{pb}, x_{pr}^*, 0^S), b, del.\pi^*)$ to C.

Note that $(h_{pb}, h_{pr,2}^*, h_{\Sigma}) = \text{Del.Digest}(cf_0), (x_{pb}, x_{pr}^*) \notin \mathcal{L}_{\mathcal{M},T}$ implies that $cf_0 \notin \mathcal{L}_{\mathcal{M},T}$, and $\mathcal{B}_{del}^{\text{sound}}$ perfectly simulates the experiment $\text{Exp}_{\mathcal{E}\text{Del}}$ for \mathcal{A} . Thus $\mathcal{B}_{del}^{\text{sound}}$ 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 $zk.crs_{enc}$ and $zk.crs_{edel}$ using $zk.crs_{enc} \leftarrow NIZK.S_1(1^{\lambda}, 1^{n_{enc}})$ and $zk.crs_{edel} \leftarrow NIZK.S_2(1^{\lambda}, 1^{n_{edel}})$, and (2) generate the proof by computing $eh_{pr} \leftarrow PKE.Enc(pke.pk, 0^{|h_{pr}|})$, $zk.\pi_{enc} \leftarrow NIZK.S_1(eh_{pr})$, $h_{pb} \leftarrow HT.Hash(ht.hk, x_{pb})$, $h_{\Sigma} \leftarrow HT.Hash(ht.hk, 0^S)$, and $zk.\pi_{edel} \leftarrow NIZK.S_2(h_{pb}, eh_{pr}, h_{\Sigma}, b)$ and letting $\pi = (eh_{pr}, zk.\pi_{enc}, zk.\pi_{edel})$.

We will define the following hybrids:

- hyb₀. This is the real experiment where the \mathcal{A} receives (crs, td) \leftarrow Setup $(1^{\lambda}, n_{pb}, 1^{n_{pr}}, S, T)$, then makes queries (x_{pb}, x_{pr}) and receives $(b, \pi) \leftarrow \mathsf{Prove}(\mathsf{crs}, x_{pb}, x_{pr})$.
- hyb₁. Similar to hyb₀ 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 $\mathsf{zk}.\pi_{\mathsf{enc}} \leftarrow \mathsf{NIZK}.\mathcal{S}_1(\mathsf{eh}_{\mathsf{pr}})$.
- hyb₂. Similar to hyb₁ 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 $\mathsf{zk}.\pi_{\mathsf{edel}} \leftarrow \mathsf{NIZK}.\mathcal{S}_2(\mathsf{h}_{\mathsf{pb}}, \mathsf{eh}_{\mathsf{pr}}, \mathsf{h}_{\Sigma}, b)$.
- hyb₃. Similar to hyb₂ except that $eh_{pr} = \mathsf{PKE}.\mathsf{Enc}(\mathsf{pke}.\mathsf{pk}, 0^{|h_{pr}|})$. 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 4.5. If NIZK is zero-knowledge, then for any adversary \mathcal{A} it holds that

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

Proof. Suppose towards the contradiction that for some adversary \mathcal{A} and some non-negligible function $\epsilon(\cdot)$ it holds that $|\Pr[\mathsf{hyb}_0(\mathcal{A}) = 1] - \Pr[\mathsf{hyb}_1(\mathcal{A}) = 1]| \ge \epsilon(\lambda)$. 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_{pb}, x_{pr}) generate the proof similar to $\mathsf{Prove}(\mathsf{crs}, x_{pb}, x_{pr})$ 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 NIZK.Setup and NIZK.Prove (resp. S_1) in the experiment then \mathcal{B}_{zk} perfectly simulates hyb_0 (resp. hyb_1). Thus \mathcal{B}_{zk} has the same advantage $\epsilon(\lambda)$ as \mathcal{A} in breaking the zero-knowledge property of the NIZK scheme.

Lemma 4.6. If NIZK is zero-knowledge, then for any adversary \mathcal{A} it holds that

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

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

- 1. Receive $(1^{\lambda}, n_{pb}, 1^{n_{pr}}, S, T)$ 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_{pb}, x_{pr}) generate the proof similar to hyb_1 except that to generate $zk.\pi_{edel}$, make a query to C on $((h_{pb}, eh_{pr}, h_{\Sigma}, b), (x_{pr}, h_{pr}, r))$.
- 5. Send π to \mathcal{A} and output whatever \mathcal{A} outputs.

Note that if the challenger uses NIZK.Setup and NIZK.Prove (resp. S_2) in the experiment then \mathcal{B}_{zk} perfectly simulates hyb_1 (resp. hyb_2). Thus \mathcal{B}_{zk} has the same advantage $\epsilon(\lambda)$ as \mathcal{A} in breaking the zero-knowledge property of the NIZK scheme.

Lemma 4.7. If PKE is secure, then for any adversary \mathcal{A} it holds that

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

Proof. Suppose towards the contradiction that for some adversary \mathcal{A} and some non-negligible function $\epsilon(\cdot)$ it holds that $|\Pr[\mathsf{hyb}_2(\mathcal{A}) = 1] - \Pr[\mathsf{hyb}_3(\mathcal{A}) = 1]| \ge \epsilon(\lambda)$. 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_{pb}, x_{pr}) compute $h_{pr} \leftarrow HT.Hash(ht.hk, x_{pr})$ send a query to C on h_{pr} and receive eh_{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 $\mathsf{PKE}.\mathsf{Enc}(\mathsf{pke}.\mathsf{pk}, \cdot)$ (resp. $\mathsf{PKE}.\mathsf{Enc}(\mathsf{pke}.\mathsf{pk}, 0^{|\mathsf{h}_{\mathsf{pr}}|})$) 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 4.5 to 4.7 and a hybrid argument.

4.2 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 4.1 to match the syntax of reusable encrypted RAM delegation.

Construction 4.8. [Encrypted RAM Delegation] Let PKE = (PKE.Gen, PKE.Enc, PKE.Dec) be a public-key encryption scheme, Del = (Del.Setup, Del.Digest, Del.Prove, Del.Verify) be a RAM delegation scheme associated with a hash tree HT, and NIZK = (NIZK.Setup, NIZK.Prove, NIZK.Verify) be a non-interactive zero-knowledge scheme. We construct an encrypted RAM delegation $\mathcal{E}Del = (Setup, Prove, Verify)$ as follows:

 $\mathsf{Setup}(1^{\lambda}, n_{\mathsf{pb}}, 1^{n_{\mathsf{pr}}}, S, T) \to (\mathsf{crs}).$ Same as in Construction 4.1.

- $\mathsf{PrivEnc}(\mathsf{crs}, x_{\mathsf{pr}}) \to (\mathsf{eh}_{\mathsf{pr}}, \pi_{\mathsf{enc}}, \mathsf{aux}).$ Perform Items 1 to 3 from the prover's algorithm in Construction 4.1 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 4.1 except that it takes $\mathsf{aux} = (x_{\mathsf{pr}}, r)$ as input and doesn't compute $\mathsf{zk}.\pi_{\mathsf{enc}}$.
- VerifyEnc(crs, eh_{pr}, π_{enc}) $\rightarrow 0/1$. Parse crs = (del.crs, pke.pk, zk.crs_{enc}, zk.crs_{edel}, S), and output 1 if it holds that NIZK.Verify(zk.crs_{enc}, eh_{pr}, π_{enc}) = 1; otherwise Output 0.
- Verify(crs, h_{pb} , eh_{pr} , b, π_{edel}) $\rightarrow 0/1$. First parse crs = (del.crs, pke.pk, zk.crs_{enc}, zk.crs_{edel}, S), and then let $h_{\Sigma} \leftarrow HT.Hash(ht.hk, 0^S)$. Output 1 if NIZK.Verify(zk.crs_{enc}, (h_{pb} , eh_{pr} , h_{Σ} , b), π_{edel}) = 1; otherwise Output 0.

Theorem 4.9. Assuming RAM delegation, PKEs, and NIZKs, the Construction 4.8 is a reusable encrypted RAM delegation (Definition 3.2) 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 4.1 for details.

4.2.1 Security Analysis.

Proof of Straight-Line Extractor. The proof is similar to the proof as in the encrypted RAM delegation. Refer to Section 4.1.1 for details. \Box

Proof of Input Hiding. The proof is similar to the proof as in the encrypted RAM delegation. Refer to Section 4.1.1 for details. \Box

Proof of Strong Soundness. First define the following hybrids:

- hyb₀. 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 $\mathsf{crs}, \mathcal{A}$ outputs $(x_{\mathsf{pb}}, \mathsf{eh}_{\mathsf{pr}}, \pi_{\mathsf{edel}}^{(0)}, \pi_{\mathsf{edel}}^{(1)})$. \mathcal{A} wins if for $b \in \{0, 1\}$, $\mathsf{Verify}(\mathsf{zk.crs}_{\mathsf{edel}}, \mathsf{h}_{\mathsf{pb}}, \mathsf{eh}_{\mathsf{pr}}, b, \pi_{\mathsf{edel}}^{(b)}) = 1$.
- hyb_1 . Similar to hyb_0 except that C generates $zk.crs_{edel}$ in the crs by running NIZK. \mathcal{E} and appends $zk.td_{edel}$ 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 $Exp_{\mathcal{E}Del}$ be the same as in hyb_1 except that \mathcal{C} additionally computes h_{Σ} and extracts the witness from NIZK proofs as follows:

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

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:

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

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:

$$\epsilon_{1} = \Pr[\mathsf{Exp}_{\mathcal{E}\mathsf{Del}}(\mathcal{A}) = 1] - \Pr\left[\begin{array}{cc} (x_{\mathsf{edel}}^{(0)}, w_{\mathsf{edel}}^{(0)}) \in \mathcal{L}_{\mathsf{edel}} \\ \mathsf{NIZK}.\mathsf{Verify}(\mathsf{zk}.\mathsf{crs}_{\mathsf{edel}}, (\mathsf{h}_{\mathsf{pb}}, \mathsf{eh}_{\mathsf{pr}}, \mathsf{h}_{\Sigma}, 1), \pi_{\mathsf{edel}}^{(1)}) = 1 \end{array} : \mathsf{Exp}_{\mathcal{E}\mathsf{Del}} \right].$$
(7)

$$\epsilon_{2} = \Pr \left[\begin{array}{cc} (x_{\text{edel}}^{(0)}, w_{\text{edel}}^{(0)}) \in \mathcal{L}_{\text{edel}} \\ \mathsf{NIZK.Verify}(\mathsf{zk.crs}_{\text{edel}}, (\mathsf{h}_{\mathsf{pb}}, \mathsf{eh}_{\mathsf{pr}}, \mathsf{h}_{\Sigma}, 1), \pi_{\text{edel}}^{(1)}) = 1 \end{array} \right] - \\ \Pr \left[\begin{array}{cc} (x_{\text{edel}}^{(b)}, w_{\text{edel}}^{(b)}) \in \mathcal{L}_{\text{edel}} \text{ for } b \in \{0, 1\} \end{array} \right] \cdot \\ \mathsf{Exp}_{\mathcal{E}\mathsf{Del}} \left] \right]. \tag{8}$$

$$\epsilon_{3} = \Pr\left[\begin{array}{cc} (x_{\mathsf{edel}}^{(b)}, w_{\mathsf{edel}}^{(b)}) \in \mathcal{L}_{\mathsf{edel}} \text{ for } b \in \{0, 1\} & : \quad \mathsf{Exp}_{\mathcal{E}\mathsf{Del}} \end{array}\right] - \\ \Pr\left[\begin{array}{cc} \mathsf{h}_{\mathsf{pr}}^{(0)} = \mathsf{h}_{\mathsf{pr}}^{(1)} \\ \mathsf{Del}.\mathsf{Verify}(\mathsf{del.crs}, (\mathsf{h}_{\mathsf{pb}}, \mathsf{h}_{\mathsf{pr}}^{(b)}, \mathsf{h}_{\Sigma}), b, \mathsf{del}.\pi^{(b)}) = 1 \text{ for } b \in \{0, 1\} \end{array}\right] : \quad \mathsf{Exp}_{\mathcal{E}\mathsf{Del}} \left]. \tag{9}$$

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

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

Now we construct adversary $\mathcal{B}_{zk}^{\mathcal{E}}$ 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 $Exp_{\mathcal{E}Del}$ where $zk.crs_{edel} = zk.crs_{edel}^*$, 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 $h_{\Sigma} = HT.Hash(ht.hk, 0^S)$.
- 6. Send $((\mathsf{h}_{\mathsf{pb}},\mathsf{eh}_{\mathsf{pr}},\mathsf{h}_{\Sigma},0),\pi^{(0)}_{\mathsf{edel}})$ to $\mathcal{C}.$

Note that $\mathcal{B}_{zk}^{\mathcal{E}}$ 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 $(x_{\text{edel}}^{(b)}, w_{\text{edel}}^{(b)}) \in \mathcal{L}_{\text{edel}}$ is equivalent with:

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

then by Eq. (9) it holds that

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

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 $\mathcal{B}_{del}^{\text{sound}}$ 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 del.crs^{*} from \mathcal{C} .
- 3. Construct crs similar to experiment $\mathsf{Exp}_{\mathcal{E}\mathsf{Del}}$ where del.crs = 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 $(\mathsf{h}_{\mathsf{pr}}^{(b)}, r^{(b)}, \mathsf{del}.\pi^{(b)}) = \mathsf{NIZK}.\mathcal{E}(\mathsf{zk}.\mathsf{td}_{\mathsf{edel}}, (\mathsf{h}_{\mathsf{pb}}, \mathsf{eh}_{\mathsf{pr}}, \mathsf{h}_{\Sigma}, b), \pi^{(b)}_{\mathsf{edel}})$ for $b \in \{0, 1\}$.
- 6. Compute $h_{\Sigma} = HT.Hash(ht.hk, 0^S)$.
- 7. Send $((h_{pb}, h_{pr}^{(0)}, h_{\Sigma}), \pi_{edel}^{(0)}, \pi_{edel}^{(1)})$ to C.

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

Corollary 4.10. Assuming either LWE, k-LIN over pairing groups for any constant $k \in \mathbb{N}$, or sub-exponential DDH over pairing-free groups, the Construction 4.8 is a reusable encrypted RAM delegation (Definition 3.2) with strong soundness, straight-line extractor, and input hiding. Additionally, if we plug-in our rate-1 NIZK from Section 5, the crs size and the setup running-time will only grow with $\log n_{\rm pr}$.

5 Rate-1 NIZK

In this section, we present a construction for a rate-1 NIZK argument scheme with straight-line extraction for any language $\mathcal{L}_{\mathcal{M},T}$ using an encrypted RAM delegation scheme. The following observation serves as the main design principle behind our rate-1 NIZK.

Rate-1 NIZK from Instance-Independent NIZK. Consider a NIZK scheme for a language $\mathcal{L} = \{x : \exists w, \mathcal{M}(x, w) = 1\}$ in which the size of the proof is independent⁷ on the size of the instance x and grows polynomially in the size of the witness w. If such an "instance-independent" NIZK scheme exists, we can generate a Rate-1 NIZK scheme by relying on PRG schemes. The idea is to use the instance-independent NIZK scheme to generate proof for the modified instance-witness pair $((x, \mathsf{PRG}(s) \oplus w), s)$. That is, we push a masked version of the witness to be a part of the instance and use the seed s for the PRG as a witness. By assuming the computational soundness of instance-independent NIZK and security of PRG, we get a soundNIZK.

Note that the new proof needs $\mathsf{PRG}(s) \oplus w$ to be part of the proof. However, the proof of the instance-independent NIZK is now only a polynomial in λ (assuming $|s| = \lambda$). So, the overall proof size is $|w| + \mathsf{poly}(\lambda)$ which means the new NIZK scheme is Rate-1 in the witness size. Moreover, all the properties of instance-independent NIZK follow-through. If the instance-independent NIZK has a knowledge extractor, we can use it to extract the seed s and then recover w using PRG. Using the

⁷...or only poly-logarithmically dependent.

simulator for instance-independent NIZK, we can simulate the proof and by relying on the security of PRG, we can output a random string of length |w| as part of the proof. It looks like if we can construct an instance-independent NIZK, we can bootstrap it to create a Rate-1 NIZK scheme using a PRG scheme.

Instance-Independent NIZK from ERDel. We observe that it is trivial to construct an instanceindependent NIZK using an encrypted RAM delegation scheme. The idea is to use $x_{pb} := x$ and $x_{pr} := w$. From the efficiency of an ERDel scheme, the size of proof is poly-logarithmic in x_{pb} and polynomial in x_{pr} . This satisfies the efficiency requirements for instance-independent NIZKs. We can extract x_{pr} using the extractor for the encrypted RAM delegation scheme which gives us our witness. The input-hiding property of ERDel translates to the computational zero-knowledge property of an instance-independent NIZK scheme. Hence, by using an encrypted RAM delegation scheme along with a PRG scheme, we get a Rate-1 NIZK scheme with an argument of knowledge. However, there is a subtle issue with the usage of PRG. As encrypted RAM delegation is defined for RAM machines, the size of the witness can vary and is dependent on the input size of instance x and the running time T. Hence, we use a single-bit PRF instead of a PRG scheme in the counter-mode to encode the witness. In what follows we formally present and analyze our construction.

5.1 Construction

Construction 5.1. Let r1Enc = (Encode, Decode) be a secure rate-1 message encoding scheme (Definition 2.14) and $\mathcal{E}Del = (Setup, Prove, Verify)$ be an encrypted RAM delegation scheme w.r.to HT (Definition 3.1) for machine \mathcal{M}' defined as follows – \mathcal{M}' has (λ, \mathcal{M}) hardwired, takes (x, ρ) as public input, and sk as private input, computes $w = r1Enc.Decode(1^{\lambda}, \rho, sk)$ and output 1 if and only if $\mathcal{M}(x, w) = 1$. We construct a Rate-1 NIZK for the language $\mathcal{L}_{\mathcal{M},T} = \{(x, w) : \mathcal{M}(x, w, T) = 1\}$ as follows:

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

Prove(crs, x, w) $\rightarrow \pi$. This probabilistic algorithm parses crs as (λ , 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} Del proof del. $\pi \leftarrow \mathcal{E}$ Del.Prove(del.crs, x_{pb}, x_{pr}) and output the proof $\pi = (\rho, del.\pi)$.

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

Theorem 5.2. If \mathcal{E} Del is a secure encrypted RAM delegation scheme w.r.to hash tree HT for language $\mathcal{L}_{\mathcal{M}'}$ (Definition 3.1) and r1Enc is a secure rate-1 message encoding scheme (Definition 2.14), then Construction 5.1 is a rate-1 NIZK scheme for language $\mathcal{L}_{\mathcal{M}}$.

Proof. We show that Construction 5.1 is correct and efficient as follows:

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

Efficiency. Note that the crs of the NIZK scheme is nothing but the crs for \mathcal{E} Del. Hence, $|crs| = |del.crs| = poly(\lambda, \log T)$. Similarly, running time of Setup follows from running time of \mathcal{E} Del.Setup's running time. In Prove, we are using \mathcal{E} Del.Prove whose running time is $poly(\lambda, T, (|x| + |w|))$ and r1Enc whose running time $poly(\lambda, |w|)$. It follows that the running time of Prove is $poly(\lambda, T, |x|, |w|)$. The size of the proof is $|\rho| + |del.\pi| = |w| + poly(\lambda, \log T, \log(|x| + |w|))$. The Verify algorithm computes using HT which takes $O((|x| + |w|)\lambda)$ time and then runs \mathcal{E} Del.Verify. Hence the running time of Verify follows from the running time \mathcal{E} Del.Verify and HT.Hash. In summary, Construction 5.1 has the following efficiency properties:

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

Moreover, we remark that the running time of verifier can reduced by considering a preprocessing phase where x, ρ (parsed from del. π) are hashed independent of Verify. This renders the running time of Verify to be virtually independent of |x| and |w|. In addition, due to the polylogarithmic dependence |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 [GKP⁺13] 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.

5.2 Security Analysis

Here we show that Construction 5.1 satisfies straight-line extraction and computational zero-knowledge.

Proof of straight-line extraction. We show that if \mathcal{E} Del satisfies straight-line extraction and r1Enc is correct, then Construction 5.1 satisfies straight-line extraction. Let the extractor do the following – (1) generate (crs, td) using \mathcal{E} Del. \mathcal{E} . Set crs = del.crs and td = del.td, (2) to extract w^* upon receiving $\pi = (\rho, \text{del}.\pi)$, compute sk^{*} $\leftarrow \mathcal{E}$ Del. \mathcal{E} (del.td, del. π) and output $w^* = r1$ Enc.Decode(1^{λ}, ρ , sk^{*}).

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

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

We say that \mathcal{A} wins if

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

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

$$\mathsf{Adv}_1 = |\Pr[\mathcal{A} \text{ wins}] - \Pr[(x, w^*) \notin \mathcal{L}_{\mathcal{M}} \land ((x, \rho), \mathsf{sk}^*) \in \mathcal{L}_{\mathcal{M}'} : \mathsf{Exp}_{\mathsf{NIZK}}]|$$
(11)

If \mathcal{A} wins in Exp_{NIZK}, then either \mathcal{A} breaks the straight-line extraction property of \mathcal{E} Del (Type 1, Adv₁ $\geq \epsilon/2$) or it breaks the correctness of r1Enc (Type 2, Adv₂ $\geq \epsilon/2$).

Type 1. Suppose Eq. (11) is correct, then it holds that

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

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 C and pass it on to 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. (12) 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. (12), \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 r1Enc is secure, then Construction 5.1 satisfies computational zero-knowledge. We define the following hybrids:

- hyb₀^{\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 crs \leftarrow Setup $(1^{\lambda}, T)$ and sends crs to \mathcal{A} . \mathcal{A} queries \mathcal{C} with (x, w) and \mathcal{C} responds with Prove(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 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 5.3. If \mathcal{E} 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 $\lambda \in \mathbb{N}$:

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

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 $|\Pr[1 \leftarrow \mathsf{hyb}_0^{\mathcal{A}}(1^{\lambda})] - \Pr[1 \leftarrow \mathsf{hyb}_1^{\mathcal{A}}(1^{\lambda})]| = \epsilon(\lambda)$ 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}_{del} . \mathcal{B} sends T to \mathcal{C} . \mathcal{B}_{del} passes the crs received from the challenger, \mathcal{C} , to \mathcal{A} .
- 2. \mathcal{A} queries (x, w) to \mathcal{B}_{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 $\mathcal{E}\mathsf{Del}.\mathsf{Setup}$ and $\mathcal{E}\mathsf{Del}.\mathsf{Prove}$ (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 $\mathcal{E}\mathsf{Del}$. \Box

Claim 5.4. If r1Enc is a secure rate-1 message encoding scheme, then for any PPT adversary \mathcal{A} , there exists a negligible function negl(·) such that for every $\lambda \in \mathbb{N}$:

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

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 $|\Pr[1 \leftarrow \mathsf{hyb}_1^{\mathcal{A}}(1^{\lambda})] - \Pr[1 \leftarrow \mathsf{hyb}_2^{\mathcal{A}}(1^{\lambda})]| = \epsilon(\lambda)$ 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}_{r1Enc} . \mathcal{B} samples del.crs $\leftarrow \mathcal{E}\mathsf{Del}.\mathcal{S}(1^{\lambda}, 1^{\lambda}, T)$ and sends 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 $(\rho, \text{del}.\pi)$ where $\text{del}.\pi \leftarrow \mathcal{E}\text{Del}.\mathcal{S}(\text{del.crs}, x, \rho)$ to \mathcal{A} . Otherwise, output \perp .
- 3. Output whatever \mathcal{A} outputs.

If C uses r1Enc.Encode (resp. using random ρ) in the experiment then \mathcal{B}_{r1Enc} perfectly simulates $hyb_1^{\mathcal{A}}$ (resp. $hyb_2^{\mathcal{A}}$). Thus \mathcal{B}_{r1Enc} has advantage $\epsilon(\lambda)$ in breaking the security of r1Enc.

Remark 5.5. If we rely on a multi-theorem NIZK argument and plug-in the Rate-1 NIZK from Construction 5.1 in the reusable \mathcal{E} Del scheme (Definition 3.2), we get an optimal rate $r\mathcal{E}$ Del scheme.

We conclude the section with an explicit restatement of Theorem 5.2.

Theorem 5.6 (Rate-1 NIZK). Assuming the existence of encrypted RAM delegation and rate-1 message encoding scheme (implied by pseudorandom functions), there exists a rate-1 NIZK argument scheme for RAM machines (universal and unbounded) with straight-line extraction.

We state the following corollary of Theorems 5.6, 4.2 and Remarks 2.12, 2.10.

Corollary 5.7. Assuming LWE/DLIN/sub-exponential DDH (and QR) assumptions, there exists a rate-1 non-interactive zero-knowledge argument scheme with straight-line extraction for RAM machines.

6 Homomorphic NIZK

In this section, we outline the definition and design of the unbounded homomorphic NIZK, which we call truly fully homomorphic NIZK, built upon rate-1 NIZKs. The primitive, initially introduced by [ADKL19], enables homomorphic evaluations on top of multiple proofs to generate a composite NIZK proof. Unlinke a standard NIZK scheme, a homomorphic NIZK includes an additional algorithm Eval that takes as input crs, C, $\{(C_j, b_j, \pi_j)\}_{j \in [k]}$ and outputs C', b, π for C' as the composition of C, C_1, \ldots, C_k . An additional property of homomorphic NIZK is unlinkability: the evaluated proof π for the instance (C', b) must be indistinguishable from a freshly generated proof using a witness ω such that $C'(\omega) = b$. Moreover, the design of [ADKL19] inherently supports unbounded circuit sizes and evaluation hops as it is based on the GOS design [GOS06]. The GOS design gives NIZK for circuit satisfiability by generating a sequence of gate-by-gate NIZK proofs under a single CRS. After presenting our design, we provide a detailed analysis including security and scalability, which examines that our approach of composing rate-1 NIZKs supports all of the above truly fully homomorphic NIZK requirements.

6.1 Definition

A NIZK proof system (Setup, Prove, Verify) is a fully homomorphic NIZK proof system if there exists a PPT algorithm Eval with the following output behavior:

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

Boolean Circuit Composition The definition of the circuit composer is exactly identical to the composer defined by [ADKL19], with only slight adjustments to the symbols. We consider each circuit C as a directed acyclic graph and provide a detailed breakdown of the algorithm, laying the groundwork for the circuit decomposition in the following section.

 $\mathsf{Compose}(C, C_1, \ldots, C_k) \to C'.$

- 1. Let directed acyclic graph G = (V, E) represent the boolean circuit $C : \{0, 1\}^k \to \{0, 1\}$. In graph G, each node $v \in V$ of in-degree 0 is an input node labelled by either 0 or 1, and there are k such nodes. For each remaining node in G, it is either an AND gate, or an OR gate, or a NOT gate, labeled with a bit representing the value of the gate's output wire. Each internal wire of the circuit corresponds to a directed edge in G, where the direction of such edge indicates the direction of signal flow. We remark that the node v_{out} which has an out-degree of 0 represents the output gate. For all $i \in [k]$, let graphs G_i represent the boolean circuit C_i .
- 2. Define graph G' = (V', E') as follows:

$$V' = (\bigcup_{i=1}^{k} V_i) \cup V$$
, and $E' = (\bigcup_{i=1}^{k} E_i) \cup E$.

3. Define the merge of vertices $u, v \in V$ in graph G = (V, E) as the following: First, for each $w \in V \setminus \{u, v\}$, replace edge (w, v) with new edge (w, u), and (v, w) with (u, w) if

there exists such edges. Next, remove node v, and edges (u, v), (v, u) if there exists such edges. Formally, the new graph G' = (V', E') resulting from merging node $u, v \in V$ is defined as the following: $V' = V \setminus \{v\}$

$$V = V \setminus \{v\},$$

$$E'' = E \setminus \{(u, v), (v, u)\},$$

$$E' = (E \setminus \{(w, v), (v, w) : w \in V\}) \cup \{(u, w) : (v, w) \in E''\} \cup \{(w, u) : (w, v) \in E''\}.$$

4. 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 and for all $i \in [k]$. It merges every pair of u_i, v_i in graph G' for $i \in [k]$ and outputs C' as the resulting circuit.

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

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

Unlinkability The homomorphic NIZK scheme requires that a proof obtained by Eval to be indistinguishable from a freshly generated proof. Formally, for any stateful PPT adversary \mathcal{A} , there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, the following equation holds, as long as $\mathsf{Verify}((C_i, b_i), \pi_i) = 1$ and $C_i(\omega_i) = b_i$ for all $i \in [k]$:

$$\Pr\left[\begin{array}{c} \operatorname{crs} \leftarrow \operatorname{Setup}(1^{\lambda}) \\ (C, \{(C_i, b_i, \omega_i, \pi_i)\}_{i \in [k]}) \leftarrow \mathcal{A}(\operatorname{crs}) \\ (C_0^*, b_0^*, \pi_0^*) \leftarrow \operatorname{Eval}(\operatorname{crs}, C, \{(C_i, b_i, \pi_i)\}_{i \in [k]}) \\ (C_0^*, b_0^*, \pi_0^*) \leftarrow \operatorname{Eval}(\operatorname{crs}, C, \{(C_i, b_i, \pi_i)\}_{i \in [k]}) \\ (C_1^* = \operatorname{Compose}(C, C_1, \dots, C_k) \\ b_1^* = C_1^*(\omega_1, \dots, \omega_k) \\ \pi_1^* = \operatorname{Prove}(\operatorname{crs}, (C_1^*, b_1^*), (\omega_1, \dots, \omega_k)) \\ b \leftarrow \{0, 1\} \\ b' \leftarrow \mathcal{A}(C_b^*, b_b^*, \pi_b^*) \end{array}\right] \leq 1/2 + \operatorname{negl}(\lambda)$$

6.2 Boolean Circuit Decomposition

To achieve unlinkability, we present a decomposition algorithm Decompose which breaks down a circuit into atomic circuits. A boolean circuit C is modeled as a directed acyclic graph G = (V, E), where each vertex of in-degree 0 are labeled as either 0 or 1, representing each input gate to circuit C. Each of the remaining vertices either corresponds to an AND gate, an OR gate, or a NOT gate. Edges of G represent the wires of circuit C. Before introducing the circuit decomposition algorithm, we explain the term "cutting node" with respect to graph G. For each $v \in V$, v is a "cutting node" if and only if by cutting the gate at vertex v breaks the boolean circuit C into two independent boolean circuits. At a high level, by parsing each cutting node of G, 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 $T = (V_T, V_E)$.

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

- 1. Let directed acyclic graph G = (V, E) represent the boolean circuit $C : \{0, 1\}^k \to \{0, 1\}$.
- 2. Consider the split of a vertex $u \in V$ as the following: The first step is adding a new vertex u' and a new edge (u, u'). Next for each vertex v such that $(u, v) \in E$, replace the edge with (u', v). Intuitively, the node u is "split" into the edge (u, u'). Formally, the new graph G' = (V', E') resulting from splitting the node u into an edge can be defined as:

$$V' = V \cup \{u'\},$$

 $E' = (E \setminus \{(u, v) : v \in V\}) \cup \{(u', v) : (u, v) \in E\} \cup \{(u, u')\}.$

- 3. Define term "cutting node" as the following: Given G = (V, E) and $u \in V$, split node u into an edge (u, u') and obtain a new graph G' = (V', E'). Vertex $u \in V$ is a cutting node if and only if the out-degree of u (in graph G) is non-zero and edge (u, u') is a cut of graph G'.
- 4. For all cutting nodes $u \in V$, the Decompose algorithm splits node u into a cutting edge and let set C be the set of all such edges. It obtains a new graph G' = (V', E'). By removing the edge set C from graph G', it represents each weakly-connected component of graph (V', E' - C) as a boolean circuit. For the weakly connected component that only contains a single vertex, it represents such component as an empty circuit.
- 5. Following from step 4, it contracts each of the above circuit in graph G' into a single vertex and obtains a tree structure $T = (V_T, E_T)$ where each vertex $v \in V_T$ represents a boolean circuit C_v as above, and the edge set E_T corresponds to C. Note that T is a tree because E_T only contains cutting edges.
- 6. The Decompose algorithm outputs $T, \{C_v\}_{v \in V_T}$.

Theorem 6.1. Given any set of boolean circuits $\{C, C_1, \ldots, C_\ell\}$ where $C = \{0, 1\}^\ell \to \{0, 1\}$, let $(T, \{C_v\}_{v \in V_T})$ 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,

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

where T_i for $i \in [\ell]$ are as defined above, and T_0 is the tree T with all leaf nodes removed.

Proof. We note that on composing $C, \{C_i\}_{i \in [\ell]}$, one is contracting 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 cut 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.

6.3 Construction

Consider circuit $C' = \{0, 1\}^{\ell} \to \{0, 1\}$, output bit *b*, and a valid homomorphic NIZK proof π for circuit satisfiability instance (C', b), then proof π could be either generated from $\mathsf{Prove}(\mathsf{crs}, C, b, \{b_i\}_{i \in [\ell]})$ algorithm taking (C, b) as instance and $\{b_i\}_{i \in [k]}$ as witness, or homomorphically evaluated from $\mathsf{Eval}(\mathsf{crs}, C, \{(C_i, b_i, \pi_i)\}_{i \in [k]})$ such that $C' = \mathsf{Compose}(C, C_1, \ldots, C_k)$. To achieve unlinkability property, a freshly generated proof must be indistinguishable from any evaluated proof. As discussed

in technical overview, the naive approach won't work as one could not argue unlinkability since a freshly generated NIZK proof is from the language

$$\exists b_1, \ldots, b_\ell, \text{ s.t. } C'(b_1, \ldots, b_\ell) = b,$$

whereas evaluated proof is by language

$$\forall i \in [k], \exists \pi_i \text{ s.t. Verify}(\operatorname{crs}, (C_i, b_i), \pi_i) = 1 \text{ and } C(b_1, \ldots, b_k) = b.$$

The idea is that we let the prover decompose the circuit C' into the smallest atomic circuits and generate the fresh homomorphic NIZK proof π by simulating algorithm Eval level by level, following the topological order of those atomic circuits. As a result, a freshly generated proof and a evaluated proof would be generated by the underlying rate-1 NIZK scheme from a universal language. The unlinkability property immediately holds by the zero-knowledge property of rate-1 NIZK. Our construction is as follows:

 $\mathsf{Setup}(1^{\lambda}) \to \mathsf{crs.}$ The Setup algorithm generates crs with respect to language \mathcal{L} as follows:

$$\mathsf{crs} \leftarrow \mathsf{NIZK}.\mathsf{Setup}(1^{\lambda}).$$

Language L
Instance: crs, C, b
Witness: {(b_i, π_i)}_{i∈[k]}.
Output: The Language goes step by step, as the following:

It runs Decompose(C) and obtains (T, {C_v}_{v∈V_T}).
Let Rt be the root node of T, it checks if C_{Rt} takes k bits of input. If the check fails, it outputs 0 and aborts.
Let v₁,..., v_k be the k child nodes of Rt. For each i ∈ [k], let circuit C_i be the circuit by the subtree rooted at v_i, combining all the circuits within that subtree.
For each i ∈ [k], if C_i is not empty, it checks if NIZK.Verify(crs, (crs, C_i, b_i), π_i) = 1. If any of the check fails, it outputs 0 and aborts.

- It outputs 1 if and only if $C_{\mathsf{Rt}}(b_1,\ldots,b_k) = b$.

Prove(crs, $C, b, \{b_i\}_{i \in [k]}$) $\rightarrow \pi$. For all $i \in [k]$, the prover's algorithm sets C_i as an empty circuit, π_i as an empty string. It runs $\mathsf{Eval}(\mathsf{crs}, C, \{(C_i, b_i, \pi_i)\}_{i \in [k]})$ and outputs proof π .

 $\mathsf{Eval}(\mathsf{crs}, C, \{(C_i, b_i, \pi_i)\}_{i \in [k]}) \to (C', b, \pi)$. The algorithm Eval follows these steps:

- 1. Set $(T, \{C_v\}_{v \in V_T})$ to be $\mathsf{Decompose}(C)$.
- 2. Given the *i*-th leaf node $v \in V_T$, it sets $C'_v = C_i$, b_v as b_i , π_v as π_i .
- 3. Given a non-leaf node $v \in V_T$ with an in-degree of ℓ , let (v_1, \ldots, v_ℓ) be the child nodes of C_v where C_v takes $(b_{v_1}, \ldots, b_{v_\ell})$ as input. It computes

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

and generates

$$\pi_v \leftarrow \mathsf{NIZK}.\mathsf{Prove}(\mathsf{crs},(\mathsf{crs},C'_v,b_v),\{(b_{v_i},\pi_{v_i})\}_{i\in\ell})$$

- 4. For every $v \in V_T$, it runs step 2 or 3 using vertex v, in a topological order.
- 5. Let node Rt be the root of T, it outputs $C' = C'_{Rt}$, $b = b_{Rt}$, and $\pi = \pi_{Rt}$.

Verify(crs, $(C, b), \pi) \rightarrow 0/1$. It outputs NIZK.Verify(crs, (crs, $C, b), \pi$).

Theorem 6.2. If NIZK is a rate-1 NIZK scheme for language \mathcal{L} , then Construction 6.3 is a homomorphic NIZK scheme.

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 rate-1 property of the underlying NIZK which is specified by the efficiency of rate-1 NIZK.

Lemma 6.3. Assume that NIZK satisfies rate-1 property, then for every $\lambda \in \mathbb{N}$, the following holds:

- For $\operatorname{crs} \leftarrow \operatorname{Setup}(1^{\lambda}), |\operatorname{crs}| \le \operatorname{poly}(\lambda).$
- 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 $\pi \leftarrow \mathsf{Prove}(\mathsf{crs}, C, b, \{b_i\}_{i \in [k]}), |\pi| \le \mathsf{poly}(\lambda, |C|).$

Proof. By rate-1 NIZK, the crs size is at most $poly(\lambda)$ 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 NIZK.Verify step. Again by efficiency of rate-1 NIZK, the NIZK verifier running time is bounded by $poly(\lambda, |x|, |\omega|)$ for some $poly(\cdot, \cdot, \cdot)$. Thus the size of language \mathcal{L} is at most $poly(\lambda, |C|)$. By the rate-1 property of NIZK, there is a $poly(\lambda)$ -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 Decompose(C) tree. Thus, the overall overhead size is bounded by $poly(\lambda, |C|)$. Since the overall witness length $k \leq |C|$, we conclude that $|\pi| \leq poly(\lambda, |C|)$.

6.4 Security Analysis

Lemma 6.4 (Argument of knowledge). Assume that NIZK satisfies straight-line extraction, then the above homomorphic NIZK design 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.

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

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

Next, it outputs crs as nizk.crs and td as (nizk.crs, 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_ℓ) , 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 $C'_{Rt} = C$ by definition of step 1. It sets π_{Rt} as π and b_{Rt} as b. It parses td as (nizk.crs, nizk.td), and we note that crs is set as 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 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_ℓ) . Since C'_v takes ℓ bits of input, v also has a number of ℓ child nodes (v_1, \ldots, v_ℓ) . 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 6.5. Assuming that NIZK satisfies adaptive argument of knowledge, then there exists a negligible function $negl(\cdot)$ such that for all $\lambda \in \mathbb{N}$, the following holds for all non-leaf node $v \in V_T$:

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

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 nizk.crs, nizk.td. \mathcal{B} sets crs = nizk.crs, td = (nizk.crs, nizk.td) and sends crs, 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 $\mathcal{E}', \omega_{\mathsf{Rt}}$ is extracted using 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 $\mathsf{NIZK}.\mathsf{Verify}(\mathsf{nizk.crs}, (\mathsf{crs}, C'_{\mathsf{Rt}}, b_{\mathsf{Rt}})$ 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} , Verify(crs, $(C, b), \pi) = 1$ but ω_v is not a valid witness for (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 $\mathsf{NIZK}.$

 \mathcal{A} sets \mathcal{L} and λ . \mathcal{B} queries the NIZK challenger with 1^{λ} and the challenger responses with nizk.crs, nizk.td. \mathcal{B} sets and outputs crs = nizk.crs and td = (nizk.crs, 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 $(\operatorname{crs}, C'_u, b_u) \in \mathcal{L}$, which already implies that NIZK.Verify $(\operatorname{crs}, (\operatorname{crs}, C'_v, b_v), \pi_v) = 1$. If ω_v is not a valid witness, then \mathcal{B} breaks the argument of knowledge property of nizk by extracting and outputting $(\operatorname{crs}, C'_v, b_v)$.

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 6.5, for each internal node v with child nodes (v_1, \ldots, v_ℓ) , 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 6.6 (Unlinkability). Assume that NIZK satisfies zero-knowledge property, then the above homomorphic NIZK design 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_ℓ , 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 6.1, whether we decompose C, C_1, \ldots, C_ℓ or directly decompose, 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 6.7 (Zero knowledge). Assume that NIZK satisfies zero-knowledge property, then the above homomorphic NIZK design 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'_{Rt}, b_{Rt}) where $C'_{Rt} = C$ and $b_{Rt} = b$. Thus our design immediately satisfies zero-knowledge property.

We close the section with the following corollaries, which are immediate by rate-1 NIZK (Corollary 5.7).

Corollary 6.8. Assuming the existence of rate-1 NIZK argument scheme, there exists a homomorphic NIZK scheme.

Corollary 6.9. Assuming LWE/DLIN/sub-exponential DDH (and QR), there exists a homomorphic NIZK scheme.

7 Rate Preserving Maliciously Secure mrNISC

In this section, we provide the definitions, construction, and analysis of a maliciously secure multiparty 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 (r \mathcal{E} Del) w.r.to HT.

7.1 Definition

We provide the definitions for a mrNISC scheme for n parties (SC) in the semi-malicious adversary setting below.

Syntax. A multi-party reusable non-interactive secure computation scheme for n parties (SC) 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 $\mathsf{crs.}$
- InpEnc(crs, id, x; r) $\rightarrow (\hat{x}, \tau)$. 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, \hat{x} and trapdoor τ .
- FuncEnc(crs, id, f, $\{\hat{x}_{idx}\}_{idx\in I}, \tau_{id}\} \to \hat{f}_{id}$. The function encoding algorithm takes as input a crs, identity of the party id, description of function f, public encodings for all $idx \in I \subseteq [n]$, $\{\hat{x}_{idx}\}_{idx}$ (id $\in I$), trapdoor for the id-th party τ_{id} , and outputs the function encoding for the id-th party, \hat{f}_{id} .
- $\mathsf{Eval}(f, \{\widehat{f}_{\mathsf{id}}\}_{\mathsf{id}\in I}, \{\widehat{x}_{\mathsf{id}}\}_{\mathsf{id}\in I}) \to y$. The evaluation algorithm takes as input 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 SC scheme for n parties (Setup, InpEnc, FuncEnc, 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:

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

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

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

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 2. If the adversary instead behaves maliciously, we say the scheme is 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, \mathcal{A} outputs guess b'. Output b'.

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

 $\mathsf{Expt}_1^{\mathcal{S},\mathcal{A}}(1^{\lambda})$. 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, \mathcal{A} outputs guess b'. Output b'.

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

Figure 2: Real and Ideal experiments for security of SMSC

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.

Remark 7.3 ([BL20, BJKL21, AJJM21, Shi22]). Assuming SXDH/LWE, there exists a mrNISC scheme for n parties for all efficient functions that is semi-maliciously secure.

7.2 Construction

Let SMSC be a semi-maliciously secure mrNISC scheme. Our high level idea for upgrading it to a maliciously secure mrNISC scheme SC is that, in addition to the input encodings (commitments to private inputs), we also generate a private input encoding using $r\mathcal{E}Del.PrivEnc$ with the party's input x and decommitment information de. Next, in order to generate function encodings for SC, along with function encodings for SMSC, we must also generate a proof that these encodings were generated in accordance with SMSC's function encoding algorithm, using inputs x, de. In particular, the overview of our design SC looks as follows:

 $\mathbf{CRS} \ \mathbf{generation.} \ \mathbf{For} \ \mathbf{each} \ \mathbf{party}, \ \mathbf{generate} \ \mathbf{a} \ \mathbf{reusable} \ \mathbf{encrypted} \ \mathbf{RAM} \ \mathbf{delegation} \ \mathbf{crs} \ \mathbf{using} \ \mathbf{r} \\ \mathcal{E} \\ \mathbf{Del}. \\ \mathbf{Setup}. \ \mathbf{crs} \ \mathbf{reusable} \$

Input Encoding. For any party, let $(\hat{x}, de) \leftarrow SMSC.InpEnc(x)$. Then sample $(del.\pi_{enc}, del.aux) \leftarrow r\mathcal{E}Del$. PrivEnc(x, de). Output SC $\hat{x} = (\hat{x}, del.\pi_{enc})$ and SC.de = del.aux.

Function Encoding. For any party, let $\hat{f} \leftarrow \mathsf{SMSC}.\mathsf{FuncEnc}(f)$. Then sample del. $\pi_{\mathsf{edel}} \leftarrow \mathsf{r}\mathcal{E}\mathsf{Del}.\mathsf{Prove}(x_{\mathsf{pb}} := (\hat{x}, \hat{f}), x_{\mathsf{pr}} := \mathsf{del.aux})$. Output $\mathsf{SC}.\hat{f} = (\hat{f}, \mathsf{del}.\pi_{\mathsf{edel}})$.

Finally to evaluate f, the evaluator checks that the proofs from all parties verify, and then it uses the semi-malicious evaluator to learn the output of the computation. In a nutshell, the semimalicious security of the underlying mrNISC protocol, and knowledge soundness and input-hiding property of our reusable encrypted RAM delegation gives a maliciously secure mrNISC protocol.

Of course, as we discussed in the introduction, this direct compiler is only nearly rate-preserving due to the decommitment information's dependence on the length of input x. Next, we give our formal construction that overcomes even this barrier to give a fully rate-1 compiler for maliciously secure mrNISC. This is accomplished by instead creating an input encoding for a short random seed for a rate-1 encoding of the input x and then creating the function encoding over a function g that first extracts x from the rate-1 encoding and then evaluates f. More formally, we have the following construction:

Construction 7.4 (Malicious SC). Let r1Enc = (Encode, Decode) be a secure rate-1 message encoding scheme (Definition 2.14) and SMSC = (InpEnc, FuncEnc, Eval), be a semi-maliciously secure mrNISC scheme (Definition 7.1). Using a reusable encrypted RAM delegation scheme $r\mathcal{E}Del = (Setup, PrivEnc, Prove, VerifyEnc, Verify)$ w.r.to HT for language $\mathcal{L}_{\mathcal{M},T}^{SC} = \{(x_{pb}, x_{pr}, T) : \mathcal{M}_{SC}(x_{pb}, x_{pr}, T) = 1\}$ where \mathcal{M}_{SC} is defined in Figure 7.4, we provide the construction of an adaptively secure mrNISC scheme SC for *n* parties for the function *f* below.

$\fbox{ RAM Machine \mathcal{M}_{SC}}$	Function g
Public Input: id, f , $\{\hat{x}_{idx}\}_{idx \in I}$, \hat{f}_{id} Private Input: x, r, τ Output: Output 1 if and only if	Hardwired: $\lambda, \{\rho_{id}\}_{id \in I}, f$ Input: $\{sk_{id}\}_{id \in I}$ Output:
$ \begin{array}{l} - (\widehat{x},\tau) = InpEnc(id,x;r). \\ - \widehat{f_{id}} = FuncEnc(id,f,\{\widehat{x}_{idx}\}_{idx\in I},\tau;r). \end{array} $	$\begin{array}{ll} - & \forall id \in I, \ x_{id} = r1Enc.Decode(1^{\lambda}, \rho_{id}, sk_{id}). \\ - & \text{Output} \ f(\{x_{id}\}_{id \in I}). \end{array}$

- Setup $(1^{\lambda}) \rightarrow \text{crs.}$ This probabilistic algorithm samples a reusable encrypted RAM delegation crs for each id $\in [n]$, del.crs_{id} $\leftarrow r\mathcal{E}$ Del.Setup $(1^{\lambda}, 1^{\ell_{\text{pr}}})$ where ℓ_{pr} is the maximum length of the private input for \mathcal{M}_{SC} across all n parties. Output crs = (del.crs_{id})_{id $\in [n]$}.
- $\mathsf{InpEnc}(\mathsf{crs}, \mathsf{id}, x; r) \to (\widehat{x}, \tau)$. This probabilistic algorithm parses crs , using $\mathsf{del.crs}_{\mathsf{id}}$ and does the following.
 - 1. Sample $r' \leftarrow \{0,1\}^{\lambda}$. Compute $(\rho, \mathsf{sk}) = \mathsf{r1Enc}.\mathsf{Encode}(1^{\lambda}, x)$.
 - 2. Sample input encoding for SMSC for sk, $(smsc.sk, smsc.\tau) \leftarrow SMSC.InpEnc(sk; r')$.
 - 3. Compute private encoding from r \mathcal{E} Del, (del.eh_{pr}, del. π_{enc} , del.aux) \leftarrow r \mathcal{E} Del.PrivEnc(del.crs_{id}, (sk, r', smsc. τ)).

- 4. Output $\hat{x} = (\rho, \text{smsc.sk}, \text{del.eh}_{pr}, \text{del.}\pi_{enc}), \tau = \text{del.aux.}$
- $\begin{aligned} \mathsf{FuncEnc}(\mathsf{crs},\mathsf{id},f,\{\widehat{x}_{\mathsf{idx}}\}_{\mathsf{idx}\in I},\tau_{\mathsf{id}}) &\to \widehat{f}_{\mathsf{id}}/\bot. \text{ This probabilistic algorithm parses the crs, uses } (\mathsf{del.crs}_{\mathsf{idx}})_{\mathsf{idx}},\\ \tau_{\mathsf{id}} \text{ as } \mathsf{del.aux}_{\mathsf{id}}, \widehat{x}_{\mathsf{idx}} \text{ as } (\rho_{\mathsf{idx}},\mathsf{smsc.sk}_{\mathsf{idx}},\mathsf{del.eh}_{\mathsf{pr},\mathsf{idx}},\mathsf{del}.\pi_{\mathsf{enc},\mathsf{idx}}) \text{ for } \mathsf{idx} \in I, \text{ and does the follow-ing.} \end{aligned}$
 - 1. If for any $idx \in I$, $r\mathcal{E}Del$. $VerifyEnc(del.crs_{idx}, del.eh_{pr,idx}, del.\pi_{enc,idx}) = 0$, abort and output \bot .
 - 2. Otherwise, sample, using SMSC, function encoding for g defined in Figure 7.4. smsc. $\hat{g}_{id} \leftarrow$ SMSC.FuncEnc(id, g, {smsc. \hat{s}_{idx} }_{idx}, smsc. τ_{id} ; r'_{id}) where smsc. τ_{id} , r'_{id} are derived from del.aux_{id}.
 - 3. Compute proof, del. $\pi_{\mathsf{edel},\mathsf{id}} \leftarrow \mathsf{r}\mathcal{E}\mathsf{Del}.\mathsf{Prove}(\mathsf{del.crs}_{\mathsf{id}}, (\mathsf{id}, g, \{\mathsf{smsc.sk}_{\mathsf{idx}}\}_{\mathsf{idx}}, \mathsf{smsc.}\widehat{g}_{\mathsf{id}}), \mathsf{del.aux}_{\mathsf{id}}).$
 - 4. Output $\widehat{f}_{id} = (\text{smsc.} \widehat{g}_{id}, \text{del.} \pi_{\text{edel}, id}).$

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

- 1. Construct g and compute for $id \in I$, $h_{pb,id} = HT.Hash(ht.hk, id, g, \{smsc.sk_{idx}\}_{idx \in I}, smsc.g_{id})$.
- 2. If for any id $\in I$, r \mathcal{E} Del.VerifyEnc(del.crs_{id}, del.eh_{pr,id}, del. $\pi_{enc,id}$) = 0 or r \mathcal{E} Del.Verify(del.crs_{id}, h_{pb,id}, del.eh_{pr,id}, 1, del. $\pi_{edel,id}$) = 0, abort and output \perp .
- 3. Output SMSC.Eval $(g, \{\text{smsc.s}\hat{k}_{id}\}_{id}, \{\text{smsc.}\hat{g}_{id}\}_{id})$.

Theorem 7.5. If SMSC is a semi-maliciously secure mrNISC scheme (Definition 7.1), r1Enc is a secure rate-1 message encoding scheme (Definition 2.14), and r \mathcal{E} Del is a reusable encrypted RAM delegation scheme w.r.to HT (Definition 3.2) for machine \mathcal{M}_{SC} , then Construction 7.4 is a Rate-1 maliciously secure mrNISC scheme.

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

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

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

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

$$\begin{split} \mathsf{r}\mathcal{E}\mathsf{Del}.\mathsf{VerifyEnc}(\mathsf{del.crs}_{\mathsf{id}},\mathsf{del.eh}_{\mathsf{pr},\mathsf{id}},\mathsf{del}.\pi_{\mathsf{enc},\mathsf{id}}) &= 1 \ \mathrm{and} \\ \mathsf{r}\mathcal{E}\mathsf{Del}.\mathsf{Verify}(\mathsf{del.crs}_{\mathsf{id}},\mathsf{h}_{\mathsf{pb},\mathsf{id}},\mathsf{del.eh}_{\mathsf{pr},\mathsf{id}},1, \end{split}$$

 $\mathsf{r}\mathcal{E}\mathsf{Del}.\mathsf{Prove}(\mathsf{del.crs}_{\mathsf{id}},\mathsf{id},g,\{\mathsf{smsc.}\widehat{\mathsf{sk}}_{\mathsf{idx}}\}_{\mathsf{idx}},\mathsf{smsc.}\widehat{g}_{\mathsf{id}},\mathsf{del.aux}_{\mathsf{id}})) = 1$

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

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

INPUT ENCODING. As K_{id} and r'_{id} are λ -bit strings, length of smsc. \hat{sk}_{id} and smsc. τ_{id} is polynomial in λ . Similarly, the length of del.eh_{pr,id} and del. $\pi_{enc,id}$ is polynomial in λ . However, ρ_{id} is the same length as x_{id} (i.e, l). Thus \hat{x}_{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, $|g| = |f| + \text{poly}(\lambda)$ where the polynomial $\text{poly}(\lambda)$ depends on r1Enc. By efficiency of r \mathcal{E} Del, we have that size of del. $\pi_{\text{edel},\text{id}}$ is $\text{poly}(\lambda) + \text{poly}(\log |g|, \lambda)$. Hence, the size of \hat{f}_{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 rate preserving.

7.3 Security Analysis

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 SMSC, r1Enc are secure, and $r\mathcal{E}$ Del is input-hiding, then Construction 7.4 is a maliciously secure mrNISC scheme. We define the following hybrids:

 $hyb_0^{\mathcal{A}}$ This is the honest experiment $Expt_0^{\mathcal{C},\mathcal{A}}$ with malicious adversary \mathcal{A} .

- $\mathsf{hyb}_{1,j}^{\mathcal{A}}$ This is the same as $\mathsf{hyb}_0^{\mathcal{A}}$ except that for $j \in [n+1]$, we simulate $\mathsf{r}\mathcal{E}\mathsf{Del}$ instantiations for the first j-1 honest parties.
- hyb^{$\mathcal{A}_{2,j}$} This is the same as hyb^{$\mathcal{A}_{1,n+1}$} except that for $j \in [n+1]$, we use the extractor r \mathcal{E} 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 \perp .
- $hyb_3^{\mathcal{A}}$ This is the same as $hyb_{2,n+1}^{\mathcal{A}}$ except that we simulate all encodings of SMSC for honest parties.
- $\mathsf{hyb}_{4,j}^{\mathcal{A}}$ This is same as $\mathsf{hyb}_{3}^{\mathcal{A}}$ except that for $j \in [n+1]$, we sample random ρ_{id} instead of using r1Enc.Encode for the first j-1 honest parties. $\mathsf{hyb}_{4,n+1}^{\mathcal{A}}$ 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 7.6. For every $\lambda \in \mathbb{N}$, any PPT adversary \mathcal{A} , $\Pr\left[1 \leftarrow \mathsf{hyb}_0^{\mathcal{A}}(1^{\lambda})\right] = \Pr\left[1 \leftarrow \mathsf{hyb}_{1,1}^{\mathcal{A}}(1^{\lambda})\right]$.

Proof. The proof of this claim is immediate from the distributions of $hyb_0^{\mathcal{A}}$ and $hyb_{1,1}^{\mathcal{A}}$.

Claim 7.7. If r \mathcal{E} 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 $\lambda \in \mathbb{N}$:

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

Proof. Note that if $j \in \widetilde{H}$ where $\widetilde{H} = [n] \setminus H$, $\mathsf{hyb}_{1,j}^{\mathcal{A}}$ and $\mathsf{hyb}_{1,j+1}^{\mathcal{A}}$ are identical. Hence, w.l.o.g assume $j \in H$. Assuming that there exists an adversary \mathcal{A} that can distinguish between $\mathsf{hyb}_{1,j}^{\mathcal{A}}$ and $\mathsf{hyb}_{1,j+1}^{\mathcal{A}}$ such that $|\Pr[1 \leftarrow \mathsf{hyb}_{1,j}^{\mathcal{A}}(1^{\lambda})] - \Pr[1 \leftarrow \mathsf{hyb}_{1,j+1}^{\mathcal{A}}(1^{\lambda})]| = \epsilon(\lambda)$ 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}_{del} receives crs from the challenger \mathcal{C} . \mathcal{A} submits the number of parties n and the set of honest parties H. If $id \in H$, id < j, sample del.crs_{id} $\leftarrow r\mathcal{E}\mathsf{Del}.\mathcal{S}(1^{\lambda}, 1^{\ell_{\mathsf{pr}}})$. Otherwise if id = j, set del.crs_{id} = crs. Otherwise, if id > j, sample del.crs_{id} $\leftarrow r\mathcal{E}\mathsf{Del}.\mathsf{Setup}(1^{\lambda}, 1^{\ell_{\mathsf{pr}}})$ in accordance with $\mathcal{M}_{\mathsf{SC}}$. Send (del.crs_{id})_{id} to \mathcal{A} .
- 2. When \mathcal{A} queries for input encoding of (id, x_{id}) for $id \in H$, do the following.
 - Sample $r'_{id} \leftarrow \{0,1\}^{\lambda}$. Compute $(\rho_{id}, \mathsf{sk}_{id}) \leftarrow \mathsf{r1Enc.Encode}(1^{\lambda}, x_{id})$.
 - Compute $(smsc.sk_{id}, smsc.\tau_{id}) \leftarrow SMSC.InpEnc(sk_{id}; r'_{id}).$
 - If id < j, sample (del.eh_{pr,id}, del. $\pi_{enc,id}$) \leftarrow r \mathcal{E} Del. \mathcal{S} (del.crs_{id}, 1^{|sk_{id}|+\lambda+|smsc. $\tau_{id}|$). Otherwise, if id > j, sample (del.eh_{pr,id}, del. $\pi_{enc,id}$, del.aux_{id}) \leftarrow r \mathcal{E} Del.PrivEnc(del.crs_{id}, (sk_{id}, r_{id} , smsc. τ_{id})). Otherwise, if id = j, send (sk_{id}, r_{id} , smsc. τ_{id}) to \mathcal{C} and receive (del.eh_{pr,id}, del. $\pi_{enc,id}$).}

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

- 3. \mathcal{A} submits input encoding for $\mathsf{id} \in \widetilde{H}$, parse $\widehat{x}_{\mathsf{id}}$ as $(\rho_{\mathsf{id}}, \mathsf{smsc.sk}_{\mathsf{id}}, \mathsf{del.eh}_{\mathsf{pr},\mathsf{id}}, \mathsf{del.}\pi_{\mathsf{enc},\mathsf{id}})$. If $\mathsf{r}\mathcal{E}\mathsf{Del}$. VerifyEnc(del.crs_{id}, 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 idx ∈ I ∩ H
 , if rEDel.VerifyEnc(del.crs_{idx}, del.eh_{pr,idx}, del.π_{enc,idx}) = 0, abort and output ⊥.
 - Sample smsc. $\hat{g}_{id} \leftarrow$ SMSC.FuncEnc(id, g, {smsc. \hat{s}_{idx} }_{idx}, smsc. τ_{id} ; r'_{id}).
 - If id < j, sample del.π_{edel,id} ← rEDel.S(del.crs_{id}, (id, g, {smsc.sk_{idx}}_{idx}, smsc.ĝ_{id}), 1). Otherwise, if id > j, sample del.π_{edel,id} ← rEDel.Prove(del.crs_{id}, (id, g, {smsc.sk_{idx}}_{idx}, smsc.ĝ_{id}), 1) to C and receive del.π_{edel,id}.

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

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

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

Claim 7.8. For every $\lambda \in \mathbb{N}$, any PPT adversary \mathcal{A} , $\Pr\left[1 \leftarrow \mathsf{hyb}_{1,n+1}^{\mathcal{A}}(1^{\lambda})\right] = \Pr\left[1 \leftarrow \mathsf{hyb}_{2,1}^{\mathcal{A}}(1^{\lambda})\right]$.

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

Claim 7.9. If 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 $\lambda \in \mathbb{N}$, $\Pr\left[\bot \leftarrow \mathsf{hyb}_{2,j}^{\mathcal{A}}(1^{\lambda})\right] \leq \mathsf{negl}(\lambda)$.

Proof. Note that if $j \in H$, $\mathsf{hyb}_{2,j}^{\mathcal{A}}$ 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 $\Pr\left[\perp \leftarrow \mathsf{hyb}_{2,j}^{\mathcal{A}}(1^{\lambda})\right] = \epsilon(\lambda)$ for some non-negligible function $\epsilon(\cdot)$, we construct an adversary $\mathcal{B}_{\mathsf{del}}$ that breaks the straight-line extraction property of r $\mathcal{E}\mathsf{Del}$ as follows:

- 1. \mathcal{B}_{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 id $\in [n]$, if id $\in H$, sample del.crs_{id} \leftarrow r \mathcal{E} Del. $\mathcal{S}(1^{\lambda}, 1^{\ell_{pr}})$. Otherwise if id $\in \tilde{H}$ and id < j, sample (del.crs_{id}, del.td_{id}) \leftarrow r \mathcal{E} Del. $\mathcal{E}(1^{\lambda}, 1^{\ell_{pr}})$ in accordance with \mathcal{M}_{SC} . Otherwise if id $\in \tilde{H}$, id > j, sample del.crs_{id} \leftarrow r \mathcal{E} Del.Setup $(1^{\lambda}, 1^{\ell_{pr}})$. If id = j, set del.crs_{id} = crs. Send (del.crs_{id}) to \mathcal{A} .
- 2. When \mathcal{A} queries for input encoding of (id, x_{id}) for $id \in H$, do the following.
 - Sample $r'_{id} \leftarrow \{0,1\}^{\lambda}$. Compute $(\rho_{id}, \mathsf{sk}_{id}) \leftarrow \mathsf{r1Enc.Encode}(1^{\lambda}, x_{id})$.
 - Compute $(smsc.sk_{id}, smsc.\tau_{id}) \leftarrow SMSC.InpEnc(sk_{id}; r'_{id}).$
 - Sample (del.eh_{pr,id}, del. $\pi_{enc,id}$) $\leftarrow r\mathcal{E}\mathsf{Del}.\mathcal{S}(del.crs_{id}).$

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

- 3. \mathcal{A} submits input encoding for $\mathsf{id} \in \widetilde{H}$, parse $\widehat{x}_{\mathsf{id}}$ as $(\rho_{\mathsf{id}}, \mathsf{smsc.sk}_{\mathsf{id}}, \mathsf{del.eh}_{\mathsf{pr},\mathsf{id}}, \mathsf{del.}\pi_{\mathsf{enc},\mathsf{id}})$. If $\mathsf{r}\mathcal{E}\mathsf{Del}$. VerifyEnc(del.crs_{id}, del.eh_{\mathsf{pr},\mathsf{id}}, \mathsf{del.}\pi_{\mathsf{enc},\mathsf{id}}) = 0, abort and output \bot . Otherwise, if $\mathsf{id} < j$, extract $(\mathsf{sk}_{\mathsf{id}}, r'_{\mathsf{id}}, \mathsf{smsc.}\tau_{\mathsf{id}}) \leftarrow \mathsf{r}\mathcal{E}\mathsf{Del}.\mathcal{E}(\mathsf{del.td}_{\mathsf{id}}, \mathsf{del.}\pi_{\mathsf{enc},\mathsf{id}})$. If $\mathsf{id} = j$, send (del.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 $id \in I \cap H$ such that all the input encodings for parties $idx \in I$ are known, do the following.
 - Sample smsc. $\hat{g}_{id} \leftarrow$ SMSC.FuncEnc(id, g, {smsc. \hat{s}_{idx} }_{idx}, smsc. τ_{id} ; r'_{id}).
 - Sample 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 $(\operatorname{smsc.}\widehat{g}_{id}, \operatorname{del.}\pi_{\operatorname{edel,id}})$ 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 $x_{pb,id} := (id, g, \{smsc.\widehat{sk}_{idx}\}_{idx}, smsc.\widehat{g}_{id})$ and compute $h_{pb,id} = HT.Hash(ht.hk, x_{pb,id})$.
 - For any id ≠ j, rEDel.Verify(del.crs_{id}, h_{pb,id}, del.eh_{pr,id}, 1, del.π_{edel,id}) = 0 abort and output ⊥.
 - If id < j, $\mathcal{M}_{SC}(x_{pb,id}, (sk_{id}, r'_{id}, smsc.\tau_{id})) = 0$, abort and output \perp .
 - Otherwise, if id = j, send $(x_{pb,id}, del.\pi_{edel,id})$ to C.
- 6. Output whatever \mathcal{A} outputs.

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

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 $\lambda \in \mathbb{N}$:

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

Proof. Assuming that there exists an adversary \mathcal{A} that can distinguish between $\mathsf{hyb}_{2,n+1}^{\mathcal{A}}$ and $\mathsf{hyb}_{3}^{\mathcal{A}}$ such that $|\Pr\left[1 \leftarrow \mathsf{hyb}_{2,n+1}^{\mathcal{A}}(1^{\lambda})\right] - \Pr\left[1 \leftarrow \mathsf{hyb}_{3}^{\mathcal{A}}(1^{\lambda})\right]| = \epsilon(\lambda)$ 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} . For $\mathsf{id} \in [n]$, if $\mathsf{id} \in H$, sample $\mathsf{del.crs}_{\mathsf{id}} \leftarrow \mathsf{r}\mathcal{E}\mathsf{Del}\mathcal{S}(1^{\lambda}, 1^{\ell_{\mathsf{pr}}})$. Otherwise, sample $(\mathsf{del.crs}_{\mathsf{id}}, \mathsf{del.td}_{\mathsf{id}}) \leftarrow \mathsf{r}\mathcal{E}\mathsf{Del}\mathcal{E}(1^{\lambda}, 1^{\ell_{\mathsf{pr}}})$ in accordance with $\mathcal{M}_{\mathsf{SC}}$. Send $(\mathsf{del.crs}_{\mathsf{id}})_{\mathsf{id}}$ to \mathcal{A} .
- 2. When \mathcal{A} queries for input encoding of (id, x_{id}) for $id \in H$, do the following.
 - Compute $(\rho_{id}, sk_{id}) \leftarrow r1Enc.Encode(1^{\lambda}, x_{id}).$
 - $\bullet \ {\rm Send} \ (id, sk_{id}) \ {\rm to} \ {\cal C} \ {\rm and} \ {\rm receive} \ smsc. \widehat{sk}_{id}.$
 - Sample (del.eh_{pr,id}, del. $\pi_{enc,id}$) $\leftarrow r\mathcal{E}\mathsf{Del}.\mathcal{S}(\mathsf{del.crs}_{\mathsf{id}}).$

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

- 3. \mathcal{A} submits input encoding for $\mathsf{id} \in \widetilde{H}$, parse $\widehat{x}_{\mathsf{id}}$ as $(\rho_{\mathsf{id}}, \mathsf{smsc.sk}_{\mathsf{id}}, \mathsf{del.eh}_{\mathsf{pr},\mathsf{id}}, \mathsf{del.}\pi_{\mathsf{enc},\mathsf{id}})$. If $\mathsf{r}\mathcal{E}\mathsf{Del}$. $\mathsf{VerifyEnc}(\mathsf{del.crs}_{\mathsf{id}}, \mathsf{del.eh}_{\mathsf{pr},\mathsf{id}}, \mathsf{del}.\pi_{\mathsf{enc},\mathsf{id}}) = 0$, abort and output \bot . Otherwise, extract $(\mathsf{sk}_{\mathsf{id}}, r'_{\mathsf{id}}, \mathsf{smsc.}\tau_{\mathsf{id}}) \leftarrow \mathsf{r}\mathcal{E}\mathsf{Del}.\mathcal{E}(\mathsf{del.td}_{\mathsf{id}}, \mathsf{del.eh}_{\mathsf{pr},\mathsf{id}}, \mathsf{del}.\pi_{\mathsf{enc},\mathsf{id}})$. Extract the input for id-th party, $x_{\mathsf{id}} = \mathsf{r}\mathsf{1}\mathsf{Enc}$. $\mathsf{Decode}(1^{\lambda}, \rho_{\mathsf{id}}, \mathsf{sk}_{\mathsf{id}})$. 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 id $\in I$ such that all the input encodings for parties idx $\in I$ are known, do the following.
 - Send (id, g, I) to C to receive smsc. \hat{g}_{id} .
 - Sample 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 $(\operatorname{smsc.}\widehat{g}_{id}, \operatorname{del.}\pi_{\operatorname{edel},id})$ 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 $x_{pb,id} := (id, g, \{smsc.\widehat{sk}_{idx}\}_{idx}, smsc.\widehat{g}_{id})$ and compute $h_{pb,id} = HT.Hash(ht.hk, x_{pb,id})$.
 - If $r\mathcal{E}$ Del.Verify(del.crs_{id}, h_{pb,id}, del.eh_{pr,id}, 1, del. $\pi_{edel,id}$) = 0 or $\mathcal{M}_{SC}(x_{pb,id}, (sk_{id}, r'_{id}, smsc.\tau_{id})) = 0$ abort and output \perp .
- 6. Output whatever \mathcal{A} outputs.

If C uses SMSC.InpEnc and SMSC.FuncEnc (resp. SMSC.S) in the experiment then \mathcal{B}_{SMSC} simulates $hyb_{2,n+1}^{\mathcal{A}}$ (resp. $hyb_3^{\mathcal{A}}$). Thus \mathcal{B}_{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} , $\Pr\left[1 \leftarrow \mathsf{hyb}_3^{\mathcal{A}}(1^{\lambda})\right] = \Pr\left[1 \leftarrow \mathsf{hyb}_{4,1}^{\mathcal{A}}(1^{\lambda})\right]$.

Proof. The proof of this claim is immediate from the distributions of $hyb_3^{\mathcal{A}}$ and $hyb_{4,1}^{\mathcal{A}}$.

Claim 7.12. If 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 $negl(\cdot)$ such that for every $\lambda \in \mathbb{N}$:

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

Proof. Note that if $j \in \widetilde{H}$, $\mathsf{hyb}_{4,j}^{\mathcal{A}}$ and $\mathsf{hyb}_{4,j+1}^{\mathcal{A}}$ are identical. Hence, w.l.o.g assume $j \in H$. Assuming that there exists an adversary \mathcal{A} that can distinguish between $\mathsf{hyb}_{4,j}^{\mathcal{A}}$ and $\mathsf{hyb}_{4,j+1}^{\mathcal{A}}$ such that $|\Pr[1 \leftarrow \mathsf{hyb}_{4,j}^{\mathcal{A}}(1^{\lambda})] - \Pr[1 \leftarrow \mathsf{hyb}_{4,j+1}^{\mathcal{A}}(1^{\lambda})]| = \epsilon(\lambda)$ for some non-negligible function $\epsilon(\cdot)$, we construct an adversary \mathcal{B}_{r1Enc} that breaks the security of r1Enc as follows:

- 1. Sample crs similar to $hyb_3^{\mathcal{A}}$ 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 id < j, sample $\rho_{id} \leftarrow \{0, 1\}^{|x_{id}|}$. Otherwise if id > j, sample $(\rho_{id}, \mathsf{sk}_{id}) \leftarrow \mathsf{r1Enc}(1^{\lambda}, x_{id})$. Otherwise, if id = j, query the challenger \mathcal{C} with x_{id} to receive ρ_{id} .
 - Perform the rest of the steps same as $hyb_3^{\mathcal{A}}$.
- 3. Perform the rest of the steps same as $hyb_3^{\mathcal{A}}$.

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

We conclude the section with an explicit restatement of Theorem 7.5.

Theorem 7.13. Assuming the existence of reusable encrypted RAM delegation scheme, rate-1 message encoding scheme (implied by pseudorandom functions), non-interactive reusable MPC scheme that is semi-maliciously secure, there exists a rate preserving (optimal/minimal communication overhead) non-interactive reusable MPC scheme which is maliciously secure. In addition,

- The transformation achieves rate-1 input encodings from any semi-maliciously secure scheme.
- If the underlying semi-malicious MPC scheme supports unbounded-arity functions / RAM computations, the transformation supports unbounded-arity functions / RAM computations.

We state the following corollary of Theorems 7.13, 4.2 and Remark 7.3.

Corollary 7.14. Assuming the hardness of LWE and secure rate-1 multi-key fully homomorphic encryption scheme (mkFHE), there exists a maliciously secure mrNISC scheme for n parties (SC) such that:

- The size of input encodings of SC is $|x| + poly(\lambda)$.
- The size of function encodings of SC is $poly(\lambda, |y|, |I|)$ where $y = f(\{x_{id}\}_{id \in I})$. In particular, function encoding sizes are independent of the function size |f| and input sizes $|x_{id}|$.

Proof. We utilize the rate-1 mkFHE scheme (Definition A.3) of [DGMR21] to prove this result. The main insight is that if we replace our rate-1 message encoding scheme r1Enc in Construction 7.4 with rate-1 mkFHE, the input encodings remain rate-1 in the private inputs x_{id} . We can publish these ciphertexts and public keys for each party as part of the public input encodings \hat{x}_{id} . We will use the corresponding secret key as input encoding for SMSC. While computing the function encodings, we can leverage the homomorphic evaluation property of mkFHE to evaluate the function f on the ciphertexts present in $\{\hat{x}_{id}\}_{id\in I}$ to get a rate-1 ciphertext for $f(\{x_{id}\}_{id\in I})$. We will use a special circuit that performs mkFHE's threshold decryption and reconstruction with this ciphertext hardwired as input the underlying SMSC's function encoding algorithm.

Thus, the correctness and efficiency of the scheme follow from the correctness of rate-1 mkFHE and other primitives used in Construction 7.4. Moreover, the security of the scheme flows naturally. In particular, we change $hyb_{4,j}^{\mathcal{A}}$ in the security analysis by relying on threshold decryption property of mkFHE to simulate the ciphertext of honest parties.

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 rate-1 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* $poly(\lambda)$ 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 indices⁸ $I = (i_1, \ldots, i_d) \in [2^{\lambda}]^d$. It outputs the crs crs which consists of d strings crs = (crs₁, ..., crs_d).

- Prove(crs₁, $(x_1, \dots, x_k), (w_1, \dots, w_k)$) $\rightarrow \pi$. The poly-time prover algorithm that takes as input crs₁ (which is the first string in crs), any (unbounded) number of instance-witness pairs of arbitrary size⁹. It runs in time poly $(\lambda, (|x_i|, |w_i|)_{i \in [k]})$ and outputs a proof π .
- ComposeProof $((\operatorname{crs}_i)_{i \in [d']}, (X^{(1)}, \ldots, X^{(\ell)}), (\pi^{(1)}, \ldots, \pi^{(\ell)})) \to \pi$. This proof combiner is a polytime algorithm that takes as input the $(\operatorname{crs})_{i \in [d']}$, 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 .
- Verify($(\operatorname{crs}_i)_{i \in [d']}, X, \pi$) $\rightarrow \{0, 1\}$. The poly-time verification algorithm takes as input $(\operatorname{crs}_i)_{i \in [d']}$, 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 $n_v = n_v(\lambda) \leq \mathsf{poly}(\lambda)$ 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 $(X^{(1)}, \ldots, X^{(\ell)})$ 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 mzkBARG = (Setup, Prove, ComposeProof, Verify) for an **NP** language \mathcal{L} is required to satisfy the following properties:

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

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

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

⁸The j'th extraction index i_j is interpreted as saying that, from an accepting zkBARG proof π created via jcompositions, we can efficiently extract the i_j 'th witness, which is itself an accepting zkBARG proof π' created via (j-1)-compositions.

⁹Note that neither the batch size nor the instance size are fixed at setup time in the multi-hop setting

• If d' = 1 then parse $X = (x_1, \ldots, x_\ell)$ and $W = (w_1, \ldots, w_\ell)$ and output

 $\mathsf{Compose}(\mathsf{crs}_1, X, W) = \mathsf{Prove}(\mathsf{crs}_1, x_1, \dots, x_\ell, w_1 \dots, w_\ell).$

• If d' > 1 then parse $X = (X^{(1)}, \ldots, X^{(\ell)})$ and $W = (W^{(1)}, \ldots, W^{(\ell)})$, 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

$$\pi = \mathsf{ComposeProof}\left((\mathsf{crs}_i)_{i \in [d']}, X^{(1)}, \dots, X^{(\ell)}, \pi^{(1)}, \dots, \pi^{(\ell)}\right).$$

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

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

- 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 $\operatorname{negl}(\cdot)$ such that for any polynomial $d = \operatorname{poly}(\lambda)$, any poly-size tree T of depth $d' \leq d$, and any set of indices $(i_1, \ldots, i_{d'}) \in \operatorname{path}(T)$ and $i_{d'+1} \ldots, i_d \in [2^{\lambda}]$, for every $\lambda \in \mathbb{N}$,
 - $\Pr \begin{bmatrix} \operatorname{Verify}(\operatorname{crs}, X, \pi) = 1 & (\operatorname{crs}, \operatorname{td}) \leftarrow \mathcal{E}(1^{\lambda}, d, (i_{1}, \dots, i_{d})) \\ \wedge X \text{ is consistent with } T & : (X, \pi) \leftarrow \mathcal{A}(\operatorname{crs}) \\ \wedge W^{*} \text{ is not a valid witness for } X_{i_{1},\dots,i_{d'}} \in \mathcal{L} & W^{*} \leftarrow \mathcal{E}(\operatorname{td}, X, \pi) \end{bmatrix} \leq \operatorname{negl}(\lambda).$
- **Zero-knowledge.** There exist PPT simulators $S = (S_1, S_2, S_3)$ sharing state-information for any poly-size adversary A, there exists a negligible function $\operatorname{negl}(\cdot)$ such that for any polynomial $d = \operatorname{poly}(\lambda)$, any poly-size tree T of depth $d' \leq d$, and any set of indices $(i_1, \ldots, i_{d'}) \in \operatorname{path}(T)$ and $i_{d'+1} \ldots, i_d \in [2^{\lambda}]$, for every $\lambda \in \mathbb{N}$,

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

where $\mathcal{O}^{\mathcal{S}_2}((x_1, \dots, x_k), (w_1, \dots, w_k))$ outputs $\mathcal{S}_2(x_1, \dots, x_k)$ if for every $i \in [k]$, w_i is a valid witness for $x_i \in \mathcal{L}$ and \perp otherwise; and $\mathcal{O}^{\mathcal{S}_3}((X^{(1)}, \dots, X^{(\ell)}), (\pi^{(1)}, \dots, \pi^{(\ell)}))$ outputs $\mathcal{S}_3(X^{(1)}, \dots, X^{(\ell)})$ if for every $i \in [k]$, Verify $((\operatorname{crs}_i)_{i \in [d_i]}, X^{(i)}, \pi^{(i)}) = 1$ 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 a rate-1 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 $(\operatorname{crs}_i)_{i \in [d']}$ (if d' = 0 then there is no crs associated with the instance) and a valid witness is a valid aggregated proof corresponding to $(\operatorname{crs}_i)_{i \in [d']}$ (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.

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

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

barg. $\pi \leftarrow s$ -seBARG.Prove(barg.crs₁, $(x_1, \ldots, x_k), (w_1, \ldots, w_k)$)

and output,

nizk. $\pi \leftarrow \mathsf{NIZK}$.Prove(nizk.crs₁, (barg.crs₁, (x_1, \ldots, x_k)), barg. π)

as the proof π .

ComposeProof $(\operatorname{crs}, (X^{(1)}, \ldots, X^{(\ell)}), (\pi^{(1)}, \ldots, \pi^{(\ell)})) \to \pi$. 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 $\operatorname{crs} = ((\operatorname{nizk.crs}_1, \operatorname{barg.crs}_1), \ldots, (\operatorname{nizk.crs}_{d'}, \operatorname{barg.crs}_{d'}, \ldots))$ and compute

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

where a valid witness corresponding to $((\mathsf{barg.crs}_j)_{j \in [d_i]}, X^{(i)}) \in \mathcal{L}'$ 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 Verify $(\perp, X^{(i)}, \pi^{(i)}) = 1$ if and only if $\pi^{(i)}$ is a valid witness for $X^{(i)} \in \mathcal{L}$. Finally, output

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

as the proof π .

Verify(crs, X, π) $\rightarrow 0/1$. Parsing $X = (X^{(1)}, \dots, X^{(\ell)})$, 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 crs = ((nizk.crs_1, barg.crs_1), \dots, (nizk.crs_{d'}, barg.crs_{d'}, \dots)) and then output

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

Theorem 8.3. If s-seBARG is a rate-1 single-hop seBARG and NIZK is a rate-1 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 s-seBARG. \mathcal{E} (resp. 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 $\mathsf{td} = ((\mathsf{nizk.td}_1, \mathsf{barg.td}_1), \dots, (\mathsf{nizk.td}_d, \mathsf{barg.td}_d))$ 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 $X = (X^{(1)}, \dots, X^{(\ell)})$, 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 $j \ge j^*$ (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

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

.

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

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

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 $X^{(i_{d'}, \dots, i_{d'-j+1})} = (X^{(1)}, \dots, X^{(\ell_j)})$, and then inductively compute (starting with j = 1)

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

and,

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

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

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 $\lambda \in \mathbb{N}$,

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

$$(13)$$

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,

$$\Pr\left[\begin{array}{cc} \operatorname{Verify}(\operatorname{crs}_{1}, (x_{1}, \dots, x_{k}), \pi) = 1 \land \\ \operatorname{Verify}(\operatorname{crs}_{1}, (x_{1}, \dots, x_{k}), \operatorname{barg}.\pi) \neq 1 \end{array} \right] \stackrel{(\operatorname{crs}, \operatorname{td})}{\underset{w^{*} \leftarrow \operatorname{s-seBARG}.\mathcal{E}(\operatorname{td}_{1}, (x_{1}, \dots, x_{k}), \operatorname{barg}.\pi)}{(x_{1}, \dots, x_{k}), \operatorname{barg}.\pi) \neq 1} \right] (14)$$

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

- 1. On receiving \overline{crs} from the challenger, $\mathcal{B}_{NIZK,\mathcal{E}}$ sets nizk.crs₁ = \overline{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 (14), then $\mathcal{B}_{NIZK,\mathcal{E}}$ breaks straight-line extraction of NIZK with the same probability.

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

$$\Pr\left[\begin{array}{ccc} (\operatorname{crs}, \operatorname{td}) \leftarrow \mathcal{E}(1^{\lambda}, d, (i_{1}, \dots, i_{d})) \\ ((x_{1}, \dots, x_{k}), \pi) = 1 \land & ((x_{1}, \dots, x_{k}), \pi) \leftarrow \mathcal{A}(\operatorname{crs}) \\ \operatorname{Verify}(\operatorname{crs}_{1}, (x_{1}, \dots, x_{k}), \operatorname{barg}. \pi) = 1 & : \operatorname{barg}. \pi \leftarrow \operatorname{NIZK}. \mathcal{E}(\operatorname{nizk}. \operatorname{td}_{1}, (x_{1}, \dots, x_{k}), \pi) \\ \wedge w^{*} \text{ is not a valid witness for } x_{i_{1}} \in \mathcal{L} & w^{*} \leftarrow \operatorname{s-seBARG}. \mathcal{E}(\operatorname{td}_{1}, (x_{1}, \dots, x_{k}), \operatorname{barg}. \pi) \\ & \operatorname{parse \ crs} = (\operatorname{crs}_{1}, \dots, \operatorname{crs}_{d}) \end{array}\right]$$
(15)

Then, we give a reduction $\mathcal{B}_{seBARG.\mathcal{E}}$ 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 $\mathsf{barg.crs}_1 = \overline{\mathsf{crs}}$.
- 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 barg. π .
- 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 (15), then $\mathcal{B}_{seBARG.\mathcal{E}}$ 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 $\lambda \in \mathbb{N}$,

$$\Pr \begin{bmatrix} \operatorname{Verify}((\operatorname{crs}_{j})_{j \in [d']}, X, \pi) = 1 \land & (\operatorname{crs}, \operatorname{td}) \leftarrow \mathcal{E}(1^{\lambda}, d, (i_{1}, \dots, i_{d})) \\ X \text{ is consistent with } T \land & (X, \pi) \leftarrow \mathcal{A}(\operatorname{crs}) \\ (\operatorname{Verify}((\operatorname{crs}_{j})_{j \in [d_{1}]}, X^{(i_{d'})}, \operatorname{barg}.\pi^{(i_{d'})}) \neq 1 \\ \vee \operatorname{Verify}((\operatorname{crs}_{j})_{j \in [d_{1}]}, X^{(i_{d'})}, \pi^{(i_{d'})}) \neq 1 \end{pmatrix} & : \operatorname{barg}.\pi^{(i_{d'})} \leftarrow \operatorname{NIZK}.\mathcal{E}(\operatorname{nizk}.\operatorname{td}_{d'}, X, \pi) \\ \pi^{(i_{d'})} \leftarrow \operatorname{s-seBARG}.\mathcal{E}(\operatorname{td}_{d'}, X, \operatorname{barg}.\pi^{(i_{d'})}) \\ \operatorname{parse } \operatorname{crs} = (\operatorname{crs}_{1}, \dots, \operatorname{crs}_{d}) \end{bmatrix} \leq \nu(\lambda)$$

$$(16)$$

Let us again define two types of possible PPT adversaries:

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

$$\Pr\left[\begin{array}{ccc} (\operatorname{crs}, \operatorname{td}) \leftarrow \mathcal{E}(1^{\lambda}, d, (i_{1}, \dots, i_{d})) \\ (X, \pi) \leftarrow \mathcal{A}(\operatorname{crs}) \\ X \text{ is consistent with } T \land & (X, \pi) \leftarrow \operatorname{A}(\operatorname{crs}) \\ \operatorname{Verify}\left((\operatorname{crs}_{j})_{j \in [d_{1}]}, X^{(i_{d'})}, \operatorname{barg}.\pi^{(i_{d'})}\right) \neq 1 & \pi^{(i_{d'})} \leftarrow \operatorname{s-seBARG}.\mathcal{E}\left(\operatorname{td}_{d'}, X, \operatorname{barg}.\pi^{(i_{d'})}\right) \\ \operatorname{parse \ crs} = (\operatorname{crs}_{1}, \dots, \operatorname{crs}_{d}) \end{array}\right]$$

$$(17)$$

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

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

- 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 (17), then $\mathcal{B}_{NIZK,\mathcal{E}}$ breaks straight-line extraction of NIZK with the same probability.

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

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

- 1. On receiving \overline{crs} from the challenger, $\mathcal{B}_{seBARG.\mathcal{E}}$ sets $barg.crs_{d'} = \overline{crs}$.
- 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 barg. $\pi^{(i_{d'})}$.
- 4. It then outputs $(X, \mathsf{barg}.\pi^{(i_{d'})})$.

It is clear that if the type 2 adversary has non-negligible probability for (18), then $\mathcal{B}_{seBARG.\mathcal{E}}$ breaks somewhere argument of knowledge of the underlying seBARG scheme with the same probability.

Induction step: Now, suppose that Equation (13) 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 $crs = ((nizk.crs_1, barg.crs_1), \dots, (nizk.crs_d, barg.crs_d)).$
- 2. Generate (barg.crs^{*}, barg.td^{*}) \leftarrow s-seBARG. $\mathcal{E}(1^{\lambda}, i_{d'})$ and (nizk.crs^{*}, nizk.td^{*}) \leftarrow NIZK. $\mathcal{E}(1^{\lambda})$.
- 3. Let $\operatorname{crs}^* = ((\operatorname{nizk.crs}_1, \operatorname{barg.crs}_1), \dots, (\operatorname{nizk.crs}_{d'-1}, \operatorname{barg.crs}_{d'-1}), (\operatorname{nizk.crs}^*, \operatorname{barg.crs}^*), (\operatorname{nizk.crs}_{d'+1}, \operatorname{barg.crs}_{d'+1}), \dots, (\operatorname{nizk.crs}_d, \operatorname{barg.crs}_d)).$
- 4. Compute $(X, \pi) \leftarrow \mathcal{A}(\mathsf{crs}^*)$.
- 5. Compute barg. $\pi^{(i_{d'})} \leftarrow \mathsf{NIZK}.\mathcal{E}$ (nizk.td^{*}, X, π) and $\pi^{(i_{d'})} \leftarrow \mathsf{s-seBARG}.\mathcal{E}$ (td_{d'}, X, barg. $\pi^{(i_{d'})}$).
- 6. Output $(X^{(i_{d'})}, \pi^{(i_{d'})})$.

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

$$\Pr\left[\begin{array}{cc} \operatorname{Verify}\left((\operatorname{crs}_{j})_{j\in[d_{1}]}, X^{(i_{d'})}, \pi^{(i_{d'})}\right) = 1 & (\operatorname{crs}, \operatorname{td}) \leftarrow \mathcal{E}(1^{\lambda}, d, (i_{1}, \dots, i_{d})) \\ \wedge X^{(i_{d'})} \text{ is consistent with } T^{(i_{d'})} & : \left(X^{(i_{d'})}, \pi^{(i_{d'})}\right) \leftarrow \mathcal{A}'(\operatorname{crs}) \\ \wedge \operatorname{Verify}\left((\operatorname{crs}_{j})_{j\in[d_{2}]}, X^{(i_{d'}, i_{d'-1})}, \pi^{(i_{d'}, i_{d'-1})}\right) \neq 1 & \pi^{(i_{d'}, i_{d'-1})} \leftarrow \mathcal{E}\left(\operatorname{td}^{*}, X^{(i_{d'})}, \pi^{(i_{d'})}\right) \end{array}\right] \leq \mu(\lambda)$$

This, together with Equation (16), implies the induction step (i.e., Equation (13)).

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

Proof. We define the following hybrids:

- $hyb_0^{\mathcal{A}}$ This is the original zero-knowledge experiment described in Definition 8.2.
- $hyb_1^{\mathcal{A}}$ This is same as hyb_0 except that we use NIZK.S to simulate NIZK instantiation. This is the description of the simulator 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 $negl(\cdot)$ 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 the hardness of LWE, or DLIN, or sub-exponential DDH (and QR) assumptions, 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 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 $\mathsf{crs.}$
- ProgEnc(crs, P) → (c_P , 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.
- Prove(crs, x, aux) $\rightarrow (y, \pi)$. 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 .
- Verify(crs, x, y, c_P, π) \rightarrow {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 zksDel = (Setup, ProgEnc, Prove, 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 $n, m < 2^{\lambda}$,

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

Efficiency. For the completeness experiment above,

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

 $\mathsf{Expt}_0^{\mathcal{C},\mathcal{A}}(1^{\lambda}, n, 1^m, T)$. 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: A sends (x, i) and receives (y, π) where (y, π) ← Prove(crs, x, 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 $c_P \leftarrow \mathcal{S}(crs)$ 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 $(y, \pi) \leftarrow \mathcal{S}(\operatorname{crs}, x)$ and y = P(x) (for P corresponding to i), otherwise, receives \perp .

Figure 3: 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 S such that for any poly-time adversary A, there is a negligible function $negl(\cdot)$ such that for all $\lambda, n, m \in \mathbb{N}$ with $n, m \leq 2^{\lambda}$:

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

where definitions of $\mathsf{Expt}_0^{\mathcal{C},\mathcal{A}}$ and $\mathsf{Expt}_1^{\mathcal{S},\mathcal{A}}$ are provided in Figure 3.

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 $\lambda \in \mathbb{N}$

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

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

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

$$\Pr \begin{bmatrix} \operatorname{Verify}(\operatorname{crs}, x, 0, c_P, \pi^{(0)}) = 1 & (n, m, T) \leftarrow \mathcal{A}(1^{\lambda}), \\ \wedge \operatorname{Verify}(\operatorname{crs}, x, 1, c_P, \pi^{(1)}) = 1 & : \operatorname{crs} \leftarrow \operatorname{Setup}(1^{\lambda}, n_{\mathsf{pb}}, 1^{n_{\mathsf{pr}}}, S, T) \\ (c_P, x, \pi^{(0)}, \pi^{(1)}) \leftarrow \mathcal{A}(\operatorname{crs}) \end{bmatrix} \leq \operatorname{negl}(\lambda)$$

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 $P \in \{0, 1\}^m$,

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

9.2 Construction

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

```
RAM Machine 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 4: RAM Machine for program P

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

ProgEnc(crs, P) \rightarrow (c_P , aux). Generate (eh_P, π_P, aux) \leftarrow r \mathcal{E} Del.PrivEnc(crs, P), where aux = (P, r), and output $c_P = (eh_P, \pi_P)$ and aux.

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

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

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

Theorem 9.2. If $r\mathcal{E}$ 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 $r\mathcal{E}Del$ for \mathcal{M}_P .

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

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

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

Strong Soundness. It follows by construction that if $r\mathcal{E}Del$ 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.
- [ACG24] 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, 2024.
- [ADKL19] Prabhanjan Ananth, Apoorvaa Deshpande, Yael Tauman Kalai, and Anna Lysyanskaya. Fully homomorphic nizk and niwi proofs. In *Theory of Cryptography Confer*ence, pages 356–385. Springer, 2019.
- [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. Springer, Heidelberg, 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.
- [BBK⁺23] Zvika Brakerski, Maya Farber Brodsky, Yael Tauman Kalai, Alex Lombardi, and Omer Paneth. Snargs for monotone policy batch np. In Annual International Cryptology Conference, pages 252–283. Springer, 2023.
- [BCC88] Gilles Brassard, David Chaum, and Claude Crépeau. Minimum disclosure proofs of knowledge. Journal of computer and system sciences, 37(2):156–189, 1988.
- [BCD⁺24] Pedro Branco, Arka Rai Choudhuri, Nico Döttling, Abhishek Jain, Giulio Malavolta, and Akshayaram Srinivasan. Black-box non-interactive zero knowledge from vector trapdoor hash. Cryptology ePrint Archive, Paper 2024/1514, 2024.

- [BDS24] Pedro Branco, Nico Döttling, and Akshayaram Srinivasan. Rate-1 statistical noninteractive zero-knowledge. Cryptology ePrint Archive, Paper 2024/1716, 2024.
- [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.
- [BGI⁺14] Amos Beimel, Ariel Gabizon, Yuval Ishai, Eyal Kushilevitz, Sigurd Meldgaard, and Anat Paskin-Cherniavsky. Non-interactive secure multiparty computation. In Advances in Cryptology–CRYPTO 2014: 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part II 34, pages 387–404. Springer, 2014.
- [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.
- [BKK⁺18] Saikrishna Badrinarayanan, Yael Tauman Kalai, Dakshita Khurana, Amit Sahai, and Daniel Wichs. Succinct delegation for low-space non-deterministic computation. In Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, pages 709–721, 2018.

- [BKM20] Zvika Brakerski, Venkata Koppula, and Tamer Mour. Nizk from lpn and trapdoor hash via correlation intractability for approximable relations. In Annual International Cryptology Conference, pages 738–767. Springer, 2020.
- [BKP⁺23] Nir Bitansky, Chethan Kamath, Omer Paneth, Ron Rothblum, and Prashant Nalini Vasudevan. Batch proofs are statistically hiding. *Cryptology ePrint Archive*, 2023.
- [BL18] Fabrice Benhamouda and Huijia Lin. k-round multiparty computation from kround oblivious transfer via garbled interactive circuits. In Advances in Cryptology– EUROCRYPT 2018: 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29-May 3, 2018 Proceedings, Part II 37, pages 500–532. Springer, 2018.
- [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.
- [BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, pages 62–73, 1993.
- [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. Springer, Heidelberg, March 2011.
- [BW07] Dan Boneh and Brent Waters. Conjunctive, subset, and range queries on encrypted data. In Salil P. Vadhan, editor, *TCC 2007*, volume 4392 of *LNCS*, pages 535–554. Springer, Heidelberg, February 2007.
- [BWW23] Eli Bradley, Brent Waters, and David J Wu. Batch arguments to NIZKs from one-way functions. *Cryptology ePrint Archive*, 2023.
- [CCH⁺19] Ran Canetti, Yilei Chen, Justin Holmgren, Alex Lombardi, Guy N Rothblum, Ron D Rothblum, and Daniel Wichs. Fiat-shamir: from practice to theory. In Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, pages 1082– 1090, 2019.
- [CG24] Jiaqi Cheng and Rishab Goyal. Boosting SNARKs and rate-1 barrier in arguments of knowledge. Cryptology ePrint Archive, Paper 2024/1603, 2024.
- [CGH04] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. *Journal of the ACM (JACM)*, 51(4):557–594, 2004.
- [CGKS23] Matteo Campanelli, Chaya Ganesh, Hamidreza Khoshakhlagh, and Janno Siim. Impossibilities in succinct arguments: Black-box extraction and more. In Nadia El Mrabet, Luca De Feo, and Sylvain Duquesne, editors, AFRICACRYPT 23, volume 14064 of LNCS, pages 465–489. Springer Nature, July 2023.
- [CHK03] Ran Canetti, Shai Halevi, and Jonathan Katz. A forward-secure public-key encryption scheme. 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 255–271. Springer, 2003.
- [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.
- [CJJ22a] Arka Rai Choudhuri, Abhishek Jain, and Zhengzhong Jin. SNARGs for \mathcal{P} from LWE. In *62nd FOCS*, pages 68–79. IEEE Computer Society Press, February 2022.
- [CJJ22b] Arka Rai Choudhuri, Abhishek Jain, and Zhengzhong Jin. SNARGs for *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.
- [CJJQ23] Geoffroy Couteau, Abhishek Jain, Zhengzhong Jin, and Willy Quach. A note on noninteractive zero-knowledge from CDH. In Helena Handschuh and Anna Lysyanskaya, editors, CRYPTO 2023, Part IV, volume 14084 of LNCS, pages 731–764. Springer, Heidelberg, August 2023.
- [CW23] Jeffrey Champion and David J Wu. Non-interactive zero-knowledge from noninteractive batch arguments. *Cryptology ePrint Archive*, 2023.
- [DGKV22] Lalita Devadas, Rishab Goyal, Yael Kalai, and Vinod Vaikuntanathan. Rate-1 noninteractive arguments for batch-np and applications. In 2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS), pages 1057–1068. IEEE, 2022.
- [DGMR21] Nico Döttling, Vipul Goyal, Giulio Malavolta, and Justin Raizes. Interactionpreserving compilers for secure computation. *Cryptology ePrint Archive*, 2021.
- [DJJ24] Quang Dao, Aayush Jain, and Zhengzhong Jin. Non-interactive zero-knowledge from LPN and MQ. In Leonid Reyzin and Douglas Stebila, editors, Advances in Cryptology - CRYPTO 2024, Part IX, volume 14928 of Lecture Notes in Computer Science, pages 321–360, Santa Barbara, CA, USA, August 18–22, 2024. Springer, Cham, Switzerland.
- [FLS99] Uriel Feige, Dror Lapidot, and Adi Shamir. Multiple noninteractive zero knowledge proofs under general assumptions. *SIAM Journal on computing*, 29(1):1–28, 1999.
- [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.
- [FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Conference on the theory and application of cryptographic techniques*, pages 186–194. Springer, 1986.
- [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.
- [GKP⁺13] Shafi Goldwasser, Yael Tauman Kalai, Raluca Ada Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. How to run turing machines on encrypted data. In Advances in Cryptology–CRYPTO 2013: 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II, pages 536–553. Springer, 2013.
- [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.
- [GO94] Oded Goldreich and Yair Oren. Definitions and properties of zero-knowledge proof systems. *Journal of Cryptology*, 7(1):1–32, 1994.
- [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.
- [GOS12] Jens Groth, Rafail Ostrovsky, and Amit Sahai. New techniques for noninteractive zero-knowledge. J. ACM, 59(3), jun 2012.
- [GPSW06] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, ACM CCS 2006, pages 89– 98. ACM Press, October / November 2006. Available as Cryptology ePrint Archive Report 2006/309.
- [GS17] Sanjam Garg and Akshayaram Srinivasan. Garbled protocols and two-round mpc from bilinear maps. In 2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS), pages 588–599. IEEE, 2017.
- [GS22] Sanjam Garg and Akshayaram Srinivasan. Two-round multiparty secure computation from minimal assumptions. *Journal of the ACM*, 69(5):1–30, 2022.
- [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. Springer, Heidelberg, 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. Springer, Heidelberg, August 2015.

- [GVW15b] Sergey Gorbunov, Vinod Vaikuntanathan, and Daniel Wichs. Leveled fully homomorphic signatures from standard lattices. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 469–477, 2015.
- [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.
- [IKO⁺11] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, Manoj Prabhakaran, and Amit Sahai. Efficient non-interactive secure computation. 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 406–425. Springer, 2011.
- [JJ21] Abhishek Jain and Zhengzhong Jin. Non-interactive zero knowledge from subexponential ddh. In Annual International Conference on the Theory and Applications of Cryptographic Techniques, pages 3–32. Springer, 2021.
- [JKKZ21] Ruta Jawale, Yael Tauman Kalai, Dakshita Khurana, and Rachel Zhang. Snargs for bounded depth computations and ppad hardness from sub-exponential lwe. In Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing, pages 708–721, 2021.
- [JKLV24] Zhengzhong Jin, Yael Kalai, Alex Lombardi, and Vinod Vaikuntanathan. Snargs under lwe via propositional proofs. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing*, pages 1750–1757, 2024.
- [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.
- [Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In Proceedings of the Twenty-Fourth Annual ACM Symposium on Theory of Computing, STOC '92, page 723–732, New York, NY, USA, 1992. Association for Computing Machinery.
- [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.
- [KMY23] Fuyuki Kitagawa, Takahiro Matsuda, and Takashi Yamakawa. Nizk from snargs. Journal of Cryptology, 36(2):14, 2023.
- [KPY19] Yael Tauman Kalai, Omer Paneth, and Lisa Yang. How to delegate computations publicly. In Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, pages 1115–1124, 2019.

- [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.
- [KSW08] Jonathan Katz, Amit Sahai, and Brent Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In Nigel P. Smart, editor, EUROCRYPT 2008, volume 4965 of LNCS, pages 146–162. Springer, Heidelberg, April 2008.
- [KVZ21] Yael Tauman Kalai, Vinod Vaikuntanathan, and Rachel Yun Zhang. Somewhere statistical soundness, post-quantum security, and snargs. In *Theory of Cryptography Conference*, pages 330–368. Springer, 2021.
- [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.
- [MPR11] Hemanta K Maji, Manoj Prabhakaran, and Mike Rosulek. Attribute-based signatures. In *Cryptographers' track at the RSA conference*, pages 376–392. Springer, 2011.
- [NWW23] Shafik Nassar, Brent Waters, and David J Wu. Monotone policy bargs from bargs and additively homomorphic encryption. *Cryptology ePrint Archive*, 2023.
- [NY90] Moni Naor and Moti Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 427–437, 1990.
- [OPWW15] Tatsuaki Okamoto, Krzysztof Pietrzak, Brent Waters, and Daniel Wichs. New realizations of somewhere statistically binding hashing and positional accumulators. In Tetsu Iwata and Jung Hee Cheon, editors, ASIACRYPT 2015, Part I, volume 9452 of LNCS, pages 121–145. Springer, Heidelberg, November / December 2015.
- [PP22] Omer Paneth and Rafael Pass. Incrementally verifiable computation via rate-1 batch arguments. In 2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS), pages 1045–1056. IEEE, 2022.
- [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.
- [PS19] Chris Peikert and Sina Shiehian. Noninteractive zero knowledge for np from (plain) learning with errors. In Annual International Cryptology Conference, pages 89–114. Springer, 2019.

- [QRW19] Willy Quach, Ron D Rothblum, and Daniel Wichs. Reusable designated-verifier nizks for all np from cdh. In Advances in Cryptology-EUROCRYPT 2019: 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19–23, 2019, Proceedings, Part II 38, pages 593–621. Springer, 2019.
- [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.
- [Reg09] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. Journal of the ACM (JACM), 56(6):1–40, 2009.
- [Shi22] Sina Shiehian. mrnisc from lwe with polynomial modulus. In International Conference on Security and Cryptography for Networks, pages 481–493. Springer, 2022.
- [SW05] Amit Sahai and Brent Waters. Fuzzy identity-based encryption. In Advances in Cryptology-EUROCRYPT 2005: 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005. Proceedings 24, pages 457–473. Springer, 2005.
- [SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pages 475–484, 2014.
- [Wat24] Brent Waters. A new approach for non-interactive zero-knowledge from learning with errors. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing*, pages 399–410, 2024.
- [WW22] Brent Waters and David J Wu. Batch arguments for np and more from standard bilinear group assumptions. In Annual International Cryptology Conference, pages 433–463. Springer, 2022.
- [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:

- $\operatorname{\mathsf{Gen}}(1^{\lambda}, N, I) \to (\mathsf{hk}, \mathsf{td})$. 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 $\rho \in \{0, 1\}^{\leq |I| \cdot |S| \cdot \mathsf{poly}(\lambda, \log N)}$.
- Verify(hk, v, $S, (b_j)_{j \in S}, \rho) \to 0/1$. This is a deterministic algorithm that takes as input a hash key hk generated by $\text{Gen}(1^{\lambda}, N, I)$, a hash value $v \in \{0, 1\}^{|I| \cdot \text{poly}(\lambda, \log N)}$, set of indices $S \subseteq [N]$, a sequence of bits b_j for $j \in S$ and an opening $\rho \in \{0, 1\}^{\leq |I| \cdot |S| \cdot \text{poly}(\lambda, \log N)}$, and outputs 0 or 1.
- Extract(td, v) $\rightarrow (b_j^*)_{j \in I}$. This is a deterministic extraction algorithm that takes as input the trapdoor td generated by $\text{Gen}(1^{\lambda}, N, I)$, a hash value $v \in \{0, 1\}^{|I| \cdot \text{poly}(\lambda, \log N)}$, and outputs extracted sequence of bits $(b_j)_{j \in I}$.

Definition A.1 (SEH). A somewhere extractable hash family SEH = (Gen, Hash, Open, Verify, Extract) is required to satisfy the following properties:

Efficiency. The size of the hash key hk and the hash value v is at most $|I| \cdot poly(\lambda, \log N)$.

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 $x \in \{0, 1\}^N$,

$$\Pr\left[\begin{array}{ll}\forall \ j \in S, b_j = x_j & (\mathsf{hk}, \mathsf{td}) \leftarrow \mathsf{Gen}(1^{\lambda}, N, I), \\ \wedge \operatorname{Verify}(\mathsf{hk}, \mathsf{v}, S, (b_j)_{j \in S}, \rho) = 1 & : & \mathsf{v} = \mathsf{Hash}(\mathsf{hk}, x), \\ & ((b_j)_{j \in S}, \rho) = \mathsf{Open}(\mathsf{hk}, x, S), \end{array}\right] = 1.$$

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 $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{cc} \mathsf{Verify}(\mathsf{hk},\mathsf{v},I,(b_j)_{j\in I},\rho) = 1 \\ \wedge b_j \neq b_j^*, \text{ for any } j \in I \end{array} : \begin{array}{c} (\mathsf{hk},\mathsf{td}) \leftarrow \mathsf{Gen}(1^\lambda,N,I), \\ (\mathsf{v},(b_j)_{j\in I},\rho) \leftarrow \mathcal{A}(\mathsf{hk}), \\ (b_j^*)_{j\in I} = \mathsf{Extract}(\mathsf{td},\mathsf{v}) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

Remark A.2 ([KLVW23, HW15, OPWW15]). Assuming the hardness of QR/DCR/DDH/LWE/k-LIN, there exists an SEH family.

A.2 Rate-1 Multi-Key Fully Homomorphic Encryption

We provide the syntax and definitions for a multi-key fully homomorphic encryption (mkFHE) for message space $\mathcal{M} = \{\mathcal{M}_{\lambda}\}_{\lambda}$ 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.
- $KeyGen(PP) \rightarrow (pk, sk)$. The probabilistic key generation algorithm takes as input the public parameters PP and outputs the public and secret key pair (pk, sk).
- $\mathsf{Enc}(\mathsf{pk}, m) \to \mathsf{ct}$. The probabilistic encryption algorithm takes as input the public key pk , (multibit) message $m \in \mathcal{M}_{\lambda}$, and outputs the ciphertext ct .
- $\mathsf{Eval}(\mathsf{pk}_1, \dots, \mathsf{pk}_n, f, \mathsf{ct}_1, \dots, \mathsf{ct}_\ell) \to \widetilde{\mathsf{ct}}$. The probabilistic evaluation algorithm takes as input the public keys for n parties $\{\mathsf{pk}_{\mathsf{id}}\}_{\mathsf{id}\in[n]}$, the description of the function f, ciphertexts $\{\mathsf{ct}_i\}_{i\in[\ell]}$, 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.
- $\mathsf{PartDec}(\mathsf{id},\mathsf{sk}_{\mathsf{id}},\widetilde{\mathsf{ct}}) \to \mathsf{sh.}$ The deterministic partial decryption algorithm takes as input the secret key of the id-th party, $\mathsf{sk}_{\mathsf{id}}$, evaluated ciphertext $\widetilde{\mathsf{ct}}$, and outputs decryption share $\mathsf{sh.}$
- $\mathsf{Recon}(\{\mathsf{sh}_{\mathsf{id}}\}_{\mathsf{id}\in[n]}) \to y$. The deterministic reconstruction algorithm takes as input decryption shares of all the parties and outputs the value y.

Definition A.3 (mkFHE). An mkFHE scheme (Setup, KeyGen, Enc, Eval, Dec, PartDec, Recon) for message space \mathcal{M}_{λ} , is required to satisfy the following properties:

Multi-Hop Correctness. For any $\lambda \in \mathbb{N}$, $n = n(\lambda)$, $x, x_1, \ldots, x_n \in \mathcal{M}_{\lambda}$, and *n*-ary P/Poly circuit f:

$$\Pr\left[\begin{array}{cc} \mathsf{PP} \leftarrow \mathsf{Setup}(1^{\lambda}), \\ x = \mathsf{Dec}(\mathsf{sk},\mathsf{ct}) & : & (\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{PP}), \\ & & \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk},x) \end{array}\right] = 1$$

$$\Pr\left[\begin{array}{cc} \mathsf{PP} \leftarrow \mathsf{Setup}(1^{\lambda}), \\ f(x_1, \dots, x_n) = & \\ \mathsf{Dec}(\mathsf{sk}_1, \dots, \mathsf{sk}_n, \widetilde{\mathsf{ct}}) \end{array} : \begin{array}{c} \mathsf{PP} \leftarrow \mathsf{Setup}(1^{\lambda}), \\ \forall \mathsf{id} \in [n], (\mathsf{pk}_{\mathsf{id}}, \mathsf{sk}_{\mathsf{id}}) \leftarrow \mathsf{KeyGen}(\mathsf{PP}), \\ \forall \mathsf{id} \in [\ell], \mathsf{ct}_{\mathsf{id}} \leftarrow \mathsf{Enc}(\mathsf{pk}_{\mathsf{id}}, x_{\mathsf{id}}), \\ \widetilde{\mathsf{ct}} = \mathsf{Eval}(\mathsf{pk}_1, \dots, \mathsf{pk}_n, f, \mathsf{ct}_1, \dots, \mathsf{ct}_\ell) \end{array}\right] = 1$$

Efficiency. For any $\lambda \in N$, $n = n(\lambda)$, $x_1, \ldots, x_n \in \mathcal{M}_{\lambda}$, and *n*-ary P/Poly circuit f:

$$\frac{|f(x_1,\ldots,x_n)|}{|\widetilde{\mathsf{ct}}|} = 1 - o(1)$$

where $\mathsf{PP} \leftarrow \mathsf{Setup}(1^{\lambda}), \forall \mathsf{id} \in [n], (\mathsf{pk}_{\mathsf{id}}, \mathsf{sk}_{\mathsf{id}}) \leftarrow \mathsf{KeyGen}(\mathsf{PP}), \mathsf{ct}_{\mathsf{id}} \leftarrow \mathsf{Enc}(\mathsf{pk}_{\mathsf{id}}, x_{\mathsf{id}}), \widetilde{\mathsf{ct}} = \mathsf{Eval}(\mathsf{pk}_1, \dots, \mathsf{pk}_n, f, \mathsf{ct}_1, \dots, \mathsf{ct}_\ell).$

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

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

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

Threshold Decryption. There exists a PPT simulator S such that for any $H \subseteq [n]$, |H| < n, $\forall \lambda \in \mathbb{N}$,

{PartDec(id, sk_{id}, \widetilde{ct})}_{id \in H} \approx_s S(y, \widetilde{ct}, H, \{sk_{id}\}_{id \in [n] \setminus H})

where \approx_s denotes statistical indistinguishability.

Remark A.4 ([DGMR21]). Assuming the hardness of LWE with sub-exponential modulus-tonoise ratio and circular security, there exists a rate-1 mkFHE scheme for all P/Poly circuits and $\mathcal{M} = \{0, 1\}^*$.

B RAM Delegation with Split Configuration

In this section, we state and provide a short proof of a property of RAM Delegation that [CJJ22b, KLVW23] satisfies implicitly. We show that the RAM delegation of [CJJ22b, KLVW23] can be done when the configuration 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 configuration 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 HT.Hash (ht.hk, 0^{*}), the output value $h = 0^{\lambda+1}$. 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 $\mathsf{h} = 1 \parallel \mathsf{HT.Hash}(\mathsf{ht.hk}, x)$.

As remarked in [CG24], 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 configuration 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 2.17 which splits the configuration into n parts below.

B.1 RAM Delegation Construction

We provide the construction of RAM delegation in accordance with Definition 2.17 as follows. We remark that $\tilde{\mathcal{L}}$ (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. [CJJ22b], 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 (RAM Delegation). Let SEH = (Gen, Hash, Open, Verify, Extract) be a somewhere extractable hash family (Definition A.1), HT = (Gen, Hash, Open, Write, VerifyRead, VerifyWrite)be a hash tree, seBARG = (Setup, Prove, Verify) be a (publicly-verifiable, non-interactive) batch argument scheme for the language $\tilde{\mathcal{L}} = \{C^{\mathcal{M}} : \exists w_i \text{ such that } C^{\mathcal{M}}(i, w_i) = 1, \forall i \in [k]\}$ where $C^{\mathcal{M}}$ is defined in Figure 5. 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, seh.hk⁰, seh.hk¹, ht.hk, h₀^{*}, v⁰, v¹, out Instance: iWitness: $w_i = (\widehat{w}_{i-1}, \rho_{i-1}^0, \rho_{i-1}^1, \widehat{w}_i, \rho_i^0, \rho_i^1, b_{rd,i}, \ell_{rd,i}, j_i, u_{rd,i}, op_{rd,i}, b_{wr,i}, \ell_{wr,i}, j'_i, u_{wr,i}, op_{wr,i})$ Output: Output 1 if and only if all of these checks pass.

- 1. Parse $\hat{w}_{i-1} = (\mathsf{st}_{i-1}, \mathsf{h}_{i-1})$, and $\hat{w}_i = (\mathsf{st}_i, \mathsf{h}_i)$.
- 2. If i = 1, check that st_0 is the initial state. Also, check $h_0 = h_0^*$.
- 3. If i = T, check that st_T is accepting/rejecting state based on out = 1/0 respectively.
- 4. SEH.Verify (seh.hk⁰, v⁰, \mathbb{I}_{i-1} , \widehat{w}_{i-1} , ρ_{i-1}^{0}) $\stackrel{?}{=} 1$ and SEH.Verify (seh.hk¹, v¹, \mathbb{I}_{i-1} , \widehat{w}_{i-1} , ρ_{i-1}^{1}) $\stackrel{?}{=} 1$.
- 5. SEH.Verify (seh.hk⁰, v⁰, \mathbb{I}_i , \widehat{w}_i , ρ_i^0) $\stackrel{?}{=} 1$ and SEH.Verify (seh.hk¹, v¹, \mathbb{I}_i , \widehat{w}_i , ρ_i^1) $\stackrel{?}{=} 1$.
- 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.
$$u_{\mathsf{rd},\mathsf{i}} \stackrel{?}{=} \ell_{\mathsf{rd},\mathsf{i}} - \left(S^{(1)} + \ldots + S^{(j_i-1)}\right)$$
 and $u_{\mathsf{wr},\mathsf{i}} \stackrel{?}{=} \ell_{\mathsf{wr},\mathsf{i}} - \left(S^{(1)} + \ldots + S^{(j'_i-1)}\right)$

8. HT.VerifyRead(ht.hk, $h_{i-1}^{(j_i)}, u_{rd,i}, b_{rd,i}, op_{rd,i}) \stackrel{?}{=} 1$.

9. HT.VerifyWrite(ht.hk, $\mathbf{h}_{i-1}^{(j'_i)}, u_{\text{wr},i}, \mathbf{h}_{i}^{(j'_i)}, \operatorname{op}_{\text{wr},i}) \stackrel{?}{=} 1.$

Figure 5: Circuit for verifying RAM configuration transition.

- Setup $(1^{\lambda}, T, S) \to \text{crs.}$ The probabilistic setup algorithm takes as input the security parameter, a vector of split sizes $S = (S^{(1)}, \ldots, S^{(n)})$, running time T for machine \mathcal{M} , and does the following.
 - 1. Sample two SEH hash keys, seh.hk⁰, seh.hk¹ \leftarrow SEH.Gen $(1^{\lambda}, m \cdot (T+1), \mathbb{I}_1)$ where $m = |\widehat{w}_i|$ (later specified in Prove) and $\mathbb{I}_i = \{(i-1) \cdot m + 1, \dots, i \cdot m\}$ for $i \in [T]$.
 - 2. Sample crs for seBARG, barg.crs \leftarrow seBARG.Setup $(1^{\lambda}, T, 1^{|C^{\mathcal{M}}|}, 1)$.
 - 3. Sample a HT hash key, $\mathsf{ht.hk} \leftarrow \mathsf{HT.Gen}(1^{\lambda})$.

Output $crs = (T, S, seh.hk^0, seh.hk^1, barg.crs, ht.hk)$.

- Digest(crs, x) \rightarrow h. The deterministic digest algorithm parses crs, uses ht.hk, parses x as $(x^{(1)}, \ldots, x^{(n)})$, such that $|x^{(j)}| = S^{(j)}$ for each $j \in [n]$, computes $h^{(j)} = HT.Hash(ht.hk, x^{(j)})$, and outputs $h = (h^{(1)}, \ldots, h^{(n)})$.
- $Prove(crs, cf_0)$ → (out, π). The probabilistic prover algorithm parses crs as (T, S, seh.hk⁰, seh.hk¹, barg.crs, 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 $h_i = \text{Digest}(\text{crs}, \text{cf}_i)$.
 - 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 $u_{rd,i}$ denote the index of $\ell_{rd,i}$ in the partition j_i . That is $u_{rd,i} = \ell_{rd,i} (S^{(1)} + \ldots + S^{(j_i-1)})$. Similarly, calculate $u_{wr,i}$.
 - Compute $(b_{\mathsf{rd},i}, \mathsf{op}_{\mathsf{rd},i}) = \mathsf{HT}.\mathsf{Open}\left(\mathsf{ht}.\mathsf{hk}, x_{i-1}^{(j_i)}, u_{\mathsf{rd},i}\right)$ and $\left(\widetilde{\mathsf{h}}_i^{(j_i')}, \mathsf{op}_{\mathsf{wr},i}\right) = \mathsf{HT}.\mathsf{Write}\left(\mathsf{ht}.\mathsf{hk}, x_{i-1}^{(j_i')}, u_{\mathsf{wr},i}, b_{\mathsf{wr},i}\right)$ 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 $v^0 = \mathsf{SEH}.\mathsf{Hash}(\mathsf{seh}.\mathsf{hk}^0, \widehat{\mathbf{w}}), v^1 = \mathsf{SEH}.\mathsf{Hash}(\mathsf{seh}.\mathsf{hk}^1, \widehat{\mathbf{w}}).$
 - 6. For $i \in [0, T]$, let $(\widehat{w}_i, \rho_i^0) = \mathsf{SEH.Open}(\mathsf{seh.hk}^0, \widehat{\mathbf{w}}, \mathbb{I}_i), (\widehat{w}_i, \rho_i^1) = \mathsf{SEH.Open}(\mathsf{seh.hk}^1, \widehat{\mathbf{w}}, \mathbb{I}_i).$
 - 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 barg. $\pi \leftarrow$ seBARG.Prove (barg.crs, $C^{\mathcal{M}}, (w_i)_{i \in [T]}$).
 - 9. Set $\pi := (v^0, v^1, \mathsf{barg}.\pi)$ and output (out, π) .
- Verify(crs, h_0^* , out, π) $\rightarrow 0/1$. The deterministic verification algorithm parses crs as $(T, S, \text{seh.hk}^0, \text{seh.hk}^1, \text{barg.crs}, \text{ht.hk})$, parses π as $(v^0, v^1, \text{barg.}\pi)$, and outputs seBARG.Verify (barg.crs, $C^{\mathcal{M}}, \text{barg.}\pi)$.

Theorem B.2. If SEH is a secure somewhere extractable hash family (Definition A.1), HT is a sound hash tree (Definition 2.6), and seBARG is a secure somewhere extractable batch argument scheme for language $\tilde{\mathcal{L}}$ (Definition 2.1), 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 HT, SEH, seBARG are efficient, the RAM delegation scheme satisfies the following properties:

- 1. Sizes of the crs and proof π are $poly(\lambda, n, \log T)$.
- 2. Running time of Setup is $poly(\lambda, n, \log T)$.
- 3. Running time of Digest is $poly(n, \lambda)$ and the size of digest is $poly(n, \lambda)$.
- 4. Running time of Prove is $poly(\lambda, T, n)$.
- 5. Running time of Verify is $poly(\lambda, n, \log T)$.

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 cf, 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 [CJJ22b, 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 2.17 for $T = T(\lambda), n = n(\lambda), S(\lambda) = S = (S^{(1)}, \ldots, S^{(n)})$. In other words,

$$\Pr\left[\begin{array}{cc}\forall \ b \in \{0,1\}, \mathsf{seBARG.Verify}(\mathsf{barg.crs}, C_b, \mathsf{barg.}\pi_b) = 1 & : & \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, T, S), \\ (\mathsf{h}_0, \pi_0, \pi_1) \leftarrow \mathcal{A}(\mathsf{crs}) \end{array}\right] = \epsilon(\lambda)$$

where C_b is $C^{\mathcal{M}}$ (Figure 5) 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 $\mathsf{seBARG.Setup}, \beta$ -th $\mathsf{SEH.Gen}$, and $(1 - \beta)$ -th SEH.Gen respectively where $\beta = (k - 1) \mod 2$. By index-hiding property of SEH, seBARG, there exists a negligible function $\mu(\cdot)$ such that for any PPT adversary $\mathcal{A}, \lambda \in \mathbb{N}$ and $k \in [T]$,

$$\Pr\left[\begin{array}{ll}\forall \ b \in \{0,1\},\\ \mathsf{seBARG.Verify}(\mathsf{barg.crs}, C_b, \mathsf{barg.}\pi_b) = 1\end{array} \right] \quad \begin{array}{ll} \mathsf{crs} \leftarrow \mathsf{Setup}_k(1^\lambda, T, S),\\ (\mathsf{h}_0, \pi_0, \pi_1) \leftarrow \mathcal{A}(\mathsf{crs})\end{array}\right] \geq \epsilon(\lambda) - \mu(\lambda)$$

Now, we proceed to inductively argue for k that there exists a negligible function $\zeta(\cdot)$ such that for any PPT adversary $\mathcal{A}, k \in [T], \lambda \in \mathbb{N}$,

$$\Pr \begin{bmatrix} \forall b \in \{0,1\}, & \operatorname{crs} \leftarrow \operatorname{Setup}_{k}(1^{\lambda}, T, S), \\ \operatorname{seBARG.Verify} & (h_{0}, \pi_{0}, \pi_{1}) \leftarrow \mathcal{A}(\operatorname{crs}) \\ (\operatorname{barg.crs}, C_{b}, \operatorname{barg.}\pi_{b}) = 1 \wedge & \forall b \in \{0,1\}, \\ \widehat{w}_{0,k} = \widehat{w}_{1,k} & \widehat{w}_{b,k} \leftarrow \operatorname{SEH.Extract}(\operatorname{seh.td}^{\beta}, v_{b}^{\beta}) \end{bmatrix} \geq \epsilon(\lambda) - k \cdot \zeta(\lambda) \quad (19)$$

where $\beta \equiv (k-1) \mod 2$. When k = T, Equation 19 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 19 holds for k. Now, we will switch to using 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 $\mathcal{A}, k \in [T], \lambda \in \mathbb{N}$,

$$\Pr \begin{bmatrix} \widehat{w}_{0,k} = \widehat{w}_{1,k} \land & \mathsf{crs} \leftarrow \mathsf{Setup}_k(1^\lambda, T, S), \\ \forall \ b \in \{0,1\}, & (\mathsf{h}_0, \pi_0, \pi_1) \leftarrow \mathcal{A}(\mathsf{crs}) \\ \mathsf{seBARG.Verify} & : \forall \ b \in \{0,1\}, \\ (\mathsf{barg.crs}, C_b, \mathsf{barg.}\pi_b) = 1 & w_{b,k} \leftarrow \mathsf{seBARG.}\mathcal{E}(\mathsf{barg.td}, C_b, \mathsf{barg.}\pi_b), \\ \land \ C_b(k, w_{b,k}) = 1 & \widehat{w}_{b,k} \leftarrow \mathsf{SEH.Extract}(\mathsf{seh.td}^\beta, v_b^\beta) \\ \geq \epsilon(\lambda) - k \cdot \zeta(\lambda) - \nu(\lambda) \end{bmatrix}$$

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 $\lambda \in \mathbb{N}$,

$$\Pr \begin{bmatrix} \widehat{w}_{0,k} = \widehat{w}_{1,k} \land & \operatorname{crs} \leftarrow \operatorname{Setup}_{k}(1^{\lambda}, T, S), \\ \forall \ b \in \{0, 1\}, & (h_{0}, \pi_{0}, \pi_{1}) \leftarrow \mathcal{A}(\operatorname{crs}) \\ \operatorname{seBARG.Verify} & \forall \ b \in \{0, 1\}, \\ (\operatorname{barg.crs}, C_{b}, \operatorname{barg.} \pi_{b}) = 1 & : \\ \land \ C_{b}(k, w_{b,k}) = 1 \land & \\ \widehat{w}_{b,k} = (\operatorname{st}_{b,k}, \operatorname{h}_{b,k}) \land & \\ \widehat{w}_{b,k+1} = (\operatorname{st}_{b,k+1}, \operatorname{h}_{b,k+1}) & \widehat{w}_{b,k+1} \leftarrow \operatorname{SEH.Extract}(\operatorname{seh.td}^{\beta}, v_{b}^{\beta}), \\ \widehat{w}_{b,k+1} \leftarrow \operatorname{SEH.Extract}(\operatorname{seh.td}^{1-\beta}, v_{b}^{1-\beta}) \end{bmatrix}$$

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 $\mathcal{A}, \lambda \in \mathbb{N}$,

$$\Pr \left[\begin{array}{c} \mathsf{crs} \leftarrow \mathsf{Setup}_k(1^\lambda, T, S), \\ (\mathsf{h}_0, \pi_0, \pi_1) \leftarrow \mathcal{A}(\mathsf{crs}) \\ \forall \ b \in \{0, 1\}, \\ \mathsf{seBARG.Verify} \\ (\mathsf{barg.crs}, C_b, \mathsf{barg.}\pi_b) = 1 \\ \varepsilon(\lambda) - k \cdot \zeta(\lambda) - \nu(\lambda) - \nu_1(\lambda) - \nu_2(\lambda) \end{array} \right]$$

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 $\mathcal{A}, \lambda \in \mathbb{N}$,

$$\Pr \begin{bmatrix} \widehat{w}_{0,k+1} = \widehat{w}_{1,k+1} \land & \mathsf{crs} \leftarrow \mathsf{Setup}_{k+1}(1^{\lambda}, T, S), \\ \forall \ b \in \{0, 1\}, & : & (\mathsf{h}_0, \pi_0, \pi_1) \leftarrow \mathcal{A}(\mathsf{crs}) \\ \mathsf{seBARG.Verify} & : & \forall \ b \in \{0, 1\}, \\ (\mathsf{barg.crs}, C_b, \mathsf{barg.}\pi_b) = 1 & \widehat{w}_{b,k+1} \leftarrow \mathsf{SEH.Extract}(\mathsf{seh.td}^{1-\beta}, v_b^{1-\beta}) \\ \geq \epsilon(\lambda) - k \cdot \zeta(\lambda) - \nu(\lambda) - \nu_1(\lambda) - \nu_2(\lambda) - \nu_3(\lambda) \end{bmatrix}$$

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.