Non-Interactive Publicly-Verifiable Delegation of Committed Programs

Riddhi Ghosal
UCLA
riddhi@cs.ucla.edu

Amit Sahai
UCLA
sahai@cs.ucla.edu

Brent Waters
UT Austin and NTT Research
bwaters@cs.utexas.edu

Abstract

In this work, we present the first construction of a fully non-interactive publicly-verifiable delegation scheme for committed programs. More specifically, we consider a setting where Alice is a trusted author who delegates to an untrusted worker the task of hosting a program P, represented as a Boolean circuit. Alice also commits to a succinct value based on P. Any arbitrary user/verifier *without knowledge of* P should be convinced that they are receiving from the worker an actual computation of Alice's program on a given input x.

Before our work, the only object known to imply this challenging form of delegation was a SNARG/SNARK for $\mathcal{NP}$. This is because from the point of view of the user/verifier, the program P is an unknown witness to the computation. However, constructing a SNARG for $\mathcal{NP}$ from standard assumptions remains a major open problem.

In our work, we show how to achieve delegation in this challenging context assuming only the hardness of the Learning With Errors (LWE) assumption, bypassing the apparent need for a SNARG for $\mathcal{NP}$.

1 Introduction

We consider a scenario where a trusted software author Alice wishes to make it possible for a set of users to make use of her program P, which we treat as a (non-uniform) Boolean circuit. In particular, this program P may have embedded within it a large proprietary database that Alice's program makes use of. However, Alice neither wants to release her program P nor does she want to host and execute the program herself. Instead she wishes to delegate this computation to an untrusted Worker, and the User/Verifier wants to be certain that they are receiving an output obtained via a computation of Alice's actual program P. As illustrated in Figure 1, the way this works is:

1. Alice sends the program P along with some computed state to the Worker, and Alice also publishes a succinct hash H_P of her program, which the User/Verifier obtains. This step is done once and for all.
2. An Input Provider chooses an input x, which is sent to both the Worker and the User/Verifier. Note that the input provider could be some public source of information like a news channel of bulletin board, and need not involve the User/Verifier.
3. Finally, the Worker computes the output $y = P(x)$ along with a succinct proof Π, and sends both of these to the User/Verifier. Steps 2 and 3 may be repeated polynomially many times.

As illustrated in Figure 1, this process involves no back–and–forth communication. The communication is entirely unidirectional – which we call non-interactive – from left to right. Furthermore, we say that this scenario is *succinct* if all communication to the User/Verifier, and the runtime of the User/Verifier, is $\mathsf{poly}(\log|P|, \lambda, |x|)$, where λ is a security parameter.

Remark 1.1. Note that on one hand, the Worker is trusted with the program P by Alice, whereas, it is not trusted by the verifier. This asymmetry of trust is inherent in our setup and is well motivated. In a typical real world situation, the verifier is typically a user on the internet who takes part in a one off interaction with a cloud service for some computation. The need to prove honesty in this situation is significant. On

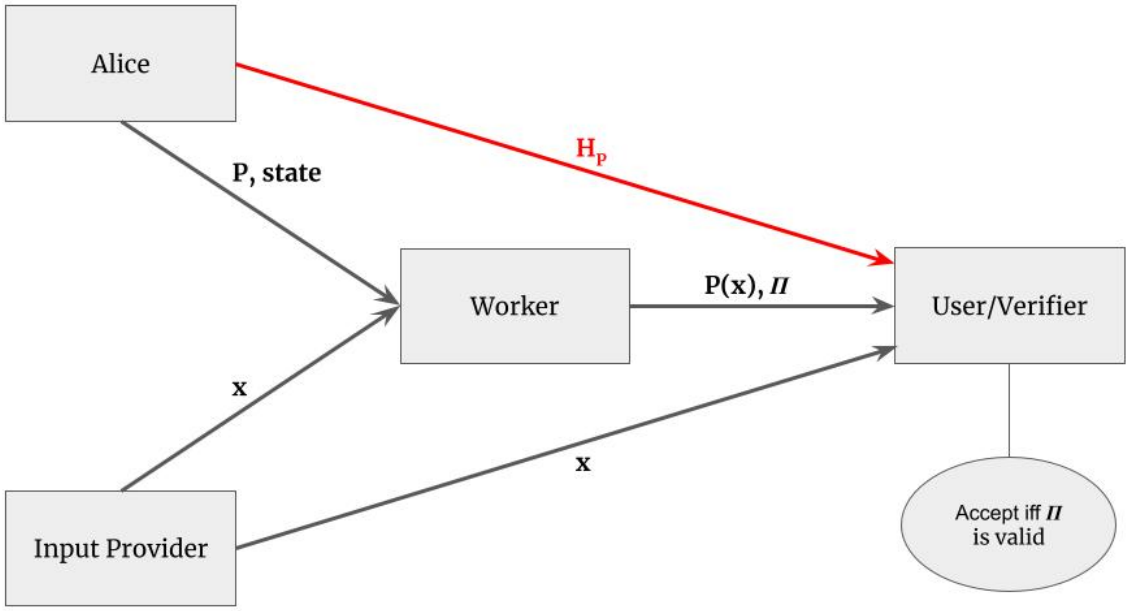

Figure 1: The Delegation Setup

the other hand, Alice might be able to have an agreement with the cloud service before handing over her program, which would make it hard for their Worker to breach trust without consequences.

Comparison to Prior Work. What we have just described is one of the most challenging variants of the classical problem of *publicly verifiable delegation* which has been the subject of intense work for decades, for many relaxed variations of the model that we describe above.

Specifically, delegation schemes *without public verification* based on standard assumptions for deterministic and non-deterministic computations have been designed [TRMP12, CMT12, SMBW12, SVP+12, CKV10, KP16, TKRR13, KRR14, BKK+18, BHK17, BK20, KRR13]. Restricting verification to a *designated* verifier implies that the worker needs to produce a fresh proof unique for each particular verifier for any computation, which is certainly not ideal. Another line of work [GGP10] achieves public verification but does not achieve public delegation. In other words, the input provider needs to run a pre-processing algorithm corresponding to the program P before being able to delegate. Another model which has been extensively explored is when the User/Verifier is allowed to have interaction with the Worker, i.e., *interactive delegation*. Influenced by the first work on interactive efficient arguments by Kilian [Kil92], there have been several works from standard assumptions [KRR13, PHGR13, PRV12, BKP18] and some even unconditional soundness[GKR15, RRR19]. These are however not applicable in our setting where only one-way communication is permitted between the parties, as can be seen in the acyclic graph in Figure 1.

With regard to *non-interactive publicly verifiable delegation*, Starting from the seminal work on computationally sound proofs by Micali [Mic00] in the random oracle model, there have been several constructions on publicly verifiable non-interactive delegation schemes [BCC+17, BCCT13, DFH12, BCI+13, GGPR13, Gro10, Lip12, PR17] based on the *Random Oracle Model* or non-standard *knowledge assumptions*. From more standard assumptions, there have been several works recently [BKK+18, BHK17, BK20, KPY19]. An illustrative example is the recent work of [KPY19] that proposed the first publicly verifiable non-interactive delegation scheme from a falsifiable decisional assumption on groups with bilinear pairings. However, in contrast with the setting we describe above, they can only achieve succinct delegation when the Verifier *knows the program* P. In our setting of Boolean circuits, this trivializes the delegation problem, since reading P's description takes as long as evaluating P. Indeed, the case that we consider — where Alice's program is

large — is extremely well motivated: the program P could be an ML model with billions of painstakingly learned parameters.

The SNARGs for $\mathcal{NP}$ barrier. Why has constructing a protocol that caters to the fully non-interactive setting which we have defined been so elusive? Note that in our problem, the User/Verifier and Input Provider do not know the program P. Hence, from User/Verifier's perspective, P is an $\mathcal{NP}$ witness. Thus, it certainly seems that finding a solution is intricately related to a major goal in the area of non interactive succinct proof systems, i.e., *SNARGs for $\mathcal{NP}$*. Unfortunately, the only known constructions of SNARGs for $\mathcal{NP}$ base their soundness on the *Random Oracle Model* or non-standard *knowledge assumptions*. Finding a solution solely relying on standard assumptions has been an open problem for over a decade. In fact, the closest that we have come is the very recent work achieving SNARGs for $\mathcal{P}$ [CJJ21b] (see also [KVZ21]).

The major technical contribution in our work is to enable *Non-Interactive Publicly Verifiable Succinct Delegation for Committed Programs* without having to use SNARGs for $\mathcal{NP}$.

Our Contribution: We present the first complete solution to achieving succinct non interactive publicly verifiable delegation for committed programs. Indeed, furthermore, we can also achieve zero-knowledge guarantees as well. Our only computational assumption is the hardness of the *Learning with Errors* (LWE) problem. Somewhat surprisingly, we show that SNARGs for $\mathcal{NP}$ are not required to solve this problem, even though the statement being proved looks like an $\mathcal{NP}$ statement to the Verifier!

Instead, we show that many ideas from SNARGs for $\mathcal{P}$ [CJJ21b] can in fact be applied here. Although P is unknown to the User/Verifier, we show that it suffices for Alice to communicate a tiny amount of information of size $\mathsf{poly}(\log|P|)$ about the program P (referred to as H_P) as shown in Figure 1. Because Alice is the author of P, this H_P can be *trusted* as correctly generated. We stress that Alice does not need to know x to compute H_P, hence this achieves public delegation and public verification in the completely non-interactive model described above. This leads to our main theorem,

Theorem 1.2. *Assuming the hardness of the LWE problem, Figure 2 gives a construction for publicly verifiable non-interactive succinct delegation for committed programs with CRS size, proof size and verifier time* $\mathsf{poly}(\lambda, \log|P|, |x|)$ *and prover run time being* $\mathsf{poly}(\lambda, |P|)$.

Finally, in order to get zero-knowledge, it suffices for Alice to commit to H_P rather than sending it out in the open. We then present a generic transformation to convert any delegation protocol of this form to attain zero-knowledge.

Theorem 1.3. *Assuming the hardness of the LWE problem and existence of a succinct delegation scheme, Figure 5 gives a construction for publicly verifiable succinct delegation scheme with zero knowledge such that CRS size, proof size and verifier time are* $\mathsf{poly}(\lambda, \log|P|)$ *and prover run time is* $\mathsf{poly}(\lambda, |P|)$.

Finally, we also show how to achieve *zero knowledge* versions of our delegation scheme, meeting the same strong succinctness and efficiency goals, and under the same assumption (LWE).

We present a more detailed explanation in the Technical Overview.

2 Technical Overview

Our Delegation Scenario Let us briefly recall the setup of our delegation scenario. There are 4 parties, namely, (1) Alice-the program author $\mathsf{ProgAuth}$ who sends a program P and some computed state state to a Worker, (2) an Input Provider I that outputs some value x, (3) Worker W that takes as input (P, state, x) and outputs $P(x)$ and a proof Π, and (4) User/Verifier V gets as inputs $(x, P(x), \Pi)$ and outputs 1 if and only if Π was a valid proof. Assume that all the parties get the security parameter λ as an input. An additional requirement is that $|\Pi|$ and runtime of V is $\mathsf{poly}(\lambda, \log|P|, |x|)$, and W runs in time $\mathsf{poly}(\lambda, |x|, |P|)$. Thus, any non-interactive publicly verifiable succinct delegation scheme can be viewed as a collection of 4 algorithms: $\mathsf{sDel} = (\mathsf{ProgAuth}, W, I, V)$ with the input output behaviour and efficiency guarantees as specified. Note that this is indeed a $\mathcal{P}$ computation for the Worker but the primary challenge is that the verifier does not have knowledge of the "witness" P, hence this is an $\mathcal{NP}$ computation from the verifier's point of view. In this work, we observe that it is indeed feasible to achieve our delegation scenario for all circuits without having

to go through SNARGs for $\mathcal{NP}$. Our technique is based on the recent work of Choudhuri et. al. [CJJ21b] on SNARGs for $\mathcal{P}$. We begin by giving a brief overview of their approach and elaborate the challenges of directly incorporating their methodology for our setting.

Challenges of implementing [CJJ21b] Roughly, the work of [CJJ21b] uses Batch Arguments for $\mathcal{NP}$ (BARGs), which they build from LWE. BARGs allow an efficient prover to compute a non-interactive and publicly verifiable "batch proof" of many $\mathcal{NP}$ instances, with size $\mathsf{poly}(|w| \log T)$ for T-many $\mathcal{NP}$ statements with each witness of size $|w|$. They begin by looking at P as a Turing machine and the steps of P's computation are interpreted as an *Index Circuit* C_{index}. Say, P terminates in T steps. Formally, they construct a BARG for the *Index Language* $\mathcal{L}^{\mathsf{index}}$, where

$$\mathcal{L}^{\mathsf{index}} = \{(C_{\mathsf{index}}, i) | \exists w_i, \text{ such that } C(i, w_i) = 1\},$$

where $i \in [T]$ is an index. Let $s_0, s_1, \ldots, s_T$ denote the encoding of internal states of P along with its tape information, and let *Step* be its step function such that $Step(s_{i-1}) = s_i$ The witness for the i^{th} intermediate computation is then defined as $w_i = (s_{i-1}, s_i)$. The index circuit is built such that $(C_{\mathsf{index}}, i) \in \mathcal{L}^{\mathsf{index}}$ essentially implies that the Turing machine step function was correctly computed on s_{i-1} to yield s_i. Note that this alone does not suffice as a proof because the BARG only confirms that (s_{i-1}, s_i) and (s'_i, s_{i+1}) are valid witnesses. If $s_{i-1}, s_i, s'_i, s_{i+1}$ are generated by the step function of the same Turing machine P, they they must be consistent with each other, i.e., $s_i = s'_i$. However, this is not guaranteed by a BARG.

To resolve this issue, the prover also sends a *Somewhere Extractable Hash (SE)* to the witnesses $(s_0, \{s_{i-1}, s_i\}_{i \in [T]})$. The extraction property of this hash allows the verifier to check if the witness of two consecutive BARG instances are indeed consistent with each other. At this stage, we would like to remind the reader of their efficiency goals where crucially, they desire proof size and verification time to be $\mathsf{poly}(\lambda, \log T)$. However, note that $|C_{\mathsf{index}}|$ grows linearly with $|s_i|$ and the known constructions [HW15] of SE hashes can only produce hashes with size $\mathsf{poly}(|s_i|)$. This means that total communication and verifier run time will be at least $\mathsf{poly}(|s_i|)$. This is certainly no good if the Turing machine has massive states. To overcome this final barrier, they make use of Hash Trees which compress the states s_i to a short hash h_i such that $|h_i| = \mathsf{poly}(\lambda)$. Such trees [Mer88] also have a soundness property where a Prover must produce a succinct proof Π_i that the hash tree was indeed implemented correctly at the i^{th} step of the Turing machine computation. Once the succinctness guarantee is ensured, the prover then produces SE hashes corresponding to $(h_0, \Pi_0, \{h_{i-1}, \Pi_{i-1}, h_i, \Pi_i\}_{i \in [T]})$ along with the openings to these hashes. To summarise, the proof consists of two parts, (1) The BARG proof, and (2) A *somewhere extractable* hash of the witnesses. Relying on the soundness of BARG, extraction correctness property of SE hash and soundness of the Hash Tree, a User/Verifier can check if each of these T intermediate steps are indeed the correct states for P, i.e., the computation was done honestly.

However, this approach only works if User/Verifier can confirm that the inputs used for the computation by the Worker, i.e. (P, x) are indeed the correct starting values as provided by the Program Author and Input Provider. This works fine for [CJJ21b] because in their setting, the User/Verifier actually knows (P, x). Unfortunately, this is not at all true in our scenario. Thus, the techniques of Choudhuri et al. [CJJ21b] cannot be implemented directly as the soundness of the BARG proof cannot provide any guarantees if there is no way for to check that the initial inputs used by the Worker are correct.

Our Idea. We start with an alternate way of interpreting the computation of P on input x as the following: Consider a Circuit-Universal Turing Machine $\mathcal{TM}$ which takes as input P, x, y and accepts (P, x, y) in $T = \tilde{O}(|P|)$ steps if $P(x) = y$. We can assume without loss of generality that $P \in \{0,1\}^m$, $x \in \{0,1\}^n$ and $y \in \{0,1\}$, where $m, n \leq 2^\lambda$. Keeping this in mind, we introduce the notion of *Semi-Trusted SNARGs* for $\mathcal{NP}$. This new kind of SNARG is one that will work for general $\mathcal{NP}$ computations, but only with a little bit of extra help from a trusted party that knows the witness – which in our delegation scenario is Alice, who knows the witness P!

A Semi-Trusted SNARG is a tuple of algorithms: $\mathsf{stSNARG} = (\mathsf{Setup}, \mathsf{TrustHash}, \mathsf{P}, \mathsf{V})$, where (1) Setup is a randomised algorithm that takes as input the security parameter and outputs a Common Random String (CRS). (2) a *trusted* deterministic $\mathsf{TrustHash}$ takes as input the (CRS, P) and outputs a digest H_P, (3) a deterministic prover P which takes as input CRS and (P, x, y), and outputs a proof Π, and (4) a deterministic

verifier V which gets CRS,(H_P, x, y, Π) as input and outputs 1 iff Π is valid. It must be that $|\Pi|$ and run time of V is $\mathsf{poly}(\lambda, \log T)$, and P runs in time $\mathsf{poly}(\lambda, |x|, |P|, T)$. A simple reduction shows that in the CRS model (or alternatively in a model where Alice chooses the CRS), existence of $\mathsf{stSNARG}$ implies the existence of sDel. We show this formally in Lemma 5.19. Hence, from here onwards, our goal is to construct a *Semi-Trusted SNARG for* $\mathcal{NP}$.

We briefly provide an informal explanation of our construction.

Like [CJJ21b], every intermediate state of the Universal Turing Machine is encoded into a succinct hash (call it $\mathsf{h}_0, \ldots, \mathsf{h}_T$) accompanied with succinct proofs $\{\Pi_i\}_{i\in[T]}$. The prover computes two independent copies of *Somewhere Extractable* (SE) hashes (c_1, c_2) of the encoding $\{\mathsf{h}_0, \{(\mathsf{h}_1, \Pi_1), \ldots, (\mathsf{h}_T, \Pi_T)\}\}$ along with their corresponding openings. Here $\mathsf{h}_0 = (\mathsf{st}_0, H_P, H_x, H_{work})$, where st_0 is that hash of $\mathcal{TM}$'s starting state which is publicly known, H_x denote the hash of x, and H_{work} is the hash of $\mathcal{TM}$'s blank work tape. The use of two independent SE hashes are pivotal for soundness which we elaborate later.

We point out that $\mathsf{TrustHash}$ computes H_P using the same hash tree which is used for hashing the Turing machine states by the Prover. This is crucial to ensure soundness of the protocol. We show in Figure 3 that once the public hash is fixed by $\mathsf{TrustHash}$, one can hard code $(y, c_1, c_2, T, H_P, H_x)$ to the index circuit C_{index} for BARG. At this point, we can now follow the approach from [CJJ21b]. V can rely upon the binding property/collision resistance of the hash to ensure that the prover has used P and x which were provided by Alice and the input provider respectively. The main observation here is that once a trusted party fixed a hash of the program P and V is convinced that computation was commenced with the correct inputs, the soundness of BARG, extraction correctness of the SE hash and soundness of hash tree ensures that the semi-trusted SNARG construction is sound.

While our proof of soundness closely follows the blueprint of [CJJ21b], we choose to present our proof in a different, and arguably simpler, way. In [CJJ21b], *No-Signaling Somewhere Extractable*(NSSE) hashes are use extensively. In our proof, we choose to omit explicit use of this notion, and instead we make direct use of two independent SE hashes as mentioned above. A simple hybrid argument then gives a straightforward proof for soundness. This shows that the "anchor and step" use of SE hashes, which dates to the introduction of somewhere-binding hashes [HW15] in 2015, is directly sufficient for this proof of soundness.

Zero-Knowledge We have only discussed soundness guarantees thus far. However, in our delegation scenario, it might also be extremely important to ensure that no information about P leaked to V during the delegation process. Hence it is important to add *zero-knowledge* guarantees to our protocol. We finally give a generic transformation to modify a semi-trusted SNARG to add zero knowledge guarantees. In order to do so we make use of a statistically binding extractable commitment scheme and a NIZK, and roughly make the following modifications:

- We add an additional commitment to 0 in the CRS which is never used in the proof but helps in proving zero knowledge.
- The public hash output by $\mathsf{TrustHash}$ is a binding commitment C_P of H_P. It then sends (P, H_P) to the worker W only.
- The SE hashes c_1, c_2 are also committed as a part of the proof and not published in the open.
- The prover wraps the BARG proof Π with a NIZK proof which proves that that the BARG verification circuit indeed accepts the BARG proof.
- The Verifier then checks if the NIZK proof is valid.

The binding and hiding property of the commitment, and *witness indistinguishability* of NIZK guarantees zero knowledge.

3 Preliminaries

We define the underlying primitives borrowed from prior work which are used as building blocks to perform the Succinct Delegation in the setup.

Definition 3.1 (Non-Interactive Zero Knowledge(NIZK) Arguments in the CRS model)**.** *A non interactive zero knowledge argument for a language L in the Common Reference String (CRS) model is defined three PPT algorithms:*

- $\mathsf{Setup}(1^n, 1^\lambda)$ *outputs a uniform random string* crs *given a statement of length n and security parameter λ.*
- *Prover $P(\mathsf{crs}, x, w)$ outputs a proof π given a statement witness pair (x, w) in the NP relation R such that $|\Pi| = \mathsf{poly}(|x|, |w|)$.*
- *Verifier $V(\mathsf{crs}, x, \pi)$ either accepts or rejects.*

The following properties must be satisfied:

- *Completeness: $V(\mathsf{crs}, x, \pi)$ must always accept if $x \in L$ and $\pi \leftarrow P(\mathsf{crs}, x, w)$.*
- *Computational Soundness: for every non-uniform* poly *time prover P^*, there exists a negligible function $\epsilon(\lambda)$ such that for any $n \in \mathbb{N}$ and $x \notin L$,*

$$\Pr[\mathsf{crs} \leftarrow \mathsf{Setup}(1^n, 1^\lambda), \pi^* \leftarrow P(\mathsf{crs}, x), V(\mathsf{crs}, x, \pi^*) \text{ accepts}] \leq \epsilon(\lambda).$$

- *Non Interactive Zero Knowledge: There exists a PPT simulator* M *such that for every $x \in L$ such that the distribution of the transcript output by* Setup *and P, i.e., $(\mathsf{crs}, P(\mathsf{crs}, x, w)) : \mathsf{crs} \leftarrow \mathsf{Setup}(1^n, 1^\lambda)$ is statistically indistinguishable from the output of $\mathsf{M}(x)$. Note that $M$is allowed to generate its own CRS.*

There has been a string of recent works, [PS19, CCH$^+$19, HLR21] which show how to instantiate such NIZKs from LWE.

Definition 3.2 (Statistically Binding Extractable Commitment Scheme)**.** *A Statistically binding commitment scheme* $\mathsf{Com}_{\mathsf{bind}}$ *in the CRS model is a tuple of efficiently polynomial time algorithms (*$\mathsf{Gen}, \mathsf{TGen}, \mathsf{C}, \mathsf{Ext}$*), where,*

- $\mathsf{Gen}(1^\lambda, 1^N)$ *which on input the security parameter λ and message length N outputs a common reference string* crs*.*
- $\mathsf{TGen}(1^\lambda, 1^N)$ *outputs a common reference string* crs *and trapdoor* td*.*
- $\mathsf{C}(\mathsf{crs}, m; r)$ *takes as input* crs*, a message m to be committed, and uses randomness r to output a commitment* com *such that $|c| = \mathsf{poly}(|m|)$.*
- $\mathsf{Ext}(\mathsf{com}, \mathsf{td})$ *is a deterministic algorithm which takes as input a commitment* com *and trapdoor* td*, and outputs a message m.*

They have the following properties:

- ***CRS indistinguishability:*** *The distribution of* crs *generated by* Gen *and* TGen *must be be indistinguishable.*
- ***Statistical Binding:*** *With high probability over the choice of $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda)$, there does not exists r_0, r_1, and messages $m_0 \neq m_1$ such that $\mathsf{C}(\mathsf{crs}, m_0; r_0) = \mathsf{C}(\mathsf{crs}, m_1; r_1)$.*
- ***Computational Hiding:*** *For messages $m_0 \neq m_1$, and randomness r_0, r_1 the distribution of $(\mathsf{crs}, \mathsf{com}_0)$ is computationally indistinguishable from $(\mathsf{crs}, \mathsf{com}_1)$. Here, $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda), \mathsf{com}_0 \leftarrow \mathsf{C}(\mathsf{crs}, m_0; r_0)$, and $\mathsf{com}_1 \leftarrow \mathsf{C}(\mathsf{crs}, m_1; r_1)$.*
- ***Extraction Correctness:*** *For any security parameter $\lambda \in \mathbb{N}$, message m, randomness r*

$$\Pr[\big((\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{TGen}(1^\lambda), \mathsf{com} \leftarrow \mathsf{C}(m, \mathsf{crs}; r)\big) \implies \mathsf{Ext}(\mathsf{com}, \mathsf{td}) = m] = 1.$$

Note that Ext *is a deterministic algorithm, hence this property itself implies statistical binding of* $\mathsf{Com}_{\mathsf{bind}}$*.*

Given a commitment com *and* crs*, a valid corresponding pair (m, r) is known as the opening for* com*.*

Any public key encryption scheme from LWE [LS19] can be used to construct a Statistically Binding Extractable Commitment Scheme.

3.1 Somewhere Extractable Hash

Somewhere Statistically Binding (SSB) Hashes were introduced in [HW15] and has been extensively used in prior works[CJJ21b, CJJ21a]. An SSB hash works in two modes, namely, (1) normal mode where the key is generated uniformly at random and (2) the trapdoor mode where the key is generated according to a subset S denoting some bits of the message to be hashed. An extension of SSB hashes are somewhere extractable (SE) hashes introduced in [CJJ21b]. Formally, a somewhere extractable (SE) hash is a tuple of algorithms $(\mathsf{Gen}, \mathsf{TGen}, \mathsf{Hash}, \mathsf{Open}, \mathsf{Verify}, \mathsf{Ext})$ described below:

- $\mathsf{SE.Gen}(1^\lambda, 1^N, 1^{|S|})$: On input the security parameter, message length N, and the size of a subset $S \subseteq [N]$, the "normal mode" key generation outputs a uniformly random key K.
- $\mathsf{SE.TGen}(1^\lambda, 1^N, S)$: On input the security parameter, message length and a subset $S \subseteq [N]$, the "trapdoor mode" key generation algorithm outputs a hash key K^* and a trapdoor td.
- $\mathsf{SE.Hash}(K, \mathbf{m} \in \{0,1\}^N)$: On input the hash key K, and vector $\mathbf{m} = (m_1, \ldots, m_N)$, outputs a hash h.
- $\mathsf{SE.Open}(K, \mathbf{m}, i)$: On input the hash key K, vector $\mathbf{m} = (m_1, \ldots, m_N)$, and an index $i \in [N]$, it outputs an opening π_i to m_i.
- $\mathsf{SE.Verify}(K, \mathsf{h}, m_i, i, \pi_i)$: On input the hash key K, a hash h, a bit $m_i \in \{0,1\}$, and a local opening π_i, the verification algorithm either accepts (output 1) or rejects (output 0) the local opening.
- $\mathsf{SE.Ext}(\mathsf{h}, \mathsf{td})$: On input a hash h and trapdoor td generated by TGen with respect to the subset S, the deterministic extraction algorithm outputs an extraction string m^*_S on the subset S.

Furthermore, we need the SE Hash to have the following properties:

- **Succinct Key.** The size of the key is bounded by $\mathsf{poly}(\lambda, |S|, \log N)$.
- **Succinct Hash.** The hash size is bounded by $\mathsf{poly}(\lambda, |S|, \log N)$.
- **Succinct Local Opening.** The size of the local opening $\pi_i \leftarrow \mathsf{Open}(K, m, i)$ is bounded by $\mathsf{poly}(\lambda, |S|, \log N)$.
- **Succinct Verification.** The running time of the verification algorithm is bounded by $\mathsf{poly}(\lambda, |S|, \log N)$.
- **Key Indistinguishability.** For any non-uniform PPT adversary $\mathcal{A} := (\mathcal{A}_1, \mathcal{A}_2)$ and any polynomial $N = N(\lambda)$, there exists a negligible function $\nu(\lambda)$ such that

$$\big|\Pr[\mathcal{A}_2(K) = 1 | S \leftarrow \mathcal{A}_1(1^\lambda, 1^N), K \leftarrow \mathsf{Gen}(1^\lambda, 1^N, 1^{|S|})] - \Pr[\mathcal{A}_2(K^*) = 1 | S \leftarrow \mathcal{A}_1(1^\lambda, 1^N), (K^*, \mathsf{td}) \leftarrow \mathsf{TGen}(1^\lambda, 1^N, S)]\big| \leq \nu(\lambda).$$

- **Opening Completeness.** For any hash key K, any message $\mathbf{m} = (m_1, \ldots, m_N)$, and index i, we have

$$\Pr[\mathsf{Verify}(K, \mathsf{h}, m_i, i, \pi_i) = 1 | \mathsf{h} \leftarrow \mathsf{Hash}(K, \mathbf{m}), \pi_i \leftarrow \mathsf{Open}(K, \mathbf{m}, i)] = 1.$$

- **Extraction Correctness.** For any subset $S \subseteq [N]$, any trapdoor key $(K^*, \mathsf{td}) \leftarrow \mathsf{TGen}(1^\lambda, 1^N, S)$, any hash h, index $i \in S$, bit m_{i^*} and proof π_{i^*}, we have,

$$\Pr[\mathsf{Verify}(K^*, \mathsf{h}, m_{i^*}, i^*, \pi_{i^*}) = 1 \implies \mathsf{Ext}(\mathsf{h}, \mathsf{td})\big|_{i^*} = m_{i^*}] = 1.$$

  We point out that this is indeed the somewhere statistical binding property of the hash as $\mathsf{Ext}(\mathsf{h}, \mathsf{td})$ is a deterministic function, hence the uniqueness of the extraction ensures binding of the hash.

[CJJ21b]+[HW15] show how to construct an SE hash from the *LWE* assumption.

3.2 Non Interactive Batch Arguments (BARG) for Index Language

Definition 3.3 (Circuit Satisfiability Language)**.** *We define the language* $\mathsf{SAT} = \{(C, x) | \exists w \text{ such that } C(x, w) = 1\}$*, where* $C : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}$ *is a boolean circuit, and* $x \in \{0,1\}^n$ *is an instance.*

A non interactive BARG for SAT involves a prover and verifier having as common input a circuit C, and a series of T instances $x_1, \ldots, x_T$. The prover then sends a single message to the verifier with a proof that $(C, x_1), \ldots, (C, x_T) \in \mathsf{SAT}$. In particular, a non interactive BARG has a tuple of four algorithms $(\mathsf{Gen}, \mathsf{TGen}, \mathsf{Prove}, \mathsf{Verify})$ that are defined as follows:

- $\mathsf{Gen}(1^\lambda, 1^T, 1^{|C|})$: On input security parameter λ, number of instances and size of circuit, the CRS generation algorithm outputs a uniformly sampled crs.
- $\mathsf{TGen}(1^\lambda, 1^T, 1^{|C|}, i^*)$: On input the security parameter, number of instances, size of circuit and an index i^*, the trapdoor CRS generation algorithm outputs crs^*.
- $\mathsf{Prove}(\mathsf{crs}, C, x_1, \ldots, x_T, w_1, \ldots, w_T)$: On input crs, circuit C, list of T instance and their corresponding witnesses, the prover outputs a proof π.
- $\mathsf{Verify}(\mathsf{crs}, C, x_1, \ldots, x_T, \pi)$: On input crs, circuit C, list of T instances and proof π, the verifier decides to accept (output 1) or reject(output 0) the proof.

BARGs satisfy the following properties:

- **Succinct Communication.** Size of π is boundede by $\mathsf{poly}(\lambda, \log T, |C|)$.
- **Compact CRS.** The crs size is bounded by $\mathsf{poly}(\lambda, \log T, |C|)$.
- **Succinct Verification.** Verification algorithm runs in time $\mathsf{poly}(\lambda, \log T, |C|) + \mathsf{poly}(\lambda, \log T + n)$.
- **CRS indistinguishability.** For any non-uniform PPT adversary $\mathcal{A} \coloneqq (\mathcal{A}_1, \mathcal{A}_2)$ and any polynomial $T = T(\lambda)$, there exists a negligible function $\nu(\lambda)$ such that

$$\Big| \Pr[\mathcal{A}_2(\mathsf{crs}) = 1 | i^* \leftarrow \mathcal{A}_1(1^\lambda, 1^T), \mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda, 1^T)] - \Pr[\mathcal{A}_2(\mathsf{crs}^*) = 1 | i^* \leftarrow \mathcal{A}_1(1^\lambda, 1^T), \mathsf{crs}^* \leftarrow \mathsf{TGen}(1^\lambda, 1^T, i^*)] \Big| \leq \nu(\lambda).$$

  Corollary 3.4. *As a direct consequence of CRS indistinguishability, we have that for any non-uniform PPT adversary* $\mathcal{A} \coloneqq (\mathcal{A}_1, \mathcal{A}_2)$ *and any polynomial* $T = T(\lambda)$*, and* $i \neq j$*, there exists a negligible function* $\nu(\lambda)$ *such that*

$$\Big| \Pr[\mathcal{A}_2(\mathsf{crs}^i) = 1 | i \leftarrow \mathcal{A}_1(1^\lambda, 1^T), \mathsf{crs}^i \leftarrow \mathsf{TGen}(1^\lambda, 1^T, i)] - \Pr[\mathcal{A}_2(\mathsf{crs}^j) = 1 | j \leftarrow \mathcal{A}_1(1^\lambda, 1^T), \mathsf{crs}^j \leftarrow \mathsf{TGen}(1^\lambda, 1^T, j)] \Big| \leq \nu(\lambda).$$

- **Completeness.** For any circuit C, T instances $x_1, \ldots, x_T$ such that $(C, x_1), \ldots, (C, x_T) \in \mathsf{SAT}$ and witnesses $w_1, \ldots, w_T$ corresponding to respective instance, we have,

$$\Pr[\mathsf{Verify}(\mathsf{crs}, C, x_1, \ldots, x_T, \pi) = 1 | \mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda, 1^T, 1^{|C|}), \pi \leftarrow \mathsf{Prove}(\mathsf{crs}, C, x_1, \ldots, x_T, w_1, \ldots, w_T)] = 1.$$

- **Semi-Adaptive Somewhere Soundness.**For any non-uniform PPT adversary $\mathcal{A}$ and any polynomial $T = T(\lambda)$, there exists a negligible function $\nu(\lambda)$ such that

$$\Pr[i^* \in [T] \wedge (C, x_{i^*}) \notin \mathsf{SAT} \wedge \mathsf{Verify}(\mathsf{crs}^*, C, x_1, \ldots, x_T, \pi) = 1 | i^* \leftarrow \mathcal{A}(1^\lambda, 1^T), \mathsf{crs}^* \leftarrow \mathsf{TGen}(1^\lambda, 1^T, i^*), (C, x_1, \ldots, x_T, \pi) \leftarrow \mathcal{A}(\mathsf{crs}^*)] \leq \nu(\lambda).$$

- **Somewhere Argument of Knowledge.** There exists a PPT extractor E sucht that, for any non uniform PPT adversary $\mathcal{A}$, and any polynomial T, there exists a negligible function $\nu(\lambda)$ such that

$$\Big|\Pr[C(x_{i^*},w)=1 | i^* \leftarrow \mathcal{A}(1^\lambda,1^T), \mathsf{crs}^* \leftarrow \mathbb{E}(1^\lambda,1^T,i^*), (C,x_1,\ldots,x_T,\pi) \leftarrow \mathcal{A}(\mathsf{crs}^*), w \leftarrow E(C,x_1,\ldots,x_T,\pi)] - \Pr[\mathsf{Verify}(\mathsf{crs},C,x_1,\ldots,x_T,\pi)=1 | i^* \leftarrow \mathcal{A}(1^\lambda,1^T), \mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda,1^T), (C,x_1,\ldots,x_T,\pi) \leftarrow \mathcal{A}(\mathsf{crs})]\Big| \leq \nu(\lambda).$$

  In addition to this, crs^* must be computationally indistinguishable from crs.

Definition 3.5 (Index Language)**.** *We define an Index Language as the following:*

$$\mathcal{L}^{\mathsf{index}} = \{(C,i) | \exists w, \text{ such that } C(i,w)=1\},$$

where C is a boolean function and i is an index.

Note that non interactive batch arguments for index language is a special case of non interactive BARGs for circuit satisfiability when the instances $(x_1,\ldots,x_T)$ are indices $(1,\ldots,T)$. In this case, one removes the instances from input to the prover and verifier algorithm. Since, the verifier does not read the instances, it reduces the succinct verification time to $\mathsf{poly}(\lambda,\log T,|C|)$.

[CJJ21b] show the existence of non interactive BARGs for index language with succinct verification property from the *LWE* assumption.

3.3 Hash Tree

To enable Turing Machine Delegation when the state space is unbounded, we use the notion of Hash Tree as defined in [KPY19]. Formally, a hash tree consists of a tuple of six algorithms:

- $\mathsf{HT.Gen}(1^\lambda)$: On input security parameter λ, outputs a hash key dk.
- $\mathsf{HT.Hash}(\mathsf{dk},D)$: On input the hash key and a string $D \in \{0,1\}^L$, outputs the hash tree tree and its root rt.
- $\mathsf{HT.Read}(\mathsf{tree},l)$: On input the hash tree and a memory location $l \in [2^\lambda]$, it outputs a bit b that is read from l^{th} location of the string corresponding to the tree and a proof π.
- $\mathsf{HT.Write}(\mathsf{tree},l,b)$:On input the hash tree, a memory location $l \in [L+1]$ and bit b, it outputs a new tree tree', root rt' along with a proof π'.
- $\mathsf{HT.VerRead}(\mathsf{dk},\mathsf{rt},l,b,\pi)$: On input the hash key dk, hash tree root rt,memory location l, bit b and proof π, outputs 0 or 1 to reject or accept the proof that b is indeed the correct bit read at location l.
- $\mathsf{HT.VerWrite}(\mathsf{dk},\mathsf{rt},l,b,\mathsf{rt}',\pi')$: On input hash key dk, tree root rt', memory location l, bit b, new root rt' an proof π, either accepts (output 1) or rejects (output 0) the proof.

A hash tree scheme must satisfy the following properties:

- **Completeness of Read.** For every $\lambda \in \mathbb{N}, D \in \{0,1\}^L$ such that $L < 2^\lambda$ and $l \in [L]$,

$$\Pr[\mathsf{HT.VerRead}(\mathsf{dk},\mathsf{rt},l,b,\pi)=1 \wedge D[l]=b \,\big|\, \mathsf{dk} \leftarrow \mathsf{HT.Gen}(1^\lambda), (\mathsf{tree},\mathsf{rt}) \coloneqq \mathsf{HT.Hash}(\mathsf{dk},D), (b,\pi) \coloneqq \mathsf{HT.Read}(\mathsf{tree},l)] = 1.$$

- **Completeness of Write.** For every $\lambda \in \mathbb{N}, D \in \{0,1\}^L$ such that $L < 2^\lambda$, $l \in [L+1]$, $b \in \{0,1\}$. Further if $l \leq L$. then let D' be the string D with its l^{th} location set to b, otherwise let D' be the string D appended with b at the end, i.e., $D' = D \| b$. Then,

$$\Pr[\mathsf{HT.VerWrite}(\mathsf{dk},\mathsf{rt},l,b,\mathsf{rt}',\pi)=1 \wedge (\mathsf{tree}',\mathsf{rt}') = \mathsf{HT.Hash}(\mathsf{dk},D') \,\big|\, \mathsf{dk} \leftarrow \mathsf{HT.Gen}(1^\lambda), (\mathsf{tree},\mathsf{rt}) \coloneqq \mathsf{HT.Hash}(\mathsf{dk},D), (\mathsf{tree}',\mathsf{rt}',\pi) \coloneqq \mathsf{HT.Write}(\mathsf{tree},l,b)] = 1.$$

- **Efficiency.** The running time of $\mathsf{HT.Hash}$ is $|D|\cdot\mathsf{poly}(\lambda)$. The length of the root rt and proofs produced by $\mathsf{HT.Read}$ and $\mathsf{HT.Write}$ are $\mathsf{poly}(\lambda)$.
- **Soundness of Read.** For every polynomial size adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that for every λ, we have

$$\Pr[b_1 \neq b_2, \mathsf{HT.VerRead}(\mathsf{dk}, \mathsf{rt}, l, b_1, \pi_1) = 1, \mathsf{HT.VerRead}(\mathsf{dk}, \mathsf{rt}, l, b_2, \pi_2) = 1 \big| \mathsf{dk} \leftarrow \mathsf{HT.Gen}(1^\lambda), (\mathsf{rt}, l, b_1, \pi_1, b_2, \pi_2) \leftarrow \mathcal{A}(\mathsf{dk})] \leq \mathsf{negl}(\lambda).$$

- **Soundness of Write.** For every polynomial size adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that for every λ, we have

$$\Pr[\mathsf{rt}_1 \neq \mathsf{rt}_2, \mathsf{HT.VerWrite}(\mathsf{dk}, \mathsf{rt}, l, b, \mathsf{rt}_1, \pi_1) = 1, \mathsf{HT.VerWrite}(\mathsf{dk}, \mathsf{rt}, l, b, \mathsf{rt}_2, \pi_2) = 1 \big| \mathsf{dk} \leftarrow \mathsf{HT.Gen}(1^\lambda), (\mathsf{rt}, l, b, \mathsf{rt}_1, \pi_1, \mathsf{rt}_2, \pi_2) \leftarrow \mathcal{A}(\mathsf{dk})] \leq \mathsf{negl}(\lambda).$$

Theorem 3.6 (Existence of hash trees [Mer88])**.** *A hash tree scheme as defined above can be efficiently constructed from any collision resistant hash function.*

4 Publicly Verifiable Non Interactive Succinct Delegation

We formally define the notion of Publicly Verifiable Non Interactive Succinct Delegation (sDel) which is similar to the definition proposed in prior works [KPY18]. Such a delegation scheme in the CRS model involves the following PPT algorithms, (1)Software/Program Author $\mathsf{ProgAuth}$ (3)Cloud Worker W , and (3) Verifier VAn sDel comprises of the following polynomial time algorithms:

- $\mathsf{sDel.Setup}(1^\lambda)$: A randomized setup algorithm which on input security parameter λ and outputs crs.
- $\mathsf{sDel.ProgAuth}(1^\lambda, \mathsf{crs})$: A program author which takes as input λ, outputs a (not public) program $P \in \{0,1\}^m$, $m \leq 2^\lambda \in \mathbb{N}$, state and a public digest H_P.
- $\mathsf{sDel}.W(\mathsf{crs}, P, \mathsf{state}, H_P, x)$: A deterministic cloud worker which on input crs, program P, input $x \in \{0,1\}^n, n \leq 2^\lambda \in \mathbb{N}$ outputs a value y and proof Π.
- $\mathsf{sDel}.V(\mathsf{crs}, x, y, H_P, \Pi)$: A deterministic verifier which on input crs, digest H_P, x, y, Π either accepts or rejects.

A publicly verifiable succinct delegation scheme ($\mathsf{sDel.Setup}, \mathsf{sDel.ProgAuth}, \mathsf{sDel}.W, \mathsf{sDel}.V$) satisfies the following properties:

- **Completeness.** For every PPT program generating algorithm $\mathsf{sDel.ProgAuth}$, every $\lambda, n, m \in \mathbb{N}$, and for all $x \in \{0,1\}^n$ such that $n, m < 2^\lambda$, we have

$$\Pr[\mathsf{sDel}.V(\mathsf{crs}, x, y, H_P, \Pi) = 1 \wedge P(x) = y \big| \mathsf{crs} \leftarrow \mathsf{sDel.Setup}(1^\lambda), ((P, \mathsf{state}), H_P) \leftarrow \mathsf{sDel.ProgAuth}(1^\lambda, \mathsf{crs}), (y, \Pi) \leftarrow \mathsf{sDel}.W(\mathsf{crs}, P, \mathsf{state}, H_P, x)] = 1.$$

- **Efficiency.** $\mathsf{sDel.Setup}$ runs in time $\mathsf{poly}(\lambda)$, $\mathsf{sDel}.W$ runs in time $\mathsf{poly}(\lambda, |P|, |x|)$ and outputs a proofs of length $\mathsf{poly}(\lambda, \log|P|, |x|)$, and $\mathsf{sDel}.V$ runs in time $\mathsf{poly}(\lambda, \log|P|, |x|)$.
- **Soundness.** For every PPT adversary $\mathcal{A} := (\mathcal{A}_1, \mathcal{A}_2)$, every PPT program generating algorithm $\mathsf{sDel.ProgAuth}$, and the tuple $n = n(\lambda), m = m(\lambda)$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that for every $\lambda \in \mathbb{N}$,

$$\Pr[\mathsf{sDel}.V(\mathsf{crs}, x, y, H_P, \Pi) = 1 \wedge P(x) \neq y \big|, \mathsf{crs} \leftarrow \mathsf{sDel.Setup}(1^\lambda), ((P, \mathsf{state}), H_P) \leftarrow \mathsf{sDel.ProgAuth}(1^\lambda, \mathsf{crs}), (x, \mathsf{aux}) \leftarrow \mathcal{A}_1(1^\lambda, \mathsf{crs}), (y, \Pi) \leftarrow \mathcal{A}_2(\mathsf{crs}, P, \mathsf{state}, H_P, x, \mathsf{aux})] \leq \mathsf{negl}(\lambda).$$

To construct sDel, we introduce a notion of *Semi-Trusted Succinct Non-Interactive Arguments* stSNARG which we formally introduce and construct in Section 5. After that, we prove the following lemma (cf. Lemma 5.19) which shows how to construct sDel using stSNARG as a building block.

Lemma 4.1. *Assuming $T = \mathsf{poly}(m, n)$, $T, m, n \leq 2^\lambda$, the* stSNARG *protocol in Figure 2 implies the unconditional existence of a publicly verifiable non interactive succinct delegation scheme* sDel *as defined above.*

4.1 sDel with Zero-Knowledge

A publicly verifiable non interactive succinct delegation scheme with zero knowledge $\mathsf{zk-sDel}$ is defined by the following efficient algorithms:

- $\mathsf{zk-sDel.Setup}(1^\lambda)$: A randomized setup algorithm which on input security parameter λ and outputs crs.
- $\mathsf{zk-sDel.ProgAuth}(1^\lambda, \mathsf{crs})$: A program author which takes as input λ, generates a program $P \in \{0,1\}^m$, $m \leq 2^\lambda \in \mathbb{N}$. Additionally, it computes a digest H_P and creates a statistically binding and extractable commitment C_P of H_P under randomness r. Finally it sends a private output (P, state) and public output C_P. Here state contains the randomness r and H_P encoded in it along with any other state information.
- $\mathsf{zk-sDel.}W(\mathsf{crs}, P, \mathsf{state}, C_P, x)$: A deterministic cloud worker which on input crs, program P, commitment C_P, $x \in \{0,1\}^n, n \leq 2^\lambda \in \mathbb{N}$ outputs a value y and proof Π.
- $\mathsf{zk-sDel.}V(\mathsf{crs}, x, y, C_P, \Pi)$: A deterministic verifier which on input $(\mathsf{crs}, C_P, x, y, \Pi)$ either accepts or rejects.

Apart from the Completeness, Efficiency and Soundness guarantees mentioned above, a publicly verifiable succinct delegation scheme $(\mathsf{zk-sDel.Setup}, \mathsf{zk-sDel.ProgAuth}, \mathsf{zk-sDel.}W, \mathsf{zk-sDel.}V)$ satisfies the following additional property:

Non Interactive Zero Knowledge. For all $\lambda, n, m \in \mathbb{N}$ such that $n, m \leq 2^\lambda$, $\forall, x \in \{0,1\}^n$ and $y \in \{0,1\}$, there exists a PPT simulator $\mathsf{Sim} := (\mathsf{Sim}_1, \mathsf{Sim}_2, \mathsf{Sim}_3)$ such that the distributions of

$$(\mathsf{crs}, x, y, C_P, \Pi) \Big| (\mathsf{crs}, \mathsf{aux}) \leftarrow \mathsf{Sim}_1(1^\lambda), (C_P, \mathsf{aux}') \leftarrow \mathsf{Sim}_2(\mathsf{crs}, \mathsf{aux}), (y, \Pi) \leftarrow \mathsf{Sim}_3(\mathsf{aux}', \mathsf{crs}, x, C_P)$$

and

$$(\mathsf{crs}, x, y, C_P, \Pi) \Big| \mathsf{crs} \leftarrow \mathsf{zk-sDel.Setup}(1^\lambda), ((P, \mathsf{state}), C_P) \leftarrow \mathsf{zk-sDel.ProgAuth}(1^\lambda, \mathsf{crs}), (y := P(x), \Pi) \leftarrow \mathsf{zk-sDel.}W(\mathsf{crs}, P, \mathsf{state}, x, C_P)$$

are indistinguishable.

In Section 6, we present a generic construction of a semi trusted non-interactive succinct arguments with zero-knowledge (ZKstSNARG) from stSNARG. Analogous to the previous lemma, we get the following corollary(cf. Corollary 6.6) from Lemma 5.19

Corollary 4.2. *Assuming $T = \mathsf{poly}(m, n)$, $T, m, n \leq 2^\lambda$, the* ZKstSNARG *protocol in Figure 5 implies the unconditional existence of a publicly verifiable non interactive succinct delegation scheme with zero knowledge.*

5 Semi-Trusted Succinct Non-Interactive Argument (stSNARG)

We introduce a notion of "Semi-Trusted" SNARGs which is similar to the general definition of SNARGs with an addition "trusted" polynomial time algorithm that outputs a hash for the witness. Further, we provide an explicit construction of an stSNARG for all of NP. Note that any SNARG for arbitrary NP language $\mathcal{L}$ can be reformulated as a Turing Machine which takes in as input an instance x along with witness w and accepts x, w in T steps if $x \in \mathcal{L}$ [CJJ21b]. In this work, we modify the definition of [CJJ21b] by using a

Universal Turing Machine $\mathcal{TM}$ which takes as input an instance (x, y), a witness which is a program P and accepts (P, x, y) in T steps if $P(x) = y$. We formalise this notion as follows:

Let $\mathcal{TM}$ be a Universal Turing Machine which takes as input a program $P \in \{0,1\}^m$ for some $m < 2^\lambda$, and $x \in \{0,1\}^n$ for some $n < 2^\lambda$ and $y \in \{0,1\}$ which serve as an input and output for P respectively. $\mathcal{TM}$ accepts (P, x, y) in T steps if $P(x) = y$. A prover produces a proof Π to convince a verifier that $\mathcal{TM}$ accepts P, x, y in T. A publicly verifiable semi-trusted SNARG (stSNARG) for $\mathcal{TM}$ has the following polynomial time algorithms:

- $\mathsf{stSNARG.Setup}(1^\lambda, 1^T)$: A randomized setup algorithm which on input security parameter λ, and number of Turing Machine steps T, outputs crs.
- $\mathsf{stSNARG.TrustHash}(\mathsf{crs}, P)$: A deterministic and honest algorithm which on input crs and a program $P \in \{0,1\}^m$ for some $m < 2^\lambda$, outputs a succinct and public digest H_P of P corresponding to crs.
- $\mathsf{stSNARG.P}(\mathsf{crs}, P, x, y, H_P)$: A deterministic prover algorithm which on input the crs, $P \in \{0,1\}^m$ for some $m < 2^\lambda$, $x \in \{0,1\}^n$ for some $n < 2^\lambda$, $y \in \{0,1\}$ and the digest H_P outputs a proof Π.
- $\mathsf{stSNARG.V}(\mathsf{crs}, x, y, H_P, \Pi)$: A deterministic verification algorithm which on input crs, x, y, digest H_P and proof Π, either accepts(output 1) or rejects(output 0) it.

A Universal Turing Machine $\mathcal{TM}$ on input (P, x, y) outputs 1 if it accepts (P, x, y) within T steps. We define the NP language $\mathcal{L}_{\mathcal{TM}}$ as,

$$\mathcal{L}_{\mathcal{TM}} \coloneqq \{(P, x, y, T, H_P, \mathsf{crs}) \big| \mathcal{TM}(P, x, y) = 1 \wedge \mathsf{stSNARG.TrustHash}(\mathsf{crs}, P) = H_P\}.$$

Note that here P is not considered a part of the witness although it is unknown to the verifier because a typical NP statement puts a there exists constraint on the witness. In that case, the statement becomes trivial because there will always exist a program P which on input x ignores the input and outputs y. We need to ensure that P is the program output by the program author independent of x. Moreover, this is indeed a $\mathcal{P}$ statement for the prover.

A publicly verifiable stSNARG scheme $(\mathsf{stSNARG.Setup}, \mathsf{stSNARG.TrustHash}, \mathsf{stSNARG.P}, \mathsf{stSNARG.V})$ satisfies the following properties:

- **Completeness.** For every $\lambda, T, n, m \in \mathbb{N}$ such that $T, n, m < 2^\lambda$, program $P \in \{0,1\}^m$, input $x \in \{0,1\}^n$ and output $y \in \{0,1\}$ such that $(P, x, y, T, H_P, \mathsf{crs}) \in \mathcal{L}_{\mathcal{TM}}$, we have

$$\Pr[\mathsf{stSNARG.V}(\mathsf{crs}, x, y, H_P, \Pi) = 1 \big| \mathsf{crs} \leftarrow \mathsf{stSNARG.Setup}(1^\lambda, 1^T), H_P \leftarrow \mathsf{stSNARG.TrustHash}(\mathsf{crs}, P), \Pi \leftarrow \mathsf{stSNARG.P}(\mathsf{crs}, P, x, y, H_P)] = 1.$$

- **Efficiency.** $\mathsf{stSNARG.Setup}$ runs in time $\mathsf{poly}(\lambda, T)$, $\mathsf{stSNARG.TrustHash}$ runs in time $\mathsf{poly}(\lambda, |P|, T)$, $\mathsf{stSNARG.P}$ runs in time $\mathsf{poly}(\lambda, |x|, |P|, T)$ and outputs a proofs of length $\mathsf{poly}(\lambda, \log T)$, and $\mathsf{stSNARG.V}$ runs in time $\mathsf{poly}(\lambda, \log T)$.
- **Soundness.** For every PPT adversary $\mathcal{A} \coloneqq (\mathcal{A}_1, \mathcal{A}_2)$ and the tuple $T = T(\lambda), n = n(\lambda), m = m(\lambda)$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that for every $\lambda \in \mathbb{N}$,

$$\Pr[\mathsf{stSNARG.V}(\mathsf{crs}, x, y, H_P, \Pi) = 1 \wedge (P, x, y, T, H_P, \mathsf{crs}) \notin \mathcal{L}_{\mathcal{TM}} \big|, \mathsf{crs} \leftarrow \mathsf{stSNARG.Setup}(1^\lambda, 1^T), (P, \mathsf{aux}) \leftarrow \mathcal{A}_1(1^\lambda, \mathsf{crs}), H_P \leftarrow \mathsf{stSNARG.TrustHash}(\mathsf{crs}, P), (x, y, \Pi) \leftarrow \mathcal{A}_2(\mathsf{crs}, P, H_P, \mathsf{aux})] \leq \mathsf{negl}(\lambda).$$

Protocol 1 (Semi-Trusted SNARG)**.**

- $\mathsf{stSNARG.Setup}(1^\lambda, 1^T)$:
  - $\mathsf{SE}.K_{\mathsf{even}} \leftarrow \mathsf{SE.Gen}(1^\lambda, 1^{M_{\lambda,T}}, 1^{L_\lambda})$[a]
  - $\mathsf{SE}.K_{\mathsf{odd}} \leftarrow \mathsf{SE.Gen}(1^\lambda, 1^M, 1^L)$
  - $\mathsf{BARG.crs} \leftarrow \mathsf{BARG.Gen}(1^\lambda, 1^{T+1}, 1^{|C_{\mathsf{index}}|})$
  - $\mathsf{dk} \leftarrow \mathsf{HT.Gen}(1^\lambda)$
  - return $\mathsf{crs} := (\mathsf{SE}.K_{\mathsf{even}}, \mathsf{SE}.K_{\mathsf{odd}}, \mathsf{BARG.crs}, \mathsf{dk})$.
- $\mathsf{stSNARG.TrustHash}(\mathsf{crs}, P)$
  - $(\mathsf{tree}_0^2, \mathsf{rt}_0^2) \leftarrow \mathsf{HT.Hash}(\mathsf{dk}, P)$, $H_P \leftarrow \mathsf{rt}_0^2$
  - return H_P.
- $\mathsf{stSNARG.P}(\mathsf{crs}, P, x, y, H_P)$:
  - $\square :=$ empty string
  - $(\mathsf{tree}_0^1, \mathsf{rt}_0^1) \leftarrow \mathsf{HT.Hash}(\mathsf{dk}, x)$, $(\mathsf{tree}_0^2, \mathsf{rt}_0^2) \leftarrow \mathsf{HT.Hash}(\mathsf{dk}, P)$, $(\mathsf{tree}_0^3, \mathsf{rt}_0^3) \leftarrow \mathsf{HT.Hash}(\mathsf{dk}, \square)$
  - initialize s with the start state of $\mathcal{TM}$
  - $\mathsf{st}_0 := (0, 0, 0, \mathsf{s})$
  - $\mathsf{h}_0 := (\mathsf{st}_0, \mathsf{rt}_0^1, \mathsf{rt}_0^2, \mathsf{rt}_0^3)$
  - for every $i = 1$ to T,
    $\mathsf{rt}_i^1 \leftarrow \mathsf{rt}_{i-1}^1, \mathsf{rt}_i^2 \leftarrow \mathsf{rt}_{i-1}^2$
    $(l_i^1, l_i^2, l_i^3) \leftarrow \mathsf{StepR}(\mathsf{st}_{i-1})$
    $\left\{(b_i^j, \Pi_i^j) \leftarrow \mathsf{HT.Read}(\mathsf{tree}_{i-1}^j, l_i^j)\right\}_{j\in[3]}$
    $(b_i', l_i', \mathsf{st}_i) \leftarrow \mathsf{StepW}(\mathsf{st}_{i-1}, b_i^1, b_i^2, b_i^3)$
    $(\mathsf{tree}_i^3, \mathsf{rt}_i^3, \Pi_i') \leftarrow \mathsf{HT.Write}(\mathsf{tree}_{i-1}^3, l_i'^3, b_i'^3)$
    $\mathsf{h}_i \leftarrow (st_i, \mathsf{rt}_i^1, \mathsf{rt}_i^2, \mathsf{rt}_i^3)$
  - $A := \left(\mathsf{h}_0, \left(\mathsf{h}_1, \{b_1^j, \Pi_1^j\}_{j\in[3]}, \Pi_1'\right), \ldots, \left(\mathsf{h}_T, \{b_T^j, \Pi_T^j\}_{j\in[3]}, \Pi_T'\right)\right)$
  - $c_{\mathsf{even}} \leftarrow \mathsf{SE.Hash}\left(\mathsf{SE}.K_{\mathsf{even}}, A\right)$ and $c_{\mathsf{odd}} \leftarrow \mathsf{SE.Hash}\left(\mathsf{SE}.K_{\mathsf{odd}}, A\right)$
  - $c := (c_{\mathsf{even}}, c_{\mathsf{odd}})$
  - $I_x \leftarrow \{[i_1, i_2] \big| A[i_1, i_2] = x\}$
  - $\rho_{\mathsf{h}_0} \leftarrow \mathsf{SE.Open}(\mathsf{SE}.K_{\mathsf{even}}, A, I_{\mathsf{h}_0})$[b]
  - for every $i \leq [\lfloor T/2 \rfloor]$,
    for $B \in \{\mathsf{h}_{2i}, \{b_{2i}^j, \Pi_{2i}^j\}_{j\in[3]}, \Pi_{2i}'\}$, $\rho_B \leftarrow \mathsf{SE.Open}(\mathsf{SE}.K_{\mathsf{even}}, A, I_B)$
  - for every $i \leq [\lfloor T/2 \rfloor]$,
    for $B \in \{\mathsf{h}_{2i+1}, \{b_{2i+1}^j, \Pi_{2i+1}^j\}_{j\in[3]}, \Pi_{2i+1}'\}$, $\rho_B := \mathsf{SE.Open}(\mathsf{SE}.K_{\mathsf{odd}}, A, I_B)$
  - Let C_{index} be as defined in Figure 3
  - $\Pi := \mathsf{BARG.P}\left(\mathsf{crs}, C_{\mathsf{index}}, \mathsf{h}_0, \{\mathsf{h}_{i-1}, \mathsf{h}_i, \{b_i^j, \Pi_i^j\}_{j\in[3]}, \Pi_i', \rho_{\mathsf{h}_{i-1}}, \rho_{\mathsf{h}_i}, \{\rho_{b_i^j}, \rho_{\Pi_i^j}\}_{j\in[3]}, \rho_{\Pi_i'}\}_{i\in[T]}\right)$
  - return (c, Π)[c].
- $\mathsf{stSNARG.V}(\mathsf{crs}, (x, y), H_P, (c, \Pi))$:
  - Compute C_{index}
  - return 1 if and only if $\mathsf{BARG.V}(\mathsf{BARG.crs}, C_{\mathsf{index}}, \Pi) = 1$.

[a] $M_{\lambda,T} = O(T\,\mathsf{poly}(\lambda))$ and $L_\lambda = O(\mathsf{poly}(\lambda))$ are arbitrary and efficiently computable values which can be fixed in advance and hardcoded to the Setup algorithm during instantiation. We ignore the subscripts and often use M and L for simpler notation.

[b] Note that for simplification, we abuse notation here by specifying opening to more than a single bit.

[c] We often abuse notation and use (c, Π) to denote a proof. This can be done without loss of generalization by defining a new proof $\Pi' = (c \| \Pi)$.

Figure 2: Semi-Trusted SNARG

Circuit 1 (Circuit C_{index})**.**

- **Hard-coded:** $y, c, \mathsf{start}, \phi, \mathsf{SE}.K_{\mathsf{even}}, \mathsf{SE}.K_{\mathsf{odd}}, T, H_P, H_x \coloneqq \mathsf{HT.Hash}(\mathsf{dk}, x)$
- **Input:**

  $\big(i, (\mathsf{h}_i \coloneqq (\mathsf{st}_i, \mathsf{rt}_i^1, \mathsf{rt}_i^2, \mathsf{rt}_i^3), \rho_{\mathsf{h}_i})\big)$, if $i = 0$

  $\Big(i, (\{\mathsf{h}_{i-1}, \mathsf{h}_i, \{b_i^j, \Pi_i^j\}_{j\in[3]}, \Pi_i', \rho_{\mathsf{h}_{i-1}}, \rho_{\mathsf{h}_i}, \{\rho_{b_i^j}, \rho_{\Pi_i^j}\}_{j\in[3]}, \rho_{\Pi_i'}\})\Big)$, $\forall i \in [T]$

- **Output:** return 1 if and only if
  - if $i = 0$
    a. $\mathsf{st}_0 = \mathsf{start}$
    b. $H_x = \mathsf{rt}_0^1$
    c. $H_P = \mathsf{rt}_0^2$
    d. $\mathsf{HT.Hash}(\mathsf{dk}, \square)$ has rt_0^3 as root
  - else
    - if i is even:
      a. $\mathsf{SE.Verify}(\mathsf{SE}.K_{\mathsf{odd}}, c_{\mathsf{odd}}, \mathsf{h}_{i-1}, \rho_{\mathsf{h}_{i-1}}) = 1$
      b. $\mathsf{SE.Verify}(\mathsf{SE}.K_{\mathsf{even}}, c_{\mathsf{even}}, \mathsf{h}_i, \rho_{\mathsf{h}_i}) = 1$
      c. $\Big\{\mathsf{SE.Verify}(\mathsf{SE}.K_{\mathsf{even}}, c_{\mathsf{even}}, b_i^j, \rho_{b_i^j}) = 1\Big\}_{j\in[3]}$
      d. $\Big\{\mathsf{SE.Verify}(\mathsf{SE}.K_{\mathsf{even}}, c_{\mathsf{even}}, \Pi_i^j, \rho_{\Pi_i^j}) = 1\Big\}_{j\in[3]}$
      e. $\mathsf{SE.Verify}(\mathsf{SE}.K_{\mathsf{even}}, c_{\mathsf{even}}, \Pi_i', \rho_{\Pi_i'}) = 1$
    - if i is odd:
      a. $\mathsf{SE.Verify}(\mathsf{SE}.K_{\mathsf{even}}, c_{\mathsf{even}}, \mathsf{h}_{i-1}, \rho_{\mathsf{h}_{i-1}}) = 1$
      b. $\mathsf{SE.Verify}(\mathsf{SE}.K_{\mathsf{odd}}, c_{\mathsf{odd}}, \mathsf{h}_i, \rho_{\mathsf{h}_i}) = 1$
      c. $\Big\{\mathsf{SE.Verify}(\mathsf{SE}.K_{\mathsf{odd}}, c_{\mathsf{odd}}, b_i^j, \rho_{b_i^j}) = 1\Big\}_{j\in[3]}$
      d. $\Big\{\mathsf{SE.Verify}(\mathsf{SE}.K_{\mathsf{odd}}, c_{\mathsf{odd}}, \Pi_i^j, \rho_{\Pi_i^j}) = 1\Big\}_{j\in[3]}$
      e. $\mathsf{SE.Verify}(\mathsf{SE}.K_{\mathsf{odd}}, c_{\mathsf{odd}}, \Pi_i', \rho_{\Pi_i'}) = 1$
    - $\phi(\mathsf{h}_{i-1}, \mathsf{h}_i, \{b_i^j, \Pi_i^j\}_{j\in[3]}, \Pi_i') = 1$
    - if $i = T$
      a. $\mathsf{HT.Hash}(\mathsf{dk}, y)$ has rt_T^3 as root.
      b. st_T indeed encodes the accept state.

Figure 3: Circuit C_{index}

5.1 Our Construction

Our construction is formulated similar to that of [CJJ21b]. Specifically, we use the notion of non-interactive BARG for index language and SE Hash functions in our scheme.

Setup for Universal Turing Machine. For a cleaner analysis, we assume without loss of generality that $\mathcal{TM}$ consists of three tapes, namely, $\mathsf{Tp}_1, \mathsf{Tp}_2, \mathsf{Tp}_3$. Tp_1 and Tp_2 are read only tapes that store x and P respectively. Tp_3 is the work tape which is initialized with $\square$ to denote an empty string.

Transition steps for $\mathcal{TM}$. $\mathcal{TM}$'s state information along with the head locations of the three tapes are encoded as st. To handle Turing Machines with arbitrarily long tapes, we encode $\{\mathsf{Tp}_i\}_{i\in[3]}$ using three Hash Trees as defined in Section 3.3 and produce tree roots $\mathsf{rt}^1, \mathsf{rt}^2, \mathsf{rt}^3$ respectively.

Let the each intermediate transition state of $\mathcal{TM}$ be encoded as $h_i \coloneqq (\mathsf{st}_i, \mathsf{rt}_i^1, \mathsf{rt}_i^2, \mathsf{rt}_i^3)$ for $i \in [T]$. A single step of $\mathcal{TM}$ can be interpreted in the manner described below which is similar to one described for a RAM in [KPY19]. We break down the step function at the i^{th} stage into two deterministic polynomial time algorithms:

- StepR: On input st_{i-1} of $\mathcal{TM}$, outputs head positions $l_{i-1}^1, l_{i-1}^2, l_{i-1}^3$ which denote the memory locations of $\mathsf{Tp}_1, \mathsf{Tp}_2, \mathsf{Tp}_3$ which $\mathcal{TM}$ in the current state st_{i-1} would read from.

- StepW: On input st_{i-1}, and bits $b^1_{i-1}, b^2_{i-1}, b^3_{i-1}$ outputs bit b', location l' and st_i such that $\mathcal{TM}$ upon reading $b^1_{i-1}, b^2_{i-1}, b^3_{i-1}$ at locations $l^1_{i-1}, l^2_{i-1}, l^3_{i-1}$ using $\mathsf{HT.Read}$, would write b' at location l' of Tp_3, thereby transition to new state st_i.

Now, we translate the i^{th} single step of $\mathcal{TM}$ to the circuit ϕ which is defined such that on input digests $h_{i-1} := (\mathsf{st}_{i-1}, \mathsf{rt}^1_{i-1}, \mathsf{rt}^2_{i-1}, \mathsf{rt}^3_{i-1})$ and $h_i := (\mathsf{st}_i, \mathsf{rt}^1_i, \mathsf{rt}^2_i, \mathsf{rt}^3_i)$, bits b^1_i, b^2_i, b^3_i, and proofs $\Pi^1_i, \Pi^2_i, \Pi^3_i, \Pi'_i$, $\phi(h_{i-1}, h_i, b^1_i, b^2_i, b^3_i, \Pi^1_i, \Pi^2_i, \Pi^3_i, \Pi'_i) = 1$ if and only if the following hold:

1. $(l^1_i, l^2_i, l^3_i) \leftarrow \mathsf{StepR}(\mathsf{st}_{i-1})$
2. $(b', l', \mathsf{st}') \leftarrow \mathsf{StepW}(\mathsf{st}_{i-1}, b^1_i, b^2_i, b^3_i)$
3. $\mathsf{st}' = \mathsf{st}_i$
4. $\mathsf{HT.VerRead}(\mathsf{dk}, \mathsf{rt}^1_{i-1}, l^1_i, b^1_i, \Pi^1_i) = 1$
5. $\mathsf{HT.VerRead}(\mathsf{dk}, \mathsf{rt}^2_{i-1}, l^2_i, b^2_i, \Pi^2_i) = 1$
6. $\mathsf{HT.VerRead}(\mathsf{dk}, \mathsf{rt}^3_{i-1}, l^3_i, b^3_i, \Pi^3_i) = 1$
7. $\mathsf{rt}^1_i = \mathsf{rt}^1_{i-1}$
8. $\mathsf{rt}^2_i = \mathsf{rt}^2_{i-1}$
9. $\mathsf{HT.VerWrite}(\mathsf{dk}, \mathsf{rt}^3_{i-1}, l', b', \mathsf{rt}^3_i, \Pi'_i) = 1$

Here, dk denote the hash keys used to build the three hash trees. Note that the efficiency of hash tree implies that ϕ can be constructed such that it can represented as a formula in $L = \mathsf{poly}(\lambda)$ variables. For the T steps of $\mathcal{TM}$, we have the following formula over $M = O(L \cdot T)$ variables:

$$\Phi(\mathsf{h}_0, \{h_i, b^1_i, b^2_i, b^3_i, \Pi^1_i, \Pi^2_i, \Pi^3_i, \Pi'_i\}_{i\in[T]}) = \bigwedge_{i\in[T]} \phi(h_{i-1}, h_i, b^1_i, b^2_i, b^3_i, \Pi^1_i, \Pi^2_i, \Pi^3_i, \Pi'_i)$$

Following the techniques in [CJJ21b], we use a combination of SE Hash along with ϕ to produce the circuit for index languages (Section 3.2).

Our semi-trusted SNARG scheme is given in Figure 2 and the corresponding index language circuit is shown as Figure 3.

Theorem 5.1. *Assuming the existence of Somewhere Extractable Hash functions, non-interactive Batch Arguments for Index Languages, and Collision Resistant Hash Trees as described in section 3,, Figure 2 is a publicly verifiable non-interactive semi-trusted SNARG with CRS size, proof size and verifier time* $\mathsf{poly}(\lambda, \log T)$ *and prover run time being* $\mathsf{poly}(\lambda, T)$.

Completeness. Here we give a sketch arguing completeness of our scheme. Our construction in Figure 2 tells that

$$\Pr[\mathsf{stSNARG.V}(\mathsf{crs}, x, y, H_P, \Pi) = 1 \big| \mathsf{crs} \leftarrow \mathsf{stSNARG.Setup}(1^\lambda, 1^T), H_P \leftarrow \mathsf{stSNARG.TrustHash}(\mathsf{crs}, P), \Pi \leftarrow \mathsf{stSNARG.P}(\mathsf{crs}, P, x, y, H_P)] = \Pr[\mathsf{BARG.V}(\mathsf{BARG.crs}, C_{\mathsf{index}}, \Pi) = 1 \big| \mathsf{crs} \leftarrow \mathsf{stSNARG.Setup}(1^\lambda, 1^T), H_P \leftarrow \mathsf{stSNARG.TrustHash}(\mathsf{crs}, P), \Pi \leftarrow \mathsf{stSNARG.P}(\mathsf{crs}, P, x, y, H_P)]$$

where C_{index} is the index circuit as shown in Figure 3. Observing $\mathsf{stSNARG.P}$ algorithm in our scheme tells it is sufficient to show that if the prover is honest and uses a valid witness, then $(C_{\mathsf{index}}, i) \in \mathcal{L}_{\mathsf{index}}, \forall i \in \{0\} \cup [T]$. If we can argue that this is indeed the case, then the completeness of BARG gives the desired result.

If $(P, x, y, T, H_P, \mathsf{crs}) \in \mathcal{L}_{\mathcal{TM}}$, then $(C_{\mathsf{index}}, 0) \in \mathcal{L}_{\mathsf{index}}$ is trivially true by observation. Now, let us look at $(C_{\mathsf{index}}, 1)$. We start by analysing that $\phi(\mathsf{h}_0, \mathsf{h}_1, \{b^j_1, \Pi^j_1\}_{j\in[3]}, \Pi'_1) = 1$ is true. $\{\mathsf{rt}^i_1 = \mathsf{rt}^i_0\}_{i\in[2]}$ follow from the read-only nature of tapes $\mathsf{Tp}_1, \mathsf{Tp}_2$. Since, $\left\{(b^j_1, \Pi^j_1) \leftarrow \mathsf{HT.Read}(\mathsf{tree}^j_0, l^j_1)\right\}_{j\in[3]}$, the hash tree completeness of read ensures that $\{\mathsf{HT.VerRead}(\mathsf{dk}, \mathsf{rt}^i_0, l^i_1, b^i_1, \Pi^i_1) = 1\}_{i\in[3]} = 1$ and $\{\mathsf{Tp}_i[l^i_1] = b^i_1\}_{i\in[3]}$. This along with the correctness of Turing Machine StepR function implies that b^1_1, b^2_1, b^3_1 are indeed the correct input for the StepW function of $\mathcal{TM}$. Finally, $(\mathsf{tree}^3_1, \mathsf{rt}^3_1, \Pi'_1) \leftarrow \mathsf{HT.Write}(\mathsf{tree}^3_0, l'_1, b'_1)$ implies $\mathsf{HT.VerWrite}(\mathsf{dk}, \mathsf{rt}^3_0, l', b', \mathsf{rt}^3_1, \Pi'_1) = 1$ from the hash tree completeness of write property. The same property also ensures that Tp_3 changes only at the l'^{th} memory location. When paired with the correctness of StepW, we get that $\mathsf{st}_1 = \mathsf{st}'$

The completeness of the SE hash implies that the verification algorithm certainly accepts all the local openings. Thus, $(C_{\mathsf{index}}, 1) \in \mathcal{L}^{\mathsf{index}}$. Now, $(C_{\mathsf{index}}, T) \in \mathcal{L}^{\mathsf{index}}$ because $\mathcal{TM}$ accept (P, x, y) in T steps. We can show in a similar manner that for all other i, $(C_{\mathsf{index}}, i) \in \mathcal{L}^{\mathsf{index}}$. This proves the completeness of the scheme in Figure 2.

Efficiency.

- Runtime of $\mathsf{stSNARG.Setup}$ is $\mathsf{poly}(\lambda, T)$. This follows from the efficiency of underlying primitives.
- $\mathsf{stSNARG.TrustHash}$ computes H_P in time $|P| \cdot \mathsf{poly}(\lambda)$ which is $\mathsf{poly}(|P|, \lambda)$.
- $|C_{\mathsf{index}}| = \mathsf{poly}(\lambda, \log T)$. This follows from the efficiency of the SE hash and the efficiency of hash tree construction.
- CRS Size: By the corresponding properties of the underlying primitives, $|\mathsf{crs}| = \mathsf{poly}(\lambda, \log T)$.
- The prover's computation time is dominated by the hashes corresponding to x, P and the Turing Machine step functions that is run T times. This requires a total time of $\mathsf{poly}(\lambda, |x|) + \mathsf{poly}(\lambda, |P|) + \mathsf{poly}(\lambda, T) = \mathsf{poly}(\lambda, |x|, |P|, T)$.
- Proof Length: $|c| + |\Pi| = \mathsf{poly}(\lambda, \log T) + \mathsf{poly}(\lambda, \log T, |C_{\mathsf{index}}|) = \mathsf{poly}(\lambda, \log T)$.
- Verifier Time: Time taken to compute C_{index} and verify the BARG. This is $\mathsf{poly}(\lambda, \log T, |C_{\mathsf{index}}|) = \mathsf{poly}(\lambda, \log T)$.

Soundness. Let us assume for the sake of contradiction that our scheme in Figure 2 is not sound, i.e., there exists a PPT adversary $\mathcal{A} \coloneqq (\mathcal{A}_1, \mathcal{A}_2)$, a value T and a polynomial function $\mathsf{poly}(\lambda)$ such that for infinitely many values of $\lambda \in \mathbb{N}$,

$$\Pr[\mathsf{G}^{\mathcal{A}} = 1] \geq \frac{1}{\mathsf{poly}(\lambda)},$$

where $\mathcal{A}$ plays Game G described below

Real Game G

- $\mathsf{crs} \leftarrow \mathsf{stSNARG.Setup}(1^\lambda, 1^T)$
- $(P, \mathsf{aux}) \leftarrow \mathcal{A}_1(1^\lambda, \mathsf{crs})$
- $H_P \leftarrow \mathsf{stSNARG.TrustHash}(\mathsf{crs}, P)$
- $((x, y)(c, \Pi)) \leftarrow \mathcal{A}_2(\mathsf{crs}, P, H_P, \mathsf{aux})$
- if $\mathsf{stSNARG.V}\,(\mathsf{crs}, (x, y), H_P, (c, \Pi)) = 1 \wedge ((x, y), T, P, H_P, \mathsf{crs}) \notin \mathcal{L}_{\mathcal{TM}}$, return 1
- else return 0

Let S_i denote the following set:

$$S_i = \begin{cases} \mathsf{h}_0 & \text{if } i = 0 \\ \left\{\mathsf{h}_i, \{b_i\}_{j\in[3]}, \{\Pi_i\}_{j\in[3]}, \Pi'_i\right\} & \text{if } i \in [T] \end{cases}$$

Let D denote the string $\left(\mathsf{h}_0, \left\{\mathsf{h}_i, \{b_i\}_{j\in[3]}, \{\Pi_i\}_{j\in[3]}, \Pi'_i\right\}_{i\in[T]}\right)$. $I_{S_i} \subset |D|$ denotes the following:

$$I_{S_i} = \left\{[a, b] \middle| a, b \in |D|, D[a, b] = S_i\right\}.$$

In game G, say we have $(\mathsf{tree}_0^1, \mathsf{rt}_0^1) \leftarrow \mathsf{HT.Hash}(\mathsf{dk}, x)$, $(\mathsf{tree}_0^2, \mathsf{rt}_0^2) \leftarrow \mathsf{HT.Hash}(\mathsf{dk}, P)$, $(\mathsf{tree}_0^3, \mathsf{rt}_0^3) \leftarrow \mathsf{HT.Hash}(\mathsf{dk}, \square)$. Also, let $\mathsf{st}_0 \coloneqq (0, 0, 0, \mathsf{s})$, where s is the start state of $\mathcal{TM}$. We say that $\bar{\mathsf{h}}_0 \coloneqq (\mathsf{st}_0, \mathsf{rt}_0^1, \mathsf{rt}_0^2, \mathsf{rt}_0^3)$ defines a unique "true" digest for the starting step of $\mathcal{TM}$.

If $\mathsf{stSNARG.V}(\mathsf{crs}, x, H_P, c, \Pi) = 1$, then Algorithm $Step(x, P, \mathsf{crs}, i)$ in Figure 4 computes the unique true digest $\bar{\mathsf{h}}_i$ after the i^{th} Turing Machine Step along with the other uniquely correct values of the set $\bar{S}_i \coloneqq \{\bar{\mathsf{h}}_i, \{\bar{b}_i\}_{j\in[3]}, \{\bar{\Pi}_i\}_{j\in[3]}, \bar{\Pi}'_i\}$. We use the notation $Step(x, P, \mathsf{crs}, i).x$ to denote $x \in \bar{S}_i$. We proceed by performing an induction on the following sequence outer hybrid games G_i, i from 1 to T. We use a sequence

of inner hybrid games to transition between subsequent outer hybrids. Our induction hypothesis is that, under suitable assumptions, for all $i \in 1$ to T, there exists a negligible function λ such that,

$$\Pr[\mathsf{G}^{\mathcal{A}} = 1] \leq \Pr[\mathsf{G}_i^{\mathcal{A}} = 1] + \mathsf{negl}(\lambda).$$

Algorithm $Step(x, y, P, \mathsf{crs}, i)$

- $\square \coloneqq$ empty string
- $(\mathsf{tree}_0^1, \mathsf{rt}_0^1) \coloneqq \mathsf{HT.Hash}(\mathsf{dk}, x)$, $(\mathsf{tree}_0^2, \mathsf{rt}_0^2) \coloneqq \mathsf{HT.Hash}(\mathsf{dk}, P)$, $(\mathsf{tree}_0^3, \mathsf{rt}_0^3) \coloneqq \mathsf{HT.Hash}(\mathsf{dk}, \square)$
- initialize s with the start state of $\mathcal{TM}$
- $\mathsf{st}_0 \coloneqq (0, 0, 0, \mathsf{s})$
- $\bar{\mathsf{h}}_0 \coloneqq (\mathsf{st}_0, \mathsf{rt}_0^1, \mathsf{rt}_0^2, \mathsf{rt}_0^3)$
- if $i = 0$, return $\bar{S}_0 \coloneqq (\mathsf{st}_0, \mathsf{rt}_0^1, \mathsf{rt}_0^2, \mathsf{rt}_0^3)$
- else
  - for $\mathsf{count} = 1$ to i,
    $(l^1_{\mathsf{count}}, l^2_{\mathsf{count}}, l^3_{\mathsf{count}}) \leftarrow \mathsf{StepR}(\mathsf{st}_{\mathsf{count}-1})$
    $\left\{(b^k_{\mathsf{count}}, \Pi^k_{\mathsf{count}}) \coloneqq \mathsf{HT.Read}(\mathsf{tree}^k_{\mathsf{count}-1}, l^k_{\mathsf{count}})\right\}_{k\in[3]}$
    $(b'^3_{\mathsf{count}}, l'^3_{\mathsf{count}}, \mathsf{st}_{\mathsf{count}}) \coloneqq \mathsf{StepW}(\mathsf{st}_{\mathsf{count}-1}, b^1_{\mathsf{count}}, b^2_{\mathsf{count}}, b^3_{\mathsf{count}})$
    $(\mathsf{tree}^3_{\mathsf{count}}, \mathsf{rt}^3_{\mathsf{count}}, \Pi'_{\mathsf{count}}) \coloneqq \mathsf{HT.Write}(\mathsf{tree}^3_{\mathsf{count}-1}, l'^3_{\mathsf{count}}, b'^3_{\mathsf{count}})$
  - $\bar{\mathsf{h}}_i \coloneqq (\mathsf{st}_i, \mathsf{rt}_i^1, \mathsf{rt}_i^2, \mathsf{rt}_i^3)$
  - $\bar{b}_i \coloneqq (b_i^1, b_i^2, b_i^3)$
  - $\bar{l}_i \coloneqq (l_i^1, l_i^2, l_i^3)$
  - $\bar{\mathsf{rt}}_i \coloneqq \mathsf{rt}_{i-1}^1, \mathsf{rt}_{i-1}^2, \mathsf{rt}_i^3$
  - $\bar{\Pi}_i \coloneqq (\Pi_i^1, \Pi_i^2, \Pi_i^3, \Pi'_i)$
  - return $\bar{S}_i \coloneqq (\bar{\mathsf{h}}_i, \bar{b}_i, \bar{\mathsf{rt}}_i, \bar{\Pi}_i)$

Figure 4: Turing Machine i^{th} step.

Intuitively, the i^{th} game G_i is similar to the real life soundness game with the following two changes: (1) The key generation for the SE hash and BARG is done in the trapdoor mode at the i^{th} game. This allows for extractability of the i^{th} block of the string D from the commitment c. (2) The adversary wins the game if they break the soundness assumption as the real life game G and the extracted block is indeed the correct one.

Outer Hybrid Game G_i

- if i is even
  $\mathsf{SE}.K_{\mathsf{even}} \leftarrow \mathsf{SE.TGen}(1^\lambda, 1^M, I_{S_i})$
  $\mathsf{SE}.K_{\mathsf{odd}} \leftarrow \mathsf{SE.TGen}(1^\lambda, 1^M, I_{S_{i-1}})$
- if i is odd
  $\mathsf{SE}.K_{\mathsf{even}} \leftarrow \mathsf{SE.TGen}(1^\lambda, 1^M, I_{S_{i-1}})$
  $\mathsf{SE}.K_{\mathsf{odd}} \leftarrow \mathsf{SE.TGen}(1^\lambda, 1^M, I_{S_i})$
- $\mathsf{BARG.crs} \leftarrow \mathsf{BARG.TGen}(1^\lambda, 1^{T+1}, 1^{|C_{\mathsf{index}}|}, i)$
- $\mathsf{dk} \leftarrow \mathsf{HT.Gen}(1^\lambda)$

- $\mathsf{crs} := (\mathsf{SE}.K_{\mathsf{even}}, \mathsf{SE}.K_{\mathsf{odd}}, \mathsf{BARG.crs}, \mathsf{dk})$.
- $(P, \mathsf{aux}) \leftarrow \mathcal{A}_1(1^\lambda, \mathsf{crs})$
- $H_P \leftarrow \mathsf{stSNARG.TrustHash}(\mathsf{crs}, P)$
- $((x, y), (c, \Pi)) \leftarrow \mathcal{A}_2(\mathsf{crs}, P, \mathsf{aux})$
- Parse c as $(c_{\mathsf{odd}}, c_{\mathsf{even}})$
- if i is even and $i \neq 0$

  $(\mathsf{h}_i, \{b_i^k\}_{k\in[3]}, \{\Pi_i^k\}_{k\in[3]}, \Pi_i') \leftarrow \mathsf{SE.Ext}_{\mathsf{even}}(c_{\mathsf{even}}, \mathsf{SE}.K_{\mathsf{even}})$

  $(\mathsf{h}_{i-1}, \{b_{i-1}^k\}_{k\in[3]}, \{\Pi_{i-1}^k\}_{k\in[3]}, \Pi_{i-1}') \leftarrow \mathsf{SE.Ext}_{\mathsf{odd}}(c_{\mathsf{odd}}, \mathsf{SE}.K_{\mathsf{odd}})$
- if i is odd

  $(\mathsf{h}_i, \{b_i^k\}_{k\in[3]}, \{\Pi_i^k\}_{k\in[3]}, \Pi_i') \leftarrow \mathsf{SE.Ext}_{\mathsf{odd}}(c_{\mathsf{odd}}, \mathsf{SE}.K_{\mathsf{odd}})$

  if $i - 1 > 0$ then $(\mathsf{h}_{i-1}, \{b_{i-1}^k\}_{k\in[3]}, \{\Pi_{i-1}^k\}_{k\in[3]}, \Pi_{i-1}') \leftarrow \mathsf{SE.Ext}_{\mathsf{even}}(c_{\mathsf{even}}, \mathsf{SE}.K_{\mathsf{even}})$

  if $i - 1 = 0$ then $\mathsf{h}_0 \leftarrow \mathsf{SE.Ext}_{\mathsf{even}}(c_{\mathsf{even}}, \mathsf{SE}.K_{\mathsf{even}})$
- if $\mathsf{stSNARG.V}(\mathsf{crs}, (x, y), H_P, (c, \Pi)) = 1 \wedge ((x, y), T, P, H_P, \mathsf{crs}) \notin \mathcal{L}_{\mathcal{TM}} \wedge \mathsf{h}_i = \mathit{Step}(x, y, P, \mathsf{crs}, i).\bar{\mathsf{h}}_i$, return 1
- else return 0

Base Case: Assuming key indistinguishability and soundness of SE hash and BARG, we need to show that $\Pr[\mathsf{G}^{\mathcal{A}} = 1] \leq \Pr[\mathsf{G}_1^{\mathcal{A}} = 1] + \mathsf{negl}(\lambda)$.

We proceed by using a sequence of hybrids via an intermediate game G_0.

Hybrid Game G_a

- $\mathsf{SE}.K_{\mathsf{even}} \leftarrow \mathsf{SE.TGen}(1^\lambda, 1^M, I_{S_0})$
- $\mathsf{SE}.K_{\mathsf{odd}} \leftarrow \mathsf{SE.Gen}(1^\lambda, 1^M)$
- $\mathsf{BARG.crs} \leftarrow \mathsf{BARG.Gen}(1^\lambda, 1^{T+1}, 1^{|C_{\mathsf{index}}|})$
- $\mathsf{dk} \leftarrow \mathsf{HT.Gen}(1^\lambda)$
- $\mathsf{crs} := (\mathsf{SE}.K_{\mathsf{even}}, \mathsf{SE}.K_{\mathsf{odd}}, \mathsf{BARG.crs}, \mathsf{dk})$.
- $(P, \mathsf{aux}) \leftarrow \mathcal{A}_1(1^\lambda, \mathsf{crs})$
- $H_P \leftarrow \mathsf{stSNARG.TrustHash}(\mathsf{crs}, P)$
- $((x, y), (c, \Pi)) \leftarrow \mathcal{A}_2(\mathsf{crs}, P, \mathsf{aux})$
- if $\mathsf{stSNARG.V}(\mathsf{crs}, (x, y), H_P, (c, \Pi)) = 1 \wedge ((x, y), T, P, H_P, \mathsf{crs}) \notin \mathcal{L}_{\mathcal{TM}}$, return 1
- else return 0

Hybrid Game G_b

- $\mathsf{SE}.K_{\mathsf{even}} \leftarrow \mathsf{SE.TGen}(1^\lambda, 1^M, I_{S_0})$
- $\mathsf{SE}.K_{\mathsf{odd}} \leftarrow \mathsf{SE.TGen}(1^\lambda, 1^M, I_{S_1})$
- $\mathsf{BARG.crs} \leftarrow \mathsf{BARG.Gen}(1^\lambda, 1^{T+1}, 1^{|C_{\mathsf{index}}|})$
- $\mathsf{dk} \leftarrow \mathsf{HT.Gen}(1^\lambda)$
- $\mathsf{crs} := (\mathsf{SE}.K_{\mathsf{even}}, \mathsf{SE}.K_{\mathsf{odd}}, \mathsf{BARG.crs}, \mathsf{dk})$.
- $(P, \mathsf{aux}) \leftarrow \mathcal{A}_1(1^\lambda, \mathsf{crs})$
- $H_P \leftarrow \mathsf{stSNARG.TrustHash}(\mathsf{crs}, P)$
- $((x, y), (c, \Pi)) \leftarrow \mathcal{A}_2(\mathsf{crs}, P, \mathsf{aux})$
- if $\mathsf{stSNARG.V}(\mathsf{crs}, (x, y), H_P, (c, \Pi)) = 1 \wedge ((x, y), T, P, H_P, \mathsf{crs}) \notin \mathcal{L}_{\mathcal{TM}}$, return 1
- else return 0

Lemma 5.2. *Assuming key indistinguishability of* SE, $\left|\Pr[\mathsf{G}^{\mathcal{A}} = 1] - \Pr[\mathsf{G}_a^{\mathcal{A}} = 1]\right| \leq \mathsf{negl}(\lambda)$.

Proof. The only difference in Game G and G_a is that the key generation algorithm of the SE hash ($\mathsf{SE.Gen}$) is replaced by the trapdoor key generation ($\mathsf{SE.TGen}$).

If $\left|\Pr[\mathsf{G}^{\mathcal{A}} = 1] - \Pr[\mathsf{G}_a^{\mathcal{A}} = 1]\right| > \mathsf{negl}(\lambda)$, then one can construct a PPT adversary $\mathcal{B}$ that breaks the key indistinguishability of SE using I_{S_0} with Key as input from the key generation algorithm of the SE hash as follows:

Adversary $\mathcal{B}$ playing SE key indistinguishability game.

- $\mathsf{SE}.K_{\mathsf{even}} \leftarrow \mathsf{Key}$
- $\mathsf{SE}.K_{\mathsf{odd}} \leftarrow \mathsf{SE.Gen}(1^\lambda, 1^M)$
- $\mathsf{BARG.crs} \leftarrow \mathsf{BARG.Gen}(1^\lambda, 1^{T+1}, 1^{|C_{\mathsf{index}}|})$

- $\mathsf{dk} \leftarrow \mathsf{HT.Gen}(1^\lambda)$
- $\mathsf{crs} := (\mathsf{SE}.K_{\mathsf{even}}, \mathsf{SE}.K_{\mathsf{odd}}, \mathsf{BARG.crs}, \mathsf{dk})$.
- $(P, \mathsf{aux}) \leftarrow \mathcal{A}_1(1^\lambda, \mathsf{crs})$
- $H_P \leftarrow \mathsf{stSNARG.TrustHash}(\mathsf{crs}, P)$
- $((x, y), (c, \Pi)) \leftarrow \mathcal{A}_2(\mathsf{crs}, P, \mathsf{aux})$
- if $\mathsf{stSNARG.V}(\mathsf{crs}, (x, y), H_P, (c, \Pi)) = 1 \wedge ((x, y), T, P, H_P, \mathsf{crs}) \notin \mathcal{L}_{\mathcal{TM}}$, return 1
- else return 0

Here, Key is either $\mathsf{SE.Gen}(1^\lambda, 1^M)$ or $\mathsf{SE.TGen}(1^\lambda, 1^M, I_{S_0})$ based on whether $\mathcal{A}$ is interacting with game G or G_a respectively. Adversary $\mathcal{B}$ breaks the key indistinguishability of SE hash if the probability that it returns 1 is significantly different when $\mathcal{A}$ interacts with the key generation algorithm in normal mode vs trapdoor mode. Now, the probability that $\mathcal{B}$ returns 1 in either case is exactly equal to the probability that $\mathcal{A}$ wins the corresponding games, hence, $\mathcal{B}$ breaks if $\left|\Pr[\mathsf{G}^{\mathcal{A}} = 1] - \Pr[\mathsf{G}_a^{\mathcal{A}} = 1]\right| \geq \mathsf{negl}(\lambda)$. This leads to a contradiction of our assumption. □

Lemma 5.3. *Assuming key indistinguishability of* SE, $\left|\Pr[\mathsf{G}_a^{\mathcal{A}} = 1] - \Pr[\mathsf{G}_b^{\mathcal{A}} = 1]\right| \leq \mathsf{negl}(\lambda)$.

This again follows from the key-indistinguishability of SE as shown in the previous lemma, hence we skip the proof.

Hybrid Game G_{ab}

- $\mathsf{SE}.K_{\mathsf{even}} \leftarrow \mathsf{SE.TGen}(1^\lambda, 1^M, I_{S_0})$
- $\mathsf{SE}.K_{\mathsf{odd}} \leftarrow \mathsf{SE.TGen}(1^\lambda, 1^M, I_{S_1})$
- $\mathsf{BARG.crs} \leftarrow \mathsf{BARG.TGen}(1^\lambda, 1^{T+1}, 1^{|C_{\mathsf{index}}|}, 0)$
- $\mathsf{dk} \leftarrow \mathsf{HT.Gen}(1^\lambda)$
- $\mathsf{crs} := (\mathsf{SE}.K_{\mathsf{even}}, \mathsf{SE}.K_{\mathsf{odd}}, \mathsf{BARG.crs}, \mathsf{dk})$.
- $(P, \mathsf{aux}) \leftarrow \mathcal{A}_1(1^\lambda, \mathsf{crs})$
- $H_P \leftarrow \mathsf{stSNARG.TrustHash}(\mathsf{crs}, P)$
- $((x, y), (c, \Pi)) \leftarrow \mathcal{A}_2(\mathsf{crs}, P, \mathsf{aux})$
- if $\mathsf{stSNARG.V}(\mathsf{crs}, (x, y), H_P, (c, \Pi)) = 1 \wedge ((x, y), T, P, H_P, \mathsf{crs}) \notin \mathcal{L}_{\mathcal{TM}}$, return 1
- else return 0

Lemma 5.4. *Assuming key indistinguishability of* BARG, $\left|\Pr[\mathsf{G}_b^{\mathcal{A}} = 1] - \Pr[\mathsf{G}_{ab}^{\mathcal{A}} = 1]\right| \leq \mathsf{negl}(\lambda)$.

Proof. The only difference in Game G_b and G_{ab} is that the key generation algorithm of the BARG (BARG.Gen) is replaced by the trapdoor key generation (BARG.TGen) at index 0.

If $\left|\Pr[\mathsf{G}_b^{\mathcal{A}} = 1] - \Pr[\mathsf{G}_{ab}^{\mathcal{A}} = 1]\right| > \mathsf{negl}(\lambda)$, then one can construct a PPT adversary $\mathcal{B}$ getting Key as input that breaks the key indistinguishability of BARG as follows:

Adversary $\mathcal{B}$ playing BARG key indistinguishability game.

- $\mathsf{SE}.K_{\mathsf{even}} \leftarrow \mathsf{SE.TGen}(1^\lambda, 1^M, I_{S_0})$
- $\mathsf{SE}.K_{\mathsf{odd}} \leftarrow \mathsf{SE.TGen}(1^\lambda, 1^M, I_{S_1})$
- $\mathsf{BARG.crs} \leftarrow \mathsf{Key}$
- $\mathsf{dk} \leftarrow \mathsf{HT.Gen}(1^\lambda)$
- $\mathsf{crs} := (\mathsf{SE}.K_{\mathsf{even}}, \mathsf{SE}.K_{\mathsf{odd}}, \mathsf{BARG.crs}, \mathsf{dk})$.
- $(P, \mathsf{aux}) \leftarrow \mathcal{A}_1(1^\lambda, \mathsf{crs})$
- $H_P \leftarrow \mathsf{stSNARG.TrustHash}(\mathsf{crs}, P)$
- $((x, y), (c, \Pi)) \leftarrow \mathcal{A}_2(\mathsf{crs}, P, \mathsf{aux})$

– if $\mathsf{stSNARG.V}(\mathsf{crs}, (x,y), H_P, (c,\Pi)) = 1 \wedge ((x,y), T, P, H_P, \mathsf{crs}) \notin \mathcal{L}_{\mathcal{TM}}$, return 1

– else return 0

Here, Key is either $\mathsf{BARG.Gen}(1^\lambda, 1^{T+1}, 1^{|C_{\mathsf{index}}|})$ or $\mathsf{BARG.TGen}(1^\lambda, 1^{T+1}, 1^{|C_{\mathsf{index}}|}, 0)$ based on whether $\mathcal{A}$ is interacting with game G_b or G_{ab} respectively. Adversary $\mathcal{B}$ breaks the key indistinguishability of BARG if the probability that it returns 1 is significantly different when $\mathcal{A}$ interacts with the key generation algorithm in normal mode vs trapdoor mode. Now, the probability that $\mathcal{B}$ returns 1 in either case is exactly equal to the probability that $\mathcal{A}$ wins its corresponding game, hence, $\mathcal{B}$ breaks if $\left|\Pr[\mathsf{G}_b^{\mathcal{A}} = 1] - \Pr[\mathsf{G}_{ab}^{\mathcal{A}} = 1]\right| \geq \mathsf{negl}(\lambda)$. This leads to a contradiction of our assumption. □

Hybrid Game G_0

– $\mathsf{SE}.K_{\mathsf{even}} \leftarrow \mathsf{SE.TGen}(1^\lambda, 1^M, I_{S_0})$

– $\mathsf{SE}.K_{\mathsf{odd}} \leftarrow \mathsf{SE.TGen}(1^\lambda, 1^M, I_{S_1})$

– $\mathsf{BARG.crs} \leftarrow \mathsf{BARG.TGen}(1^\lambda, 1^{T+1}, 1^{|C_{\mathsf{index}}|}, 0)$

– $\mathsf{dk} \leftarrow \mathsf{HT.Gen}(1^\lambda)$

– $\mathsf{crs} := (\mathsf{SE}.K_{\mathsf{even}}, \mathsf{SE}.K_{\mathsf{odd}}, \mathsf{BARG.crs}, \mathsf{dk})$.

– $(P, \mathsf{aux}) \leftarrow \mathcal{A}_1(1^\lambda, \mathsf{crs})$

– $H_P \leftarrow \mathsf{stSNARG.TrustHash}(\mathsf{crs}, P)$

– $((x,y),(c,\Pi)) \leftarrow \mathcal{A}_2(\mathsf{crs}, P, \mathsf{aux})$

– $\mathsf{h}_0 \leftarrow \mathsf{SE.Ext}_{\mathsf{even}}(c_{\mathsf{even}}, \mathsf{SE}.K_{\mathsf{even}})$

– if $\mathsf{stSNARG.V}(\mathsf{crs}, (x,y), H_P, (c,\Pi)) = 1 \wedge ((x,y), T, P, H_P, \mathsf{crs}) \notin \mathcal{L}_{\mathcal{TM}} \wedge \mathsf{h}_0 = \mathit{Step}(x, y, P, \mathsf{crs}, 0).\bar{\mathsf{h}}$, return 1

– else return 0

Lemma 5.5. *Assuming soundness of* BARG*,*

$$\left|\Pr[\mathsf{G}_{ab}^{\mathcal{A}} = 1] - \Pr[\mathsf{G}_0^{\mathcal{A}} = 1]\right| \leq \mathsf{negl}(\lambda).$$

Proof. The only difference in Games G_{ab} and G_0 is that there is an additional step which computes the true digest at index 0 and extracts at the 0^{th} index from c_{even} using the extraction function of SE. Finally, the adversary wins if and only if the extracted value matches the true digest along with the usual win conditions in the previous game.

Note that,

$$\begin{aligned}\left|\Pr[\mathsf{G}_{ab}^{\mathcal{A}} = 1] - \Pr[\mathsf{G}_0^{\mathcal{A}} = 1]\right| \leq \Pr[\mathsf{BARG.V}(\mathsf{BARG.crs}, C_{\mathsf{index}}, \Pi) = 1 \wedge ((x,y), T, P, H_P, \mathsf{crs}) \notin \mathcal{L}_{\mathcal{TM}} \wedge \\ \mathsf{h}_0 \neq \mathit{Step}(x, P, \mathsf{crs}, 0).\bar{\mathsf{h}}] \leq \Pr[\mathsf{BARG.V}(\mathsf{BARG.crs}, C_{\mathsf{index}}, \Pi) = 1 \wedge \mathsf{h}_0 \neq \mathit{Step}(x, P, \mathsf{crs}, 0).\bar{\mathsf{h}}].\end{aligned}$$

Let us assume that there exists a PPT adversary $\mathcal{A}$ such that for infinitely many values of $\lambda \in \mathbb{N}$,

$$\Pr[\mathsf{BARG.V}(\mathsf{BARG.crs}, C_{\mathsf{index}}, \Pi) = 1 \wedge \mathsf{h}_0 \neq \mathit{Step}(x, y, P, \mathsf{crs}, 0).\bar{\mathsf{h}}] \geq \frac{1}{\mathsf{poly}(\lambda)}.$$

Notice that $\mathsf{h}_0 \neq \mathit{Step}(x, P, \mathsf{crs}, 0).\bar{\mathsf{h}}$ implies that at least one of the conditions $\mathsf{st}_0 = \mathsf{start}$, $H_x = \mathsf{rt}_0^1$, $H_P = \mathsf{rt}_0^2$ and $\mathsf{HT.Hash}(\mathsf{dk}, \square)$ having rt_0^3 as root must not be true. If this is indeed true then our construction of C_{index} in Figure 3 implies that $(C_{\mathsf{index}}, 0) \notin \mathcal{L}^{\mathsf{index}}$.

We now construct the following PPT adversary $\mathcal{B}$ playing the semi-adaptive somewhere soundness game of the BARG as follows:

Adversary $\mathcal{B}$ playing semi adaptive somewhere soundness game of BARG.

– $\mathsf{SE}.K_{\mathsf{even}} \leftarrow \mathsf{SE.TGen}(1^\lambda, 1^M, I_{S_0})$

– $\mathsf{SE}.K_{\mathsf{odd}} \leftarrow \mathsf{SE.TGen}(1^\lambda, 1^M, I_{S_1})$

- $\mathsf{BARG.crs} \leftarrow \mathsf{BARG.TGen}(1^\lambda, 1^{T+1}, 1^{|C_{\mathsf{index}}|}, 0)$
- $\mathsf{dk} \leftarrow \mathsf{HT.Gen}(1^\lambda)$
- $\mathsf{crs} := (\mathsf{SE}.K_{\mathsf{even}}, \mathsf{SE}.K_{\mathsf{odd}}, \mathsf{BARG.crs}, \mathsf{dk})$.
- $(P, \mathsf{aux}) \leftarrow \mathcal{A}_1(1^\lambda, \mathsf{crs})$
- $H_P \leftarrow \mathsf{stSNARG.TrustHash}(\mathsf{crs}, P)$
- $((x, y), (c, \Pi)) \leftarrow \mathcal{A}_2(\mathsf{crs}, P, \mathsf{aux})$
- return $(C_{\mathsf{index}}, \Pi)$

By our assumption, it is clear that $\mathsf{BARG.V}(\mathsf{BARG.crs}, C_{\mathsf{index}}, \Pi) = 1$ with non negligible probability but $(C_{\mathsf{index}}, 0) \notin \mathcal{L}^{\mathsf{index}}$. Thus, $\mathcal{B}$ will break the semi-adaptive somewhere soundness of BARG at index 0. Therefore, it must be the case that for every PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that for all $\lambda \in \mathbb{N}$,

$$\Pr[\mathsf{BARG.V}(\mathsf{BARG.crs}, C_{\mathsf{index}}, \Pi) = 1 \wedge \mathsf{h}_0 \neq \mathit{Step}(x, y, P, \mathsf{crs}, 0).\bar{\mathsf{h}}] \leq \mathsf{negl}(\lambda)$$
$$\implies \left|\Pr[\mathsf{G}^{\mathcal{A}}_{ab} = 1] - \Pr[\mathsf{G}^{\mathcal{A}}_0 = 1]\right| \leq \mathsf{negl}(\lambda)$$

.

□

Hybrid Game $\mathsf{G}_{0,a}$

- $\mathsf{SE}.K_{\mathsf{even}} \leftarrow \mathsf{SE.TGen}(1^\lambda, 1^M, I_{S_0})$
- $\mathsf{SE}.K_{\mathsf{odd}} \leftarrow \mathsf{SE.TGen}(1^\lambda, 1^M, I_{S_1})$
- $\mathsf{BARG.crs} \leftarrow \mathsf{BARG.TGen}(1^\lambda, 1^{T+1}, 1^{|C_{\mathsf{index}}|}, 0)$
- $\mathsf{dk} \leftarrow \mathsf{HT.Gen}(1^\lambda)$
- $\mathsf{crs} := (\mathsf{SE}.K_{\mathsf{even}}, \mathsf{SE}.K_{\mathsf{odd}}, \mathsf{BARG.crs}, \mathsf{dk})$.
- $(P, \mathsf{aux}) \leftarrow \mathcal{A}_1(1^\lambda, \mathsf{crs})$
- $H_P \leftarrow \mathsf{stSNARG.TrustHash}(\mathsf{crs}, P)$
- $((x, y), (c, \Pi)) \leftarrow \mathcal{A}_2(\mathsf{crs}, P, \mathsf{aux})$
- $\mathsf{h}_0 \leftarrow \mathsf{SE.Ext}_{\mathsf{even}}(c_{\mathsf{even}}, \mathsf{SE}.K_{\mathsf{even}})$
- $(\mathsf{h}_1, \{b^k_1\}_{k\in[3]}, \{\Pi^k_1\}_{k\in[3]}, \Pi'_i) \leftarrow \mathsf{SE.Ext}_{\mathsf{odd}}(c_{\mathsf{odd}}, \mathsf{SE}.K_{\mathsf{odd}})$
- if $\mathsf{stSNARG.V}(\mathsf{crs}, (x, y), H_P, (c, \Pi)) = 1 \wedge ((x, y), T, P, H_P, \mathsf{crs}) \notin \mathcal{L}_{\mathcal{TM}} \wedge \mathsf{h}_0 = \mathit{Step}(x, y, P, \mathsf{crs}, 0).\bar{\mathsf{h}}$, return 1
- else return 0

Lemma 5.6.

$$\Pr[\mathsf{G}^{\mathcal{A}}_0 = 1] = \Pr[\mathsf{G}^{\mathcal{A}}_{0a} = 1].$$

This lemma follows from a straightforward observation that both the games are indeed identical except an additional extraction of c at index 1 which is not being used anywhere in the game.

Hybrid Game $\mathsf{G}_{0,b}$

- $\mathsf{SE}.K_{\mathsf{even}} \leftarrow \mathsf{SE.TGen}(1^\lambda, 1^M, I_{S_0})$
- $\mathsf{SE}.K_{\mathsf{odd}} \leftarrow \mathsf{SE.TGen}(1^\lambda, 1^M, I_{S_1})$
- $\mathsf{BARG.crs} \leftarrow \mathsf{BARG.TGen}(1^\lambda, 1^{T+1}, 1^{|C_{\mathsf{index}}|}, 1)$
- $\mathsf{dk} \leftarrow \mathsf{HT.Gen}(1^\lambda)$
- $\mathsf{crs} := (\mathsf{SE}.K_{\mathsf{even}}, \mathsf{SE}.K_{\mathsf{odd}}, \mathsf{BARG.crs}, \mathsf{dk})$.
- $(P, \mathsf{aux}) \leftarrow \mathcal{A}_1(1^\lambda, \mathsf{crs})$

- $H_P \leftarrow \mathsf{stSNARG.TrustHash}(\mathsf{crs}, P)$
- $((x,y),(c,\Pi)) \leftarrow \mathcal{A}_2(\mathsf{crs}, P, \mathsf{aux})$
- $\mathsf{h}_0 \leftarrow \mathsf{SE.Ext}_{\mathsf{even}}(c_{\mathsf{even}}, \mathsf{SE}.K_{\mathsf{even}})$
- $(\mathsf{h}_1, \{b_1^k\}_{k\in[3]}, \{\Pi_1^k\}_{k\in[3]}, \Pi_i') \leftarrow \mathsf{SE.Ext}_{\mathsf{odd}}(c_{\mathsf{odd}}, \mathsf{SE}.K_{\mathsf{odd}})$
- if $\mathsf{stSNARG.V}\,(\mathsf{crs},(x,y),H_P,(c,\Pi)) = 1 \wedge ((x,y),T,P,H_P,\mathsf{crs}) \notin \mathcal{L}_{\mathcal{TM}} \wedge \mathsf{h}_0 = \mathit{Step}(x,y,P,\mathsf{crs},0).\bar{\mathsf{h}}$, return 1
- else return 0

Lemma 5.7. *Assuming key indistinguishability of* BARG, $\left|\Pr[\mathsf{G}_{0,b}^{\mathcal{A}} = 1] - \Pr[\mathsf{G}_{0,a}^{\mathcal{A}} = 1]\right| \leq \mathsf{negl}(\lambda)$.

The only difference in the above transition is that the BARG key is generated with a trapdoor at index 1 rather than 0. Hence, the lemma follows from the key-indistinguishability of BARG as shown in Lemma 5.4.

Hybrid Game $\mathsf{G}_{0,c}$

- $\mathsf{SE}.K_{\mathsf{even}} \leftarrow \mathsf{SE.TGen}(1^\lambda, 1^M, I_{S_0})$
- $\mathsf{SE}.K_{\mathsf{odd}} \leftarrow \mathsf{SE.TGen}(1^\lambda, 1^M, I_{S_1})$
- $\mathsf{BARG.crs} \leftarrow \mathsf{BARG.TGen}(1^\lambda, 1^{T+1}, 1^{|C_{\mathsf{index}}|}, 1)$
- $\mathsf{dk} \leftarrow \mathsf{HT.Gen}(1^\lambda)$
- $\mathsf{crs} := (\mathsf{SE}.K_{\mathsf{even}}, \mathsf{SE}.K_{\mathsf{odd}}, \mathsf{BARG.crs}, \mathsf{dk})$.
- $(P, \mathsf{aux}) \leftarrow \mathcal{A}_1(1^\lambda, \mathsf{crs})$
- $H_P \leftarrow \mathsf{stSNARG.TrustHash}(\mathsf{crs}, P)$
- $((x,y),(c,\Pi)) \leftarrow \mathcal{A}_2(\mathsf{crs}, P, \mathsf{aux})$
- $\mathsf{h}_0 \leftarrow \mathsf{SE.Ext}_{\mathsf{even}}(c_{\mathsf{even}}, \mathsf{SE}.K_{\mathsf{even}})$
- $(\mathsf{h}_1, \{b_1^k\}_{k\in[3]}, \{\Pi_1^k\}_{k\in[3]}, \Pi_i') \leftarrow \mathsf{SE.Ext}_{\mathsf{odd}}(c_{\mathsf{odd}}, \mathsf{SE}.K_{\mathsf{odd}})$
- if $\mathsf{stSNARG.V}\,(\mathsf{crs},(x,y),H_P,(c,\Pi)) = 1 \wedge ((x,y),T,P,H_P,\mathsf{crs}) \notin \mathcal{L}_{\mathcal{TM}} \wedge \mathsf{h}_0 = \mathit{Step}(x,y,P,\mathsf{crs},0).\bar{\mathsf{h}} \wedge \{b_1^k\}_{k\in[3]} = \mathit{Step}(x,y,P,\mathsf{crs},1).\bar{b} \wedge \{\mathsf{rt}_1^k\}_{k\in[3]} = \mathit{Step}(x,y,P,\mathsf{crs},1).\bar{\mathsf{rt}} \wedge \mathsf{st}_1 = \mathit{Step}(x,y,P,\mathsf{crs},1).\bar{\mathsf{st}}$, return 1
- else return 0

Lemma 5.8. *Assuming semi-adaptive somewhere soundness of* BARG*, extraction correctness of* SE*, read and write soundness of* HT*,*

$$\left|\Pr[\mathsf{G}_{0,b}^{\mathcal{A}} = 1] - \Pr[\mathsf{G}_{0,c}^{\mathcal{A}} = 1]\right| \leq \mathsf{negl}(\lambda).$$

Proof. The only difference in Games $\mathsf{G}_{0,b}$ and $\mathsf{G}_{0,c}$ is that we have added some additional conditions for the adversary to win along with the ones in the previous game.

Note that,

$$\left|\Pr[\mathsf{G}_{0,b}^{\mathcal{A}} = 1] - \Pr[\mathsf{G}_{0,c}^{\mathcal{A}} = 1]\right| \leq \Pr[\mathsf{BARG.V}(\mathsf{BARG.crs}, C_{\mathsf{index}}, \Pi) = 1 \wedge \mathsf{h}_0 = \mathit{Step}(x,P,\mathsf{crs},0).\bar{\mathsf{h}} \wedge \left(\{b_1^k\}_{k\in[3]} \neq \mathit{Step}(x,y,P,\mathsf{crs},1).\bar{b} \vee \{\mathsf{rt}_1^k\}_{k\in[3]} \neq \mathit{Step}(x,y,P,\mathsf{crs},1).\bar{\mathsf{rt}} \vee \mathsf{st}_1 = \mathit{Step}(x,y,P,\mathsf{crs},1).\bar{\mathsf{st}}\right)].$$

Let us assume that there exists a PPT adversary $\mathcal{A}$ such that for infinitely many values of $\lambda \in \mathbb{N}$,

$$\Pr[\mathsf{BARG.V}(\mathsf{BARG.crs}, C_{\mathsf{index}}, \Pi) = 1 \wedge \mathsf{h}_0 = \mathit{Step}(x,y,P,\mathsf{crs},0).\bar{\mathsf{h}} \wedge \left(\{b_1^k\}_{k\in[3]} \neq \mathit{Step}(x,y,P,\mathsf{crs},1).\bar{b} \vee \{\mathsf{rt}_1^k\}_{k\in[3]} \neq \mathit{Step}(x,y,P,\mathsf{crs},1).\bar{\mathsf{rt}} \vee \mathsf{st}_1 = \mathit{Step}(x,y,P,\mathsf{crs},1).\bar{\mathsf{st}}\right)] \geq \frac{1}{\mathsf{poly}(\lambda)}.$$

Notice that $\mathsf{h}_0 = \mathit{Step}(x,P,\mathsf{crs},0).\bar{\mathsf{h}}$ implies that the conditions $\mathsf{st}_0 = \mathsf{start}$, $H_x = \mathsf{rt}_0^1$, $H_P = \mathsf{rt}_0^2$ and $\mathsf{HT.Hash}(\mathsf{dk}, \square)$ having rt_0^3 as root are true. In other words, h_0 is indeed the true digest at step 0.

Assuming extraction correctness of SE, read and write soundness of HT, we construct the following PPT adversary $\mathcal{B}$ playing the semi-adaptive somewhere soundness game of the BARG as follows:

Adversary $\mathcal{B}$ playing semi adaptive somewhere soundness game of BARG.

- $\mathsf{SE}.K_{\mathsf{even}} \leftarrow \mathsf{SE.TGen}(1^\lambda, 1^M, I_{S_0})$
- $\mathsf{SE}.K_{\mathsf{odd}} \leftarrow \mathsf{SE.TGen}(1^\lambda, 1^M, I_{S_1})$
- $\mathsf{BARG.crs} \leftarrow \mathsf{BARG.TGen}(1^\lambda, 1^{T+1}, 1^{|C_{\mathsf{index}}|}, 1)$
- $\mathsf{dk} \leftarrow \mathsf{HT.Gen}(1^\lambda)$
- $\mathsf{crs} := (\mathsf{SE}.K_{\mathsf{even}}, \mathsf{SE}.K_{\mathsf{odd}}, \mathsf{BARG.crs}, \mathsf{dk})$.
- $(P, \mathsf{aux}) \leftarrow \mathcal{A}_1(1^\lambda, \mathsf{crs})$
- $H_P \leftarrow \mathsf{stSNARG.TrustHash}(\mathsf{crs}, P)$
- $((x, y), (c, \Pi)) \leftarrow \mathcal{A}_2(\mathsf{crs}, P, \mathsf{aux})$
- return $(C_{\mathsf{index}}, \Pi)$

By our assumption, it is clear that $\mathsf{BARG.V}(\mathsf{BARG.crs}, C_{\mathsf{index}}, \Pi) = 1$ with non negligible probability. Thus, $\mathcal{B}$ will break the semi-adaptive somewhere soundness of BARG at index 1 if $(C_{\mathsf{index}}, 1) \notin \mathcal{L}^{\mathsf{index}}$. Thus, if $(C_{\mathsf{index}}, 1) \notin \mathcal{L}^{\mathsf{index}}$ it must be the case that for every PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that for all $\lambda \in \mathbb{N}$,

$$\Pr[\mathsf{BARG.V}(\mathsf{BARG.crs}, C_{\mathsf{index}}, \Pi) = 1 \wedge \mathsf{h}_0 = \mathit{Step}(x, y, P, \mathsf{crs}, 0).\bar{\mathsf{h}} \wedge \big(\{b_1^k\}_{k\in[3]} \neq \mathit{Step}(x, y, P, \mathsf{crs}, 1).\bar{b} \vee \{\mathsf{rt}_1^k\}_{k\in[3]} \neq \mathit{Step}(x, y, P, \mathsf{crs}, 1).\bar{\mathsf{rt}} \vee \mathsf{st}_1 \neq \mathit{Step}(x, y, P, \mathsf{crs}, 1).\bar{\mathsf{st}}\big)] \leq \mathsf{negl}(\lambda)$$
$$\implies \big|\Pr[\mathsf{G}_{0,b}^{\mathcal{A}} = 1] - \Pr[\mathsf{G}_{0,c}^{\mathcal{A}} = 1]\big| \leq \mathsf{negl}(\lambda).$$

It is now left to show that $(C_{\mathsf{index}}, 1) \notin \mathcal{L}^{\mathsf{index}}$.

Case 1 If the SE verifications in C_{index} do not all return 1, then by construction of C_{index}, we have that $(C_{\mathsf{index}}, 1) \notin \mathcal{L}^{\mathsf{index}}$.

Case 2 All SE verifications return 1. Extraction Correctness/ Somewhere binding property of SE hash implies that $\mathsf{h}_0 = (\mathsf{st}_0, \mathsf{rt}_0^1, \mathsf{rt}_0^2, \mathsf{rt}_0^3), \mathsf{h}_1, \{b_1^k, \Pi_1^k\}_{k\in[3]}, \Pi_1'$ were indeed committed by the prover as the Turing machine output at step 0 and step 1. Now, let us analyze $\phi(\mathsf{h}_0, \mathsf{h}_1, \{b_1^k, \Pi_1^k\}_{k\in[3]}, \Pi_1')$. By assumption, we know that $\mathsf{h}_0 = \bar{\mathsf{h}}_0$, i.e., $\bar{\mathsf{st}}_0, \bar{\mathsf{rt}}_0^1, \bar{\mathsf{rt}}_0^2, \bar{\mathsf{rt}}_0^3 = \mathsf{st}_0, \mathsf{rt}_0^1, \mathsf{rt}_0^2, \mathsf{rt}_0^3$. StepR being a deterministic function ensures that (l_1^1, l_1^2, l_1^3) are indeed the correct Turing machine memory locations to be read at step 1. Thus $(\bar{l}_1^1, \bar{l}_1^2, \bar{l}_1^3) = (l_1^1, l_1^2, l_1^3)$. This along with the deterministic nature of hash tree read write operations means that we must have,

- $(\bar{l}_1^1, \bar{l}_1^2, \bar{l}_1^3) \leftarrow \mathsf{StepR}(\bar{\mathsf{st}}_0)$
- $\Big\{(\bar{b}_1^j, \bar{\Pi}_1^k) \coloneqq \mathsf{HT.Read}(\bar{\mathsf{tree}}_0^k, \bar{l}_1^k)\Big\}_{k\in[3]}$
- $(\bar{b}_1'^3, \bar{l}_1'^3, \bar{\mathsf{st}}_1) \coloneqq \mathsf{StepW}(\bar{\mathsf{st}}_0, \bar{b}_1^1, \bar{b}_1^2, \bar{b}_1^3)$
- $(\bar{\mathsf{tree}}_1^3, \bar{\mathsf{rt}}_1^3, \bar{\Pi}_1') \coloneqq \mathsf{HT.Write}(\bar{\mathsf{tree}}_0^3, \bar{l}_1'^3, \bar{b}_1'^3)$

Read and Write Completeness of the hash tree implies

$$\mathsf{HT.VerRead}(\mathsf{dk}_1, \bar{\mathsf{rt}}_0^1, \bar{l}_1^1, \bar{b}_1^1, \bar{\Pi}_1^1) = 1 \qquad \mathsf{HT.VerRead}(\mathsf{dk}_3, \bar{\mathsf{rt}}_0^3, \bar{l}_1'^3, \bar{b}_1'^3, \bar{\Pi}_1'^3) = 1$$
$$\mathsf{HT.VerRead}(\mathsf{dk}_2, \bar{\mathsf{rt}}_0^2, \bar{l}_1'^2, \bar{b}_1'^2, \bar{\Pi}_1'^2) = 1 \qquad \mathsf{HT.VerWrite}(\mathsf{dk}_3, \bar{\mathsf{rt}}_0^3, \bar{l}_1'^3, \bar{b}_1'^3, \bar{\mathsf{rt}}_1^3 \bar{\Pi}_1') = 1$$

If $\{b_1^k\}_{k\in[3]} \neq \mathit{Step}(x, y, P, \mathsf{crs}, 1).\bar{b}$, then the read soundness assumption of HT implies that $\big(\mathsf{HT.VerRead}(\mathsf{dk}, \bar{\mathsf{rt}}_0^1, \bar{l}_1^k, b_1^k, \Pi_1^k) = 1\big)_{k\in[3]}$ happens with a negligible probability. Thus, with all but negligible probability we have that $(C_{\mathsf{index}}, 1) \notin \mathcal{L}^{\mathsf{index}}$ and we are done.

Let us say this is not the case, i.e., $\{b_1^k\}_{k\in[3]} = \mathit{Step}(x, y, P, \mathsf{crs}, 1).\bar{b}$, then the deterministic nature of the Turing machine write function StepW implies that $\mathsf{st}_1 = \bar{\mathsf{st}}_1$. Thus, for our assumption to be valid, it must be that $\{\mathsf{rt}_1^k\}_{k\in[3]} \neq \mathit{Step}(x, y, P, \mathsf{crs}, 1).\bar{\mathsf{rt}}$. If $\mathsf{rt}_1^1 \neq \bar{\mathsf{rt}}_1^1 = \mathsf{rt}_0^1$ or $\mathsf{rt}_1^2 \neq \bar{\mathsf{rt}}_1^2 = \mathsf{rt}_0^2$, then

the definition of ϕ implies that $(C_{\mathsf{index}}, 1) \notin \mathcal{L}^{\mathsf{index}}$. If this is not the case, then the only other possible option is $\mathsf{rt}_1^3 \neq \bar{\mathsf{rt}}_1^3$. Now, the write soundness of HT implies that with all but negligible probability, $\mathsf{HT.VerWrite}(\mathsf{dk}_3, \bar{\mathsf{rt}}_0^3, \bar{l}_1'^3, \bar{b}_1'^3, \mathsf{rt}_1^3, \Pi_1) \neq 1$ must hold. If this is indeed true then our construction of C_{index} in Figure 3 implies that $(C_{\mathsf{index}}, 1) \notin \mathcal{L}^{\mathsf{index}}$.

□

Hybrid Game $\mathsf{G}_{0,d}$

- $\mathsf{SE}.K_{\mathsf{even}} \leftarrow \mathsf{SE.TGen}(1^\lambda, 1^M, I_{S_0})$
- $\mathsf{SE}.K_{\mathsf{odd}} \leftarrow \mathsf{SE.TGen}(1^\lambda, 1^M, I_{S_1})$
- $\mathsf{BARG.crs} \leftarrow \mathsf{BARG.TGen}(1^\lambda, 1^{T+1}, 1^{|C_{\mathsf{index}}|}, 1)$
- $\mathsf{dk} \leftarrow \mathsf{HT.Gen}(1^\lambda)$
- $\mathsf{crs} := (\mathsf{SE}.K_{\mathsf{even}}, \mathsf{SE}.K_{\mathsf{odd}}, \mathsf{BARG.crs}, \mathsf{dk})$.
- $(P, \mathsf{aux}) \leftarrow \mathcal{A}_1(1^\lambda, \mathsf{crs})$
- $H_P \leftarrow \mathsf{stSNARG.TrustHash}(\mathsf{crs}, P)$
- $((x, y), (c, \Pi)) \leftarrow \mathcal{A}_2(\mathsf{crs}, P, \mathsf{aux})$
- $\mathsf{h}_0 \leftarrow \mathsf{SE.Ext}_{\mathsf{even}}(c_{\mathsf{even}}, \mathsf{SE}.K_{\mathsf{even}})$
- $(\mathsf{h}_1, \{b_1^k\}_{k\in[3]}, \{\Pi_1^k\}_{k\in[3]}, \Pi_i') \leftarrow \mathsf{SE.Ext}_{\mathsf{odd}}(c_{\mathsf{odd}}, \mathsf{SE}.K_{\mathsf{odd}})$
- if $\mathsf{stSNARG.V}(\mathsf{crs}, (x, y), H_P, (c, \Pi)) = 1 \wedge ((x, y), T, P, H_P, \mathsf{crs}) \notin \mathcal{L}_{\mathcal{TM}} \wedge \mathsf{h}_0 = \mathit{Step}(x, y, P, \mathsf{crs}, 0).\bar{\mathsf{h}} \wedge \{b_1^k\}_{k\in[3]} = \mathit{Step}(x, y, P, \mathsf{crs}, 1).\bar{b} \wedge \mathsf{rt}_1^3 = \mathit{Step}(x, y, P, \mathsf{crs}, 1).\bar{\mathsf{rt}} \wedge \mathsf{st}_1 = \mathit{Step}(x, y, P, \mathsf{crs}, 1).\bar{\mathsf{st}} \wedge \mathsf{h}_1 = \mathit{Step}(x, y, P, \mathsf{crs}, 1).\bar{\mathsf{h}}$, return 1
- else return 0

Lemma 5.9.

$$\Pr[\mathsf{G}_{0,c}^{\mathcal{A}} = 1] = \Pr[\mathsf{G}_{0,d}^{\mathcal{A}} = 1].$$

Proof. Note that by definition, $\mathsf{h}_1 = \mathsf{st}_1, \mathsf{rt}_1^1, \mathsf{rt}_1^2, \mathsf{rt}_1^3$. We already have that $\mathsf{rt}_1^1, \mathsf{rt}_1^2, \mathsf{rt}_1^3 = \mathit{Step}(x, y, P, \mathsf{crs}, 1).\bar{\mathsf{rt}}$ and $\mathsf{st}_1 = \mathit{Step}(x, y, P, \mathsf{crs}, 1).\bar{\mathsf{st}}$. Thus $\mathsf{h}_1 = \mathit{Step}(x, y, P, \mathsf{crs}, 1).\bar{\mathsf{h}}$ if and only if $\mathsf{rt}_1^3 = \mathit{Step}(x, y, P, \mathsf{crs}, 1).\bar{\mathsf{rt}} \wedge \mathsf{st}_1 = \mathit{Step}(x, y, P, \mathsf{crs}, 1).\bar{\mathsf{st}}$. □

Hybrid Game $\mathsf{G}_{0,e}$

- $\mathsf{SE}.K_{\mathsf{even}} \leftarrow \mathsf{SE.TGen}(1^\lambda, 1^M, I_{S_0})$
- $\mathsf{SE}.K_{\mathsf{odd}} \leftarrow \mathsf{SE.TGen}(1^\lambda, 1^M, I_{S_1})$
- $\mathsf{BARG.crs} \leftarrow \mathsf{BARG.TGen}(1^\lambda, 1^{T+1}, 1^{|C_{\mathsf{index}}|}, 1)$
- $\mathsf{dk} \leftarrow \mathsf{HT.Gen}(1^\lambda)$
- $\mathsf{crs} := (\mathsf{SE}.K_{\mathsf{even}}, \mathsf{SE}.K_{\mathsf{odd}}, \mathsf{BARG.crs}, \mathsf{dk})$.
- $(P, \mathsf{aux}) \leftarrow \mathcal{A}_1(1^\lambda, \mathsf{crs})$
- $H_P \leftarrow \mathsf{stSNARG.TrustHash}(\mathsf{crs}, P)$
- $((x, y), (c, \Pi)) \leftarrow \mathcal{A}_2(\mathsf{crs}, P, \mathsf{aux})$
- $\mathsf{h}_0 \leftarrow \mathsf{SE.Ext}_{\mathsf{even}}(c_{\mathsf{even}}, \mathsf{SE}.K_{\mathsf{even}})$
- $(\mathsf{h}_1, \{b_1^k\}_{k\in[3]}, \{\Pi_1^k\}_{k\in[3]}, \Pi_i') \leftarrow \mathsf{SE.Ext}_{\mathsf{odd}}(c_{\mathsf{odd}}, \mathsf{SE}.K_{\mathsf{odd}})$
- if $\mathsf{stSNARG.V}(\mathsf{crs}, (x, y), H_P, (c, \Pi)) = 1 \wedge ((x, y), T, P, H_P, \mathsf{crs}) \notin \mathcal{L}_{\mathcal{TM}} \wedge \mathsf{h}_1 = \mathit{Step}(x, y, P, \mathsf{crs}, 1).\bar{\mathsf{h}}$, return 1
- else return 0

Lemma 5.10.

$$\Pr[\mathsf{G}_{0,d}^{\mathcal{A}} = 1] \leq \Pr[\mathsf{G}_{0,e}^{\mathcal{A}} = 1].$$

Proof. The number of conditions for the adversary to win simply decreases from Game $\mathsf{G}_{0,d}$ to Game $\mathsf{G}_{0,e}$, thus the probability of success must not increase. □

A closer observation shows that $\mathsf{G}_{0,e}$ is indeed identical to the case when one puts $i = 1$ in game G_i. Combining these together, we show the base case of the induction to be true. Thus,

$$\Pr[\mathsf{G}^{\mathcal{A}} = 1] \leq \Pr[\mathsf{G}_1^{\mathcal{A}} = 1] + \mathsf{negl}(\lambda).$$

Assuming that our induction hypothesis holds for some $j \in [T-1]$, we prove that it holds for $j+1$ as well. We note that by chain rule, it suffices to show that $\Pr[\mathsf{G}_j^{\mathcal{A}} = 1] \leq \Pr[\mathsf{G}_{j+1}^{\mathcal{A}} = 1] + \mathsf{negl}(\lambda)$. We show this by a sequence of inner hybrids to transition from Game G_j to G_{j+1}.

Inner Hybrid Game $\mathsf{G}_{j,a}$

- if j is even
  $\mathsf{SE}.K_{\mathsf{even}} \leftarrow \mathsf{SE.TGen}(1^\lambda, 1^M, I_{S_j})$
  $\mathsf{SE}.K_{\mathsf{odd}} \leftarrow \mathsf{SE.TGen}(1^\lambda, 1^M, I_{S_{j+1}})$
- if j is odd
  $\mathsf{SE}.K_{\mathsf{even}} \leftarrow \mathsf{SE.TGen}(1^\lambda, 1^M, I_{S_{j+1}})$
  $\mathsf{SE}.K_{\mathsf{odd}} \leftarrow \mathsf{SE.TGen}(1^\lambda, 1^M, I_{S_j})$
- $\mathsf{BARG.crs} \leftarrow \mathsf{BARG.TGen}(1^\lambda, 1^{T+1}, 1^{|C_{\mathsf{index}}|}, j)$
- $\mathsf{dk} \leftarrow \mathsf{HT.Gen}(1^\lambda)$
- $\mathsf{crs} := (\mathsf{SE}.K_{\mathsf{even}}, \mathsf{SE}.K_{\mathsf{odd}}, \mathsf{BARG.crs}, \mathsf{dk})$.
- $(P, \mathsf{aux}) \leftarrow \mathcal{A}_1(1^\lambda, \mathsf{crs})$
- $H_P \leftarrow \mathsf{stSNARG.TrustHash}(\mathsf{crs}, P)$
- $((x, y)(c, \Pi)) \leftarrow \mathcal{A}_2(\mathsf{crs}, P, \mathsf{aux})$
- if j is even
  $(\mathsf{h}_j, \{b_j^k\}_{k\in[3]}, \{\Pi_j^k\}_{k\in[3]}, \Pi'_j) \leftarrow \mathsf{Ext}_{\mathsf{even}}(c_{\mathsf{even}}, \mathsf{SE}.K_{\mathsf{even}})$
  $(\mathsf{h}_{j-1}, \{b_{j-1}^k\}_{k\in[3]}, \{\Pi_{j-1}^k\}_{k\in[3]}, \Pi'_{j-1}) \leftarrow \mathsf{Ext}_{\mathsf{odd}}(c_{\mathsf{odd}}, \mathsf{SE}.K_{\mathsf{odd}})$
- if j is odd
  $(\mathsf{h}_j, \{b_j^k\}_{k\in[3]}, \{\Pi_j^k\}_{k\in[3]}, \Pi'_j) \leftarrow \mathsf{Ext}_{\mathsf{odd}}(c_{\mathsf{odd}}, \mathsf{SE}.K_{\mathsf{odd}})$
  $(\mathsf{h}_{j-1}, \{b_{j-1}^k\}_{k\in[3]}, \{\Pi_{j-1}^k\}_{k\in[3]}, \Pi'_{j-1}) \leftarrow \mathsf{Ext}_{\mathsf{even}}(c_{\mathsf{even}}, \mathsf{SE}.K_{\mathsf{even}})$
- if $\mathsf{stSNARG.V}(\mathsf{crs}, (x, y), H_P, (c, \Pi)) = 1 \wedge ((x, y), T, P, H_P, \mathsf{crs}) \notin \mathcal{L}_{\mathcal{TM}} \wedge \mathsf{h}_j = \mathit{Step}(x, y, P, \mathsf{crs}, j).\bar{\mathsf{h}}_j$, return 1
- else return 0

The only difference between Games G_j and $\mathsf{G}_{j,a}$ is that the SE hash is binding at both indices j and $j+1$ instead of j and $j-1$.

Lemma 5.11. *Assuming key indistinguishability of* SE, $\left|\Pr[\mathsf{G}_j^{\mathcal{A}} = 1] - \Pr[\mathsf{G}_{j,a}^{\mathcal{A}} = 1]\right| \leq \mathsf{negl}(\lambda)$.

Proof. If $\left|\Pr[\mathsf{G}_j^{\mathcal{A}} = 1] - \Pr[\mathsf{G}_{j,1}^{\mathcal{A}} = 1]\right| > \mathsf{negl}(\lambda)$, then one can construct a PPT adversary $\mathcal{B}$ that breaks the key indistinguishability of SE with Key as input from the key generation algorithm of SE hash as follows:

Adversary $\mathcal{B}$ playing SE key indistinguishability game.

- if j is even
  $\mathsf{SE}.K_{\mathsf{even}} \leftarrow \mathsf{SE.TGen}(1^\lambda, 1^M, I_{S_j})$
  $\mathsf{SE}.K_{\mathsf{odd}} \leftarrow \mathsf{Key}$
- if j is odd

$\mathsf{SE}.K_{\mathsf{even}} \leftarrow \mathsf{Key}$
$\mathsf{SE}.K_{\mathsf{odd}} \leftarrow \mathsf{SE.TGen}(1^\lambda, 1^M, I_{S_j})$

- $\mathsf{BARG.crs} \leftarrow \mathsf{BARG.TGen}(1^\lambda, 1^{T+1}, 1^{|C_{\mathsf{index}}|}, j)$
- $\mathsf{dk} \leftarrow \mathsf{HT.Gen}(1^\lambda)$
- $\mathsf{crs} := (\mathsf{SE}.K_{\mathsf{even}}, \mathsf{SE}.K_{\mathsf{odd}}, \mathsf{BARG.crs}, \mathsf{dk})$.
- $(P, \mathsf{aux}) \leftarrow \mathcal{A}_1(1^\lambda, \mathsf{crs})$
- $H_P \leftarrow \mathsf{stSNARG.TrustHash}(\mathsf{crs}, P)$
- $((x, y), (c, \Pi)) \leftarrow \mathcal{A}_2(\mathsf{crs}, P, \mathsf{aux})$
- if j is even
  $(\mathsf{h}_j, \{b_j^k\}_{k\in[3]}, \{\Pi_j^k\}_{k\in[3]}, \Pi'_j) \leftarrow \mathsf{Ext}_{\mathsf{even}}(c_{\mathsf{even}}, \mathsf{SE}.K_{\mathsf{even}})$
- if j is odd
  $(\mathsf{h}_j, \{b_j^k\}_{k\in[3]}, \{\Pi_j^k\}_{k\in[3]}, \Pi'_j) \leftarrow \mathsf{Ext}_{\mathsf{odd}}(c_{\mathsf{odd}}, \mathsf{SE}.K_{\mathsf{odd}})$
- if $\mathsf{stSNARG.V}\,(\mathsf{crs}, (x, y), H_P, (c, \Pi)) = 1 \wedge ((x, y), T, P, H_P, \mathsf{crs}) \notin \mathcal{L}_{\mathcal{TM}} \wedge \mathsf{h}_j = \mathit{Step}(x, y, P, \mathsf{crs}, j).\bar{\mathsf{h}}_j$, return 1
- else return 0

Here, Key is either $\mathsf{SE.TGen}(1^\lambda, 1^M, I_{S_{j-1}})$ or $\mathsf{SE.TGen}(1^\lambda, 1^M, I_{S_{j+1}})$ based on whether $\mathcal{A}$ is interacting with game G_j or $\mathsf{G}_{j,a}$ respectively. Adversary $\mathcal{B}$ breaks the key indistinguishability of SE hash if the probability that it returns 1 is significantly different when $\mathcal{A}$ interacts with the key generation algorithm in normal mode vs trapdoor mode. Now, the probability that $\mathcal{B}$ returns 1 in either case is exactly equal to the probability that $\mathcal{A}$ wins the corresponding games, hence, $\mathcal{B}$ breaks if $\left|\Pr[\mathsf{G}^{\mathcal{A}} = 1] - \Pr[\mathsf{G}_a^{\mathcal{A}} = 1]\right| \geq \mathsf{negl}(\lambda)$. This leads to a contradiction of our assumption. □

Inner Hybrid Game $\mathsf{G}_{j,b}$

- if j is even
  $\mathsf{SE}.K_{\mathsf{even}} \leftarrow \mathsf{SE.TGen}(1^\lambda, 1^M, I_{S_j})$
  $\mathsf{SE}.K_{\mathsf{odd}} \leftarrow \mathsf{SE.TGen}(1^\lambda, 1^M, I_{S_{j+1}})$
- if j is odd
  $\mathsf{SE}.K_{\mathsf{even}} \leftarrow \mathsf{SE.TGen}(1^\lambda, 1^M, I_{S_{j+1}})$
  $\mathsf{SE}.K_{\mathsf{odd}} \leftarrow \mathsf{SE.TGen}(1^\lambda, 1^M, I_{S_j})$
- $\mathsf{BARG.crs} \leftarrow \mathsf{BARG.TGen}(1^\lambda, 1^{T+1}, 1^{|C_{\mathsf{index}}|}, j)$
- $\mathsf{dk} \leftarrow \mathsf{HT.Gen}(1^\lambda)$
- $\mathsf{crs} := (\mathsf{SE}.K_{\mathsf{even}}, \mathsf{SE}.K_{\mathsf{odd}}, \mathsf{BARG.crs}, \mathsf{dk})$.
- $(P, \mathsf{aux}) \leftarrow \mathcal{A}_1(1^\lambda, \mathsf{crs})$
- $H_P \leftarrow \mathsf{stSNARG.TrustHash}(\mathsf{crs}, P)$
- $((x, y)(c, \Pi)) \leftarrow \mathcal{A}_2(\mathsf{crs}, P, \mathsf{aux})$
- if j is even
  $(\mathsf{h}_j, \{b_j^k\}_{k\in[3]}, \{\Pi_j^k\}_{k\in[3]}, \Pi'_j) \leftarrow \mathsf{Ext}_{\mathsf{even}}(c_{\mathsf{even}}, \mathsf{SE}.K_{\mathsf{even}})$
  $(\mathsf{h}_{j+1}, \{b_{j+1}^k\}_{k\in[3]}, \{\Pi_{j+1}^k\}_{k\in[3]}, \Pi'_{j+1}) \leftarrow \mathsf{Ext}_{\mathsf{odd}}(c_{\mathsf{odd}}, \mathsf{SE}.K_{\mathsf{odd}})$
- if j is odd
  $(\mathsf{h}_j, \{b_j^k\}_{k\in[3]}, \{\Pi_j^k\}_{k\in[3]}, \Pi'_j) \leftarrow \mathsf{Ext}_{\mathsf{odd}}(c_{\mathsf{odd}}, \mathsf{SE}.K_{\mathsf{odd}})$
  $(\mathsf{h}_{j+1}, \{b_{j+1}^k\}_{k\in[3]}, \{\Pi_{j+1}^k\}_{k\in[3]}, \Pi'_{j+1}) \leftarrow \mathsf{Ext}_{\mathsf{even}}(c_{\mathsf{even}}, \mathsf{SE}.K_{\mathsf{even}})$
- if $\mathsf{stSNARG.V}\,(\mathsf{crs}, (x, y), H_P, (c, \Pi)) = 1 \wedge ((x, y), T, P, H_P, \mathsf{crs}) \notin \mathcal{L}_{\mathcal{TM}} \wedge \mathsf{h}_j = \mathit{Step}(x, y, P, \mathsf{crs}, j).\bar{\mathsf{h}}_j$, return 1

– else return 0

Lemma 5.12.

$$\Pr[\mathsf{G}^{\mathcal{A}}_{j,a} = 1] = \Pr[\mathsf{G}^{\mathcal{A}}_{j,b} = 1]$$

Proof. The only difference between Games $\mathsf{G}_{j,a}$ and $\mathsf{G}_{j,b}$ is that extraction for one of the SE hashes changes from $I_{S_{j-1}}$ to $I_{S_{j+1}}$. However, this does not affect the reduction in any way as extraction at indices $j-1$ and $j+1$ are not used by the reduction at any stage. □

Inner Hybrid Game $\mathsf{G}_{j,c}$

– if j is even
  $\mathsf{SE}.K_{\mathsf{even}} \leftarrow \mathsf{SE.TGen}(1^\lambda, 1^M, I_{S_j})$
  $\mathsf{SE}.K_{\mathsf{odd}} \leftarrow \mathsf{SE.TGen}(1^\lambda, 1^M, I_{S_{j+1}})$
– if j is odd
  $\mathsf{SE}.K_{\mathsf{even}} \leftarrow \mathsf{SE.TGen}(1^\lambda, 1^M, I_{S_{j+1}})$
  $\mathsf{SE}.K_{\mathsf{odd}} \leftarrow \mathsf{SE.TGen}(1^\lambda, 1^M, I_{S_j})$
– $\mathsf{BARG.crs} \leftarrow \mathsf{BARG.TGen}(1^\lambda, 1^{T+1}, 1^{|C_{\mathsf{index}}|}, j+1)$
– $\mathsf{dk} \leftarrow \mathsf{HT.Gen}(1^\lambda)$
– $\mathsf{crs} := (\mathsf{SE}.K_{\mathsf{even}}, \mathsf{SE}.K_{\mathsf{odd}}, \mathsf{BARG.crs}, \mathsf{dk})$.
– $(P, \mathsf{aux}) \leftarrow \mathcal{A}_1(1^\lambda, \mathsf{crs})$
– $H_P \leftarrow \mathsf{stSNARG.TrustHash}(\mathsf{crs}, P)$
– $((x,y),(c,\Pi)) \leftarrow \mathcal{A}_2(\mathsf{crs}, P, \mathsf{aux})$
– if j is even
  $(\mathsf{h}_j, \{b_j^k\}_{k\in[3]}, \{\Pi_j^k\}_{k\in[3]}, \Pi'_j) \leftarrow \mathsf{Ext}_{\mathsf{even}}(c, \mathsf{SE}.K_{\mathsf{even}})$
  $(\mathsf{h}_{j+1}, \{b_{j+1}^k\}_{k\in[3]}, \{\Pi_{j+1}^k\}_{k\in[3]}, \Pi'_{j+1}) \leftarrow \mathsf{Ext}_{\mathsf{odd}}(c, \mathsf{SE}.K_{\mathsf{odd}})$
– if j is odd
  $(\mathsf{h}_j, \{b_j^k\}_{k\in[3]}, \{\Pi_j^k\}_{k\in[3]}, \Pi'_j) \leftarrow \mathsf{Ext}_{\mathsf{odd}}(c, \mathsf{SE}.K_{\mathsf{odd}})$
  $(\mathsf{h}_{j+1}, \{b_{j+1}^k\}_{k\in[3]}, \{\Pi_{j+1}^k\}_{k\in[3]}, \Pi'_{j+1}) \leftarrow \mathsf{Ext}_{\mathsf{even}}(c, \mathsf{SE}.K_{\mathsf{even}})$
– if $\mathsf{stSNARG.V}(\mathsf{crs}, (x,y), H_P, (c,\Pi)) = 1 \wedge ((x,y), T, P, H_P, \mathsf{crs}) \notin \mathcal{L}_{\mathcal{TM}} \wedge \mathsf{h}_j = \mathit{Step}(x, y, P, \mathsf{crs}, j).\bar{\mathsf{h}}_j$, return 1
– else return 0

Game $\mathsf{G}_{j,c}$ has the BARG key generation with a trapdoor at $j+1$.

Lemma 5.13. *Assuming key indistinguishability of* BARG, $\left|\Pr[\mathsf{G}^{\mathcal{A}}_{j,b} = 1] - \Pr[\mathsf{G}^{\mathcal{A}}_{j,c} = 1]\right| \leq \mathsf{negl}(\lambda)$.

This follows from the key indistinguishability of BARG as shown in the proof of Lemma 5.4 , hence we skip the detailed proof.

Inner Hybrid Game $\mathsf{G}_{j,d}$

– if j is even
  $\mathsf{SE}.K_{\mathsf{even}} \leftarrow \mathsf{SE.TGen}(1^\lambda, 1^M, I_{S_j})$
  $\mathsf{SE}.K_{\mathsf{odd}} \leftarrow \mathsf{SE.TGen}(1^\lambda, 1^M, I_{S_{j+1}})$
– if j is odd
  $\mathsf{SE}.K_{\mathsf{even}} \leftarrow \mathsf{SE.TGen}(1^\lambda, 1^M, I_{S_{j+1}})$
  $\mathsf{SE}.K_{\mathsf{odd}} \leftarrow \mathsf{SE.TGen}(1^\lambda, 1^M, I_{S_j})$
– $\mathsf{BARG.crs} \leftarrow \mathsf{BARG.TGen}(1^\lambda, 1^{T+1}, 1^{|C_{\mathsf{index}}|}, j+1)$

- $\mathsf{dk} \leftarrow \mathsf{HT.Gen}(1^\lambda)$
- $\mathsf{crs} := (\mathsf{SE}.K_{\mathsf{even}}, \mathsf{SE}.K_{\mathsf{odd}}, \mathsf{BARG.crs}, \mathsf{dk})$.
- $(P, \mathsf{aux}) \leftarrow \mathcal{A}_1(1^\lambda, \mathsf{crs}, \mathsf{aux})$
- $H_P \leftarrow \mathsf{stSNARG.TrustHash}(\mathsf{crs}, P)$
- $((x, y), (c, \Pi)) \leftarrow \mathcal{A}_2(\mathsf{crs}, P, \mathsf{aux})$
- if j is even
  $(\mathsf{h}_j, \{b_j^k\}_{k\in[3]}, \{\Pi_j^k\}_{k\in[3]}, \Pi'_j) \leftarrow \mathsf{Ext}_{\mathsf{even}}(c, \mathsf{SE}.K_{\mathsf{even}})$
  $(\mathsf{h}_{j+1}, \{b_{j+1}^k\}_{k\in[3]}, \{\Pi_{j+1}^k\}_{k\in[3]}, \Pi'_{j+1}) \leftarrow \mathsf{Ext}_{\mathsf{odd}}(c, \mathsf{SE}.K_{\mathsf{odd}})$
- if j is odd
  $(\mathsf{h}_j, \{b_j^k\}_{k\in[3]}, \{\Pi_j^k\}_{k\in[3]}, \Pi'_j) \leftarrow \mathsf{Ext}_{\mathsf{odd}}(c, \mathsf{SE}.K_{\mathsf{odd}})$
  $(\mathsf{h}_{j+1}, \{b_{j+1}^k\}_{k\in[3]}, \{\Pi_{j+1}^k\}_{k\in[3]}, \Pi'_{j+1}) \leftarrow \mathsf{Ext}_{\mathsf{even}}(c, \mathsf{SE}.K_{\mathsf{even}})$
- if $\mathsf{stSNARG.V}(\mathsf{crs}, (x, y), H_P, (c, \Pi)) = 1 \wedge ((x, y), T, P, H_P, \mathsf{crs}) \notin \mathcal{L}_{\mathcal{TM}} \wedge \mathsf{h}_j = \mathit{Step}(x, y, P, \mathsf{crs}, j).\bar{\mathsf{h}}_j \wedge \{b_{j+1}^k\}_{k\in[3]} = \mathit{Step}(x, y, P, \mathsf{crs}, j{+}1).\bar{b}_{j+1} \wedge \mathsf{rt}_{j+1}^3 = \mathit{Step}(x, y, P, \mathsf{crs}, j{+}1).\bar{\mathsf{rt}}_{j+1} \wedge \mathsf{st}_1 = \mathit{Step}(x, y, P, \mathsf{crs}, 1).\bar{\mathsf{st}}$, return 1
- else return 0

Game $\mathsf{G}_{j,c}$ is identical to $\mathsf{G}_{j,d}$ except that we added additional conditions for the adversary to win the game.

Lemma 5.14. *Assuming extraction correctness of* SE*, read and write soundness of* HT *and semi-adaptive somewhere soundness of* BARG, $\left|\Pr[\mathsf{G}_{j,c}^{\mathcal{A}} = 1] - \Pr[\mathsf{G}_{j,d}^{\mathcal{A}} = 1]\right| \leq \mathsf{negl}(\lambda)$.

The proof for this lemma is identical to the one for Lemma 5.8.

Inner Hybrid Game $\mathsf{G}_{j,e}$

- if j is even
  $\mathsf{SE}.K_{\mathsf{even}} \leftarrow \mathsf{SE.TGen}(1^\lambda, 1^M, I_{S_j})$
  $\mathsf{SE}.K_{\mathsf{odd}} \leftarrow \mathsf{SE.TGen}(1^\lambda, 1^M, I_{S_{j+1}})$
- if j is odd
  $\mathsf{SE}.K_{\mathsf{even}} \leftarrow \mathsf{SE.TGen}(1^\lambda, 1^M, I_{S_{j+1}})$
  $\mathsf{SE}.K_{\mathsf{odd}} \leftarrow \mathsf{SE.TGen}(1^\lambda, 1^M, I_{S_j})$
- $\mathsf{BARG.crs} \leftarrow \mathsf{BARG.TGen}(1^\lambda, 1^{T+1}, 1^{|C_{\mathsf{index}}|}, j+1)$
- $\mathsf{dk} \leftarrow \mathsf{HT.Gen}(1^\lambda)$
- $\mathsf{crs} := (\mathsf{SE}.K_{\mathsf{even}}, \mathsf{SE}.K_{\mathsf{odd}}, \mathsf{BARG.crs}, \mathsf{dk})$.
- $(P, \mathsf{aux}) \leftarrow \mathcal{A}_1(1^\lambda, \mathsf{crs})$
- $H_P \leftarrow \mathsf{stSNARG.TrustHash}(\mathsf{crs}, P)$
- $((x, y), (c, \Pi)) \leftarrow \mathcal{A}_2(\mathsf{crs}, P, \mathsf{aux})$
- if j is even
  $(\mathsf{h}_j, \{b_j^k\}_{k\in[3]}, \{\Pi_j^k\}_{k\in[3]}, \Pi'_j) \leftarrow \mathsf{Ext}_{\mathsf{even}}(c, \mathsf{SE}.K_{\mathsf{even}})$
  $(\mathsf{h}_{j+1}, \{b_{j+1}^k\}_{k\in[3]}, \{\Pi_{j+1}^k\}_{k\in[3]}, \Pi'_{j+1}) \leftarrow \mathsf{Ext}_{\mathsf{odd}}(c, \mathsf{SE}.K_{\mathsf{odd}})$
- if j is odd
  $(\mathsf{h}_j, \{b_j^k\}_{k\in[3]}, \{\Pi_j^k\}_{k\in[3]}, \Pi'_j) \leftarrow \mathsf{Ext}_{\mathsf{odd}}(c, \mathsf{SE}.K_{\mathsf{odd}})$
  $(\mathsf{h}_{j+1}, \{b_{j+1}^k\}_{k\in[3]}, \{\Pi_{j+1}^k\}_{k\in[3]}, \Pi'_{j+1}) \leftarrow \mathsf{Ext}_{\mathsf{even}}(c, \mathsf{SE}.K_{\mathsf{even}})$
- if $\mathsf{stSNARG.V}(\mathsf{crs}, (x, y), H_P, (c, \Pi)) = 1 \wedge ((x, y), T, P, H_P, \mathsf{crs}) \notin \mathcal{L}_{\mathcal{TM}} \wedge \mathsf{h}_j = \mathit{Step}(x, y, P, \mathsf{crs}, j).\bar{\mathsf{h}}_j \wedge \{b_{j+1}^k\}_{k\in[3]} = \mathit{Step}(x, y, P, \mathsf{crs}, j{+}1).\bar{b}_{j+1} \wedge \mathsf{rt}_{j+1}^3 = \mathit{Step}(x, P, \mathsf{crs}, j{+}1).\bar{\mathsf{rt}}_{j+1} \wedge \mathsf{st}_1 = \mathit{Step}(x, y, P, \mathsf{crs}, 1).\bar{\mathsf{st}} \mathsf{h}_{j+1} = \mathit{Step}(x, y, P, \mathsf{crs}, j+1).\bar{\mathsf{h}}$, return 1

– else return 0

Again, Game $\mathsf{G}_{j,e}$ is similar to $\mathsf{G}_{j,d}$ with modified winning conditions.

Lemma 5.15. *Assuming semi-adaptive somewhere soundness of* BARG, $|\Pr[\mathsf{G}^{\mathcal{A}}_{j,d} = 1] - \Pr[\mathsf{G}^{\mathcal{A}}_{j,e} = 1]| \leq \mathsf{negl}(\lambda)$.

This follows directly from the definition of h_{j+1} as analyzed in proof of lemma 5.9.

Inner Hybrid Game $\mathsf{G}_{j,f}$

– if j is even
  $\mathsf{SE}.K_{\mathsf{even}} \leftarrow \mathsf{SE.TGen}(1^\lambda, 1^M, I_{S_j})$
  $\mathsf{SE}.K_{\mathsf{odd}} \leftarrow \mathsf{SE.TGen}(1^\lambda, 1^M, I_{S_{j+1}})$
– if j is odd
  $\mathsf{SE}.K_{\mathsf{even}} \leftarrow \mathsf{SE.TGen}(1^\lambda, 1^M, I_{S_{j+1}})$
  $\mathsf{SE}.K_{\mathsf{odd}} \leftarrow \mathsf{SE.TGen}(1^\lambda, 1^M, I_{S_j})$
– $\mathsf{BARG.crs} \leftarrow \mathsf{BARG.TGen}(1^\lambda, 1^{T+1}, 1^{|C_{\mathsf{index}}|}, j+1)$
– $\mathsf{dk} \leftarrow \mathsf{HT.Gen}(1^\lambda)$
– $\mathsf{crs} := (\mathsf{SE}.K_{\mathsf{even}}, \mathsf{SE}.K_{\mathsf{odd}}, \mathsf{BARG.crs}, \mathsf{dk})$.
– $(P, \mathsf{aux}) \leftarrow \mathcal{A}_1(1^\lambda, \mathsf{crs})$
– $H_P \leftarrow \mathsf{stSNARG.TrustHash}(\mathsf{crs}, P)$
– $((x, y), (c, \Pi)) \leftarrow \mathcal{A}_2(\mathsf{crs}, P, \mathsf{aux})$
– if j is even
  $(\mathsf{h}_j, \{b^k_j\}_{k\in[3]}, \{\Pi^k_j\}_{k\in[3]}, \Pi'_j) \leftarrow \mathsf{Ext}_{\mathsf{even}}(c, \mathsf{SE}.K_{\mathsf{even}})$
  $(\mathsf{h}_{j+1}, \{b^k_{j+1}\}_{k\in[3]}, \{\Pi^k_{j+1}\}_{k\in[3]}, \Pi'_{j+1}) \leftarrow \mathsf{Ext}_{\mathsf{odd}}(c, \mathsf{SE}.K_{\mathsf{odd}})$
– if j is odd
  $(\mathsf{h}_j, \{b^k_j\}_{k\in[3]}, \{\Pi^k_j\}_{k\in[3]}, \Pi'_j) \leftarrow \mathsf{Ext}_{\mathsf{odd}}(c, \mathsf{SE}.K_{\mathsf{odd}})$
  $(\mathsf{h}_{j+1}, \{b^k_{j+1}\}_{k\in[3]}, \{\Pi^k_{j+1}\}_{k\in[3]}, \Pi'_{j+1}) \leftarrow \mathsf{Ext}_{\mathsf{even}}(c, \mathsf{SE}.K_{\mathsf{even}})$
– if $\mathsf{stSNARG.V}(\mathsf{crs}, (x, y), H_P, (c, \Pi)) = 1 \wedge ((x, y), T, P, H_P, \mathsf{crs}) \notin \mathcal{L}_{\mathcal{TM}} \wedge \mathsf{h}_{j+1} = \mathit{Step}(x, y, P, \mathsf{crs}, j+1).\bar{\mathsf{h}}$, return 1
– else return 0

Game $\mathsf{G}_{j,f}$ has a more relaxed winning condition than $\mathsf{G}_{j,e}$. This gives us the following lemma.

Lemma 5.16. $\Pr[\mathsf{G}^{\mathcal{A}}_{j,e} = 1] \leq \Pr[\mathsf{G}^{\mathcal{A}}_{j,f} = 1]$.

Again, this is identical to Lemma 5.10

Inner Hybrid Game $\mathsf{G}_{j,g}$

– if $j+1$ is even
  $\mathsf{SE}.K_{\mathsf{even}} \leftarrow \mathsf{SE.TGen}(1^\lambda, 1^M, I_{S_{j+1}})$
  $\mathsf{SE}.K_{\mathsf{odd}} \leftarrow \mathsf{SE.TGen}(1^\lambda, 1^M, I_{S_j})$
– if $j+1$ is odd
  $\mathsf{SE}.K_{\mathsf{even}} \leftarrow \mathsf{SE.TGen}(1^\lambda, 1^M, I_{S_j})$
  $\mathsf{SE}.K_{\mathsf{odd}} \leftarrow \mathsf{SE.TGen}(1^\lambda, 1^M, I_{S_{j+1}})$
– $\mathsf{BARG.crs} \leftarrow \mathsf{BARG.TGen}(1^\lambda, 1^{T+1}, 1^{|C_{\mathsf{index}}|}, j+1)$
– $\mathsf{dk} \leftarrow \mathsf{HT.Gen}(1^\lambda)$

- $\mathsf{crs} := (\mathsf{SE}.K_{\mathsf{even}}, \mathsf{SE}.K_{\mathsf{odd}}, \mathsf{BARG.crs}, \mathsf{dk})$.
- $(P, \mathsf{aux}) \leftarrow \mathcal{A}_1(1^\lambda, \mathsf{crs}, \mathsf{aux})$
- $H_P \leftarrow \mathsf{stSNARG.TrustHash}(\mathsf{crs}, P)$
- $((x, y), (c, \Pi)) \leftarrow \mathcal{A}_2(\mathsf{crs}, P, \mathsf{aux})$
- if $j+1$ is even
  $(\mathsf{h}_{j+1}, \{b^k_{j+1}\}_{k\in[3]}, \{\Pi^k_{j+1}\}_{k\in[3]}, \Pi'_{j+1}) \leftarrow \mathsf{Ext}_{\mathsf{even}}(c, \mathsf{SE}.K_{\mathsf{even}})$
  $(\mathsf{h}_j, \{b^k_j\}_{k\in[3]}, \{\Pi^k_j\}_{k\in[3]}, \Pi'_j) \leftarrow \mathsf{Ext}_{\mathsf{odd}}(c, \mathsf{SE}.K_{\mathsf{odd}})$
- if $j+1$ is odd
  $(\mathsf{h}_j, \{b^k_j\}_{k\in[3]}, \{\Pi^k_j\}_{k\in[3]}, \Pi'_j) \leftarrow \mathsf{Ext}_{\mathsf{even}}(c, \mathsf{SE}.K_{\mathsf{even}})$
  $(\mathsf{h}_{j+1}, \{b^k_{j+1}\}_{k\in[3]}, \{\Pi^k_{j+1}\}_{k\in[3]}, \Pi'_{j+1}) \leftarrow \mathsf{Ext}_{\mathsf{odd}}(c, \mathsf{SE}.K_{\mathsf{odd}})$
- if $\mathsf{stSNARG.V}(\mathsf{crs}, (x, y), H_P, (c, \Pi)) = 1 \wedge ((x, y), T, P, H_P, \mathsf{crs}) \notin \mathcal{L}_{\mathcal{TM}} \wedge \mathsf{h}_{j+1} = \mathit{Step}(x, y, P, \mathsf{crs}, j+1).\bar{\mathsf{h}}$, return 1
- else return 0

Game $\mathsf{G}_{j,g}$ is identical to $\mathsf{G}_{j,f}$ with the indices renamed. Thus, $\Pr[\mathsf{G}^{\mathcal{A}}_{j,g} = 1] = \Pr[\mathsf{G}^{\mathcal{A}}_{j,f} = 1]$. Observe $\mathsf{G}_{j,g}$ is identical to outer Game G_{j+1}.

Thus, combining the lemmas above, we get

Lemma 5.17. *Assuming extraction correctness of* SE*, semi-adaptive somewhere soundness of* BARG*, read and write soundness of* HT*,*

$$\Pr[\mathsf{G}^{\mathcal{A}}_j = 1] \leq \Pr[\mathsf{G}^{\mathcal{A}}_{j+1} = 1] + \mathsf{negl}(\lambda).$$

This follows from the combination of previous lemmas where we showed that the winning probability in the sequence of inner hybrids are either negligibly close to each other or increases (from Game $\mathsf{G}_{j,e}$ to Game $\mathsf{G}_{j,f}$).

Finally, we will show that the winning probability of $\mathcal{A}$ is 0 in the final game G_T.

Lemma 5.18. *Assuming extraction correctness of* SE *hash,*

$$\Pr[\mathsf{G}^{\mathcal{A}}_T = 1] = 0.$$

Proof. The extraction correctness of SE ensures that h_T was indeed the state committed by the prover. Now, $\mathsf{h}_T = \bar{\mathsf{h}}_T$ cannot be true since our assumption of $(x, T, P, H_P, \mathsf{crs}) \notin \mathcal{L}_{\mathcal{TM}}$ means that Turing Machine state after T steps cannot be an accept state. Thus, the adversary's win conditions cannot be simultaneously satisfied.

Note that this step does not require us to resort to BARG soundness. Due to our specific construction of $\bar{\mathsf{h}}_T$, all we need ensure is that the state committed by the prover does not correspond to the correct state. □

Compiling the lemmas together and using chain rule, it must be true that

$$\Pr[\mathsf{G}^{\mathcal{A}} = 1] \leq \mathsf{negl}(\lambda)$$

which is a contradiction to our assumption that the scheme is not sound.

Lemma 5.19. *Assuming $T = \mathsf{poly}(m, n)$, $T, m, n \leq 2^\lambda$, the* stSNARG *protocol in Figure 2 implies the unconditional existence of a publicly verifiable non interactive succinct delegation scheme* sDel *as defined above.*

Proof. We provide an explicit construction of sDel assuming a semi-trusted SNARG stSNARG. Without loss of generality, we can assume that T is known a-priory.

- $\mathsf{sDel.Setup}(1^\lambda)$: Run stSNARG.Setup to generate crs.
- $\mathsf{sDel.ProgAuth}(1^\lambda, \mathsf{crs})$: Generate a program $P \in \{0,1\}^m$, state and run $\mathsf{stSNARG.TrustHash}(\mathsf{crs}, P)$ to get H_P.

- $\mathsf{sDel}.I(1^\lambda, \mathsf{crs})$: Generate $x \in \{0,1\}^n$.
- $\mathsf{sDel}.W(\mathsf{crs}, P, \mathsf{state}, H_P, x)$: Generate $y \in \{0,1\}$ and run $\mathsf{stSNARG.P}(\mathsf{crs}, P, x, y, H_P)$ to get Π.
- $\mathsf{sDel}.V(\mathsf{crs}, x, y, H_P, \Pi)$: Run $\mathsf{stSNARG.V}(\mathsf{crs}, x, y, H_P, \Pi)$ return V's output.

Completeness of sDel follows from the completeness of stSNARG in a straightforward way. The proof size and verifier run time of stSNARG is $\mathsf{poly}(\lambda, \log T)$=$\mathsf{poly}(\lambda, \log |P|, \log |x|)$. Similarly, the prover run time of sDel is also $\mathsf{poly}(\lambda, |P|, |x|)$.

Soundness: Let us assume the there exists an adversary $\mathcal{A} := (\mathcal{A}_1, \mathcal{A}_2)$ which wins the sDel soundness game as described above. We use $\mathcal{A}$ to construct an adversary $\mathcal{B} := (\mathcal{B}_1, \mathcal{B}_2)$ which can win the stSNARG soundness game as shown in Game G (cf. Previous Section on semi-trusted SNARGS). Following is the reduction.

Adversary $\mathcal{B} := (\mathcal{B}_1, \mathcal{B}_2)$ playing stSNARG soundness.

$\mathcal{B}_1(1^\lambda, \mathsf{crs})$

- $((P, \mathsf{state}), H_P) \leftarrow \mathsf{sDel.ProgAuth}(1^\lambda, \mathsf{crs})$
- Output $(P, \mathsf{aux} = (\mathsf{state}, H_P))$

$\mathcal{B}_2(\mathsf{crs}, P, H^*_P, \mathsf{aux})$

- $(x, \mathsf{aux}') \leftarrow \mathcal{A}_1(1^\lambda, \mathsf{crs})$
- $(y, \Pi) \leftarrow \mathcal{A}_2(\mathsf{crs}, P, \mathsf{aux}, x, \mathsf{aux}')$
- Return $((x, y), (c, \Pi))$

Note that in the reduction above, $H^*_P = \mathsf{stSNARG.TrustHash}(\mathsf{crs}, P)$ which is input to $\mathcal{B}_2$. In fact, our construccction of sDel ensures that $H^*_P = H_P$, but this is not relevant for the context of this reduction. By our assumption that sDel is not sound, we have that $P(x) \neq y$ and $\mathsf{stSNARG.V}(\mathsf{crs}, x, y, H_P, \Pi) = 1$. Also by definition, $P(x) \neq y \implies ((x, y), T, P, H^*_P, \mathsf{crs}) \notin \mathcal{L}_{\mathcal{TM}}$ Thus, $\mathcal{B}$ clearly wins the stSNARG soundness game. □

6 Semi-Trusted Succinct Non-Interactive Argument with Zero Knowledge (ZK-stSNARG)

A publicly verifiable semi-trusted non interactive argument with zero-knowledge scheme ZKstSNARG : (ZKstSNARG.Setup, ZKstSNARG.TrustHash, ZKstSNARG.P, ZKstSNARG.V) is defined as

- $\mathsf{ZKstSNARG.Setup}(1^\lambda, 1^T)$: A randomized setup algorithm which on input security parameter λ, and number of Turing Machine steps T, outputs crs.
- $\mathsf{ZKstSNARG.TrustHash}(\mathsf{crs}, P)$: A deterministic an honest algorithm which on input crs and a program $P \in \{0,1\}^m$ for some $m < 2^\lambda$, computes a succinct digest H_P of P. It then produces a statistically binding and extractable commitment C_P of H_P under randomness r_1. It then gives out a pair public output $\mathsf{POut} = C_P$ and private output $\mathsf{SOut} = (H_P, r)$. Here SOut is made available to the prover only.
- $\mathsf{ZKstSNARG.P}(\mathsf{crs}, P, x, y, \mathsf{SOut}, \mathsf{POut})$: A deterministic prover algorithm which on input the crs, $P \in \{0,1\}^m$ for some $m < 2^\lambda$, $x \in \{0,1\}^n$ for some $n < 2^\lambda$, $y \in \{0,1\}$, SOut, and POut outputs a proof Π.
- $\mathsf{ZKstSNARG.V}(\mathsf{crs}, x, y, \mathsf{POut}, \Pi)$: A deterministic verification algorithm which on input crs, x, y, public output POut of stSNARG.TrustHash and proof Π, either accepts(output 1) or rejects(output 0) it.

We define the following language

$$\mathcal{L}_{\mathcal{TM}} := \{(P, x, y, T, \mathsf{POut}, \mathsf{crs}) \,|\, \exists (H_P, r_1) \text{ such that } \mathcal{TM}(P, x, y) = 1 \wedge (\mathsf{POut}, (H_P, r_1)) = \mathsf{ZKstSNARG.TrustHash}(\mathsf{crs}, P)\}.$$

A ZKstSNARG satisfies the following properties:

- **Completeness.** For every $\lambda, T, n, m \in \mathbb{N}$ such that $T, n, m < 2^\lambda$, program $P \in \{0,1\}^m$, input $x \in \{0,1\}^n$ and output $y \in \{0,1\}$ such that $(P, x, y, T, \mathsf{POut}, \mathsf{crs}) \in \mathcal{L}_{\mathcal{TM}}$, we have

$$\Pr[\mathsf{ZKstSNARG.V}(\mathsf{crs}, x, y, \mathsf{POut}, \Pi) = 1 \big| \mathsf{crs} \leftarrow \mathsf{ZKstSNARG.Setup}(1^\lambda, 1^T), \\ (\mathsf{POut}, \mathsf{SOut})) \leftarrow \mathsf{ZKstSNARG.TrustHash}(\mathsf{crs}, P), \Pi \coloneqq \mathsf{ZKstSNARG.P}(\mathsf{crs}, x, y, \mathsf{POut}, \mathsf{SOut})] = 1.$$

- **Efficiency.** $\mathsf{ZKstSNARG.Setup}$ runs in time $\mathsf{poly}(\lambda, T)$, $\mathsf{ZKstSNARG.TrustHash}$ runs in time $\mathsf{poly}(\lambda, |P|, T)$, $\mathsf{ZKstSNARG.P}$ runs in time $\mathsf{poly}(\lambda, |x|, |P|, T)$ and outputs a proofs of length $\mathsf{poly}(\lambda, \log T)$, and $\mathsf{ZKstSNARG.V}$ runs in time $\mathsf{poly}(\lambda, \log T)$.

- **Soundness.** For every PPT adversary $\mathcal{A} \coloneqq (\mathcal{A}_1, \mathcal{A}_2)$ and the tuple $T = T(\lambda), n = n(\lambda), m = m(\lambda)$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that for every $\lambda \in \mathbb{N}$,

$$\Pr[\mathsf{ZKstSNARG.V}(\mathsf{crs}, x, y, \mathsf{POut}, \Pi) = 1 \wedge (P, x, y, T, \mathsf{POut}, \mathsf{SOut}, \mathsf{crs}) \notin \mathcal{L}_{\mathcal{TM}} \big| \\ \mathsf{crs} \leftarrow \mathsf{ZKstSNARG.Setup}(1^\lambda, 1^T), (P, \mathsf{aux}) \leftarrow \mathcal{A}_1(1^\lambda), \\ (\mathsf{POut}, \mathsf{SOut}) \leftarrow \mathsf{ZKstSNARG.TrustHash}(\mathsf{crs}, P), (x, y, \Pi) \leftarrow \mathcal{A}_2(\mathsf{crs}, P, \mathsf{POut}, \mathsf{SOut}, \mathsf{aux})] \leq \mathsf{negl}(\lambda).$$

- **Non Interactive Zero Knowledge.** For all $(P, x, y, T, \mathsf{POut}, \mathsf{crs}) \in \mathcal{L}_{\mathcal{TM}}$, there exists a PPT simulator $\mathsf{Sim} \coloneqq (\mathsf{Sim}_1, \mathsf{Sim}_2, \mathsf{Sim}_3)$ such that the distributions of

$$(\mathsf{crs}, x, y, \mathsf{POut}, \Pi) \big| (\mathsf{crs}, \mathsf{aux}) \leftarrow \mathsf{Sim}_1(1^\lambda, 1^T), \\ (\mathsf{POut}, \mathsf{aux}') \leftarrow \mathsf{Sim}_2(\mathsf{crs}, \mathsf{aux}), \\ \Pi \leftarrow \mathsf{Sim}_3(\mathsf{aux}', \mathsf{crs}, (x, y), \mathsf{POut})$$

and

$$(\mathsf{crs}, x, y, \mathsf{POut}, \Pi) \big| \mathsf{crs} \leftarrow \mathsf{ZKstSNARG.Setup}(1^\lambda, 1^T), \\ (\mathsf{POut}, \mathsf{SOut}) \leftarrow \mathsf{ZKstSNARG.TrustHash}(\mathsf{crs}, P), \\ \Pi \leftarrow \mathsf{ZKstSNARG.P}(\mathsf{crs}, P, x, y, \mathsf{POut}, \mathsf{SOut})$$

are indistinguishable.

To extend our delegation scheme to achieve non interactive zero knowledge, we use the following additional primitives, namely (1) a statistically binding extractable commitment scheme $\mathsf{Com}_{\mathsf{bind}}$ as defined in Section 3, and (2) a Non Interactive Zero Knowledge argument $\mathsf{NIZK} \coloneqq (\mathsf{NIZK.Gen}, \mathsf{NIZK.P}, \mathsf{NIZK.V})$.

The protocol in Figure 5 demonstrates the extension of $\mathsf{stSNARG}$ to achieve Zero-Knowledge. The CRS in Figure 5 contains a statistically binding commitment to 0. This lets us extend $\mathcal{L}_{\mathcal{TM}}$ to the language,

$$\mathcal{L}_{\mathsf{hyb}} \coloneqq \Bigg\{ (P, x, y, T, C_P, \mathsf{crs}) \Big| \exists (H_P, r_1) \text{ such that } \mathcal{TM}(P, x, y) = 1 \wedge (C_P, (H_P, r_1)) = \mathsf{ZKstSNARG.TrustHash}(\mathsf{crs}, P) \\ \vee \Big(\exists r \text{ such that } \mathsf{crs} \text{ contains a commitment to 1 under randomness } r \Big). \Bigg\}$$

such that any witness to $\mathcal{L}_{\mathcal{TM}}$ is vacuously a witness to $\mathcal{L}_{\mathsf{hyb}}$ due to binding property of the commitment. We use NIZK for the following NP language:

$$\mathcal{L} := \Big\{ (c.\mathsf{com}, \Pi.\mathsf{com}, (\mathsf{crs}, x, y, T), C_P) \Big| \exists r_1, r_2, r_3, r_4, c, \Pi, H_P \text{ such that}$$

$$\Big(C_P = \mathsf{Com.C}(\mathsf{Com}_{\mathsf{bind}}.Key_1, H_P; r_1) \wedge c.\mathsf{com} = \mathsf{Com.C}(\mathsf{Com}_{\mathsf{bind}}.Key_2, c; r_2)$$

$$\wedge \Pi.\mathsf{com} = \mathsf{Com.C}(\mathsf{Com}_{\mathsf{bind}}.Key_3, \Pi; r_3) \wedge \mathsf{stSNARG.V}(\mathsf{crs}, ((x, y), T, H_P), (c, \Pi)) = 1 \Big)$$

$$\vee \mathsf{crs} \text{ contains } \mathsf{Com.C}(\mathsf{Com}_{\mathsf{bind}}.Key_4, 1; r_4) \Big\}$$

Also, note that in this construction, the underlying stSNARG is built for the index circuit C'_{index}.

Protocol 2 (Semi-Trusted Non-Interactive Argument with Zero-Knowledge)**.**

- $\mathsf{ZKstSNARG.Setup}(1^\lambda, T)$:
  - $\mathsf{crs}_1 \leftarrow \mathsf{stSNARG.Setup}(1^\lambda, 1^T)$
  - $\mathsf{Com}_{\mathsf{bind}}.Key_1 \leftarrow \mathsf{Com.Gen}(1^\lambda)$
  - $\mathsf{Com}_{\mathsf{bind}}.Key_2 \leftarrow \mathsf{Com.Gen}(1^\lambda)$
  - $\mathsf{Com}_{\mathsf{bind}}.Key_3 \leftarrow \mathsf{Com.Gen}(1^\lambda)$
  - $\mathsf{Com}_{\mathsf{bind}}.Key_4 \leftarrow \mathsf{Com.Gen}(1^\lambda), r_4 \leftarrow\!\!\$\ \{0,1\}^\lambda, z \leftarrow \mathsf{Com.C}(\mathsf{Com}.Key_4, 0; r_4)$
  - $\mathsf{NIZK.crs} \leftarrow \mathsf{NIZK.Gen}(1^\lambda)$
  - return $(\mathsf{crs}_1, \mathsf{Com}_{\mathsf{bind}}.Key_1.\mathsf{Com}_{\mathsf{bind}}.Key_2, \mathsf{Com}_{\mathsf{bind}}.Key_3, z, \mathsf{NIZK.crs})$.
- $\mathsf{ZKstSNARG.TrustHash}(\mathsf{crs}, P)$
  - $H_P \leftarrow \mathsf{stSNARG.TrustHash}(\mathsf{crs}, P)$
  - $r_1 \leftarrow\!\!\$\ \{0,1\}^\lambda$, $C_P \leftarrow \mathsf{Com.C}(\mathsf{Com}_{\mathsf{bind}}.Key_1, H_P; r_1)$ return $(\mathsf{SOut} := (P, r_1), \mathsf{POut} := C_P)$.
- $\mathsf{ZKstSNARG.P}(\mathsf{crs}, x, y, \mathsf{SOut}, \mathsf{POut})$:
  - $(c, \Pi) \leftarrow \mathsf{stSNARG.P}(\mathsf{crs}, x, y, H_P)$
  - $r_2 \leftarrow\!\!\$\ \{0,1\}^\lambda, c.\mathsf{com} \leftarrow \mathsf{Com.C}(\mathsf{Com}_{\mathsf{bind}}.Key_2, c; r_2)$
  - $r_3 \leftarrow\!\!\$\ \{0,1\}^\lambda, \Pi.\mathsf{com} \leftarrow \mathsf{Com.C}(\mathsf{Com}_{\mathsf{bind}}.Key_3, \Pi; r_3)$
  - $\mathsf{NIZK}.\Pi \leftarrow \mathsf{NIZK.Prove}\left(\mathsf{NIZK.crs}, (c.\mathsf{com}, \Pi.\mathsf{com}, (\mathsf{crs}, x, y, T), C_P), ((H_P, r_1), (c, r_2), (\Pi, r_3), \perp)\right)$
  - return $(c.\mathsf{com}, \Pi.\mathsf{com}, \mathsf{NIZK}.\Pi)$.
- $\mathsf{ZKstSNARG.V}(\mathsf{crs}, (x, y), \mathsf{POut} = C_P, c.\mathsf{com}, \Pi.\mathsf{com}, \mathsf{NIZK}.\Pi)$:
  - return 1 if and only if $\mathsf{NIZK.V}(\mathsf{NIZK.crs}, (c.\mathsf{com}, \Pi.\mathsf{com}, (\mathsf{crs}, x, y, T), C_P), \mathsf{NIZK}.\Pi) = 1$.

Figure 5: Semi-Trusted Universal Turing Machine Delegation with Non Interactive Zero-Knowledge

Theorem 6.1. *Assuming the existence of semi-trusted SNARGs and Extractable Statistically Binding Commitment Schemes, and NIZK as described in sections 3 and 5, Figure 5 is a publicly verifiable non-interactive semi-trusted SNARG with zero knowledge such that CRS size, proof size and verifier time are* $\mathsf{poly}(\lambda, \log T)$ *and prover run time is* $\mathsf{poly}(\lambda, T)$.

Completeness. An honest prover ignores the additional commitment to 0 in the CRS and follows Figure 5. Also, observe that for any honest prover, any witness to an instance in $\mathcal{L}_{\mathsf{hyb}}$ is also a witness for the language $\mathcal{L}_{\mathsf{hyb}}$ with the same instance. Now, $\mathsf{BARG.V}(\mathsf{BARG.crs}, C_{\mathsf{index}}, \Pi) = 1$ follows from the completeness of the

Circuit 2 (Circuit C'_{index}).

- **Hard-coded:** $y, \mathsf{start}, \phi, \mathsf{SE}.K_{\mathsf{even}}, \mathsf{SE}.K_{\mathsf{odd}}, T, H_x := \mathsf{HT.Hash}(\mathsf{dk}, x)$
- **Input:**

  $\left(i, (c, H_P, \mathsf{h}_i := (\mathsf{st}_i, \mathsf{rt}_i^1, \mathsf{rt}_i^2, \mathsf{rt}_i^3), \rho_{\mathsf{h}_i})\right)$, if $i = 0$

  $\left(i, (c, H_P, \{\mathsf{h}_{i-1}, \mathsf{h}_i, \{b_i^j, \Pi_i^j\}_{j\in[3]}, \Pi'_i, \rho_{\mathsf{h}_{i-1}}, \rho_{\mathsf{h}_i}, \{\rho_{b_i^j}, \rho_{\Pi_i^j}\}_{j\in[3]}, \rho_{\Pi'_i}\})\right)$, $\forall i \in [T]$

- **Output:** return 1 if and only if
  - if $i = 0$
    a. $\mathsf{st}_0 = \mathsf{start}$
    b. $H_x = \mathsf{rt}_0^1$
    c. $H_P = \mathsf{rt}_0^2$
    d. $\mathsf{HT.Hash}(\mathsf{dk}, \square)$ has rt_0^3 as root
  - else
    * if i is even:
      a. $\mathsf{SE.Verify}(\mathsf{SE}.K_{\mathsf{odd}}, c_{\mathsf{odd}}, \mathsf{h}_{i-1}, \rho_{\mathsf{h}_{i-1}}) = 1$
      b. $\mathsf{SE.Verify}(\mathsf{SE}.K_{\mathsf{even}}, c_{\mathsf{even}}, \mathsf{h}_i, \rho_{\mathsf{h}_i}) = 1$
      c. $\left\{\mathsf{SE.Verify}(\mathsf{SE}.K_{\mathsf{even}}, c_{\mathsf{even}}, b_i^j, \rho_{b_i^j}) = 1\right\}_{j\in[3]}$
      d. $\left\{\mathsf{SE.Verify}(\mathsf{SE}.K_{\mathsf{even}}, c_{\mathsf{even}}, \Pi_i^j, \rho_{\Pi_i^j}) = 1\right\}_{j\in[3]}$
      e. $\mathsf{SE.Verify}(\mathsf{SE}.K_{\mathsf{even}}, c_{\mathsf{even}}, \Pi'_i, \rho_{\Pi'_i}) = 1$
    * if i is odd:
      a. $\mathsf{SE.Verify}(\mathsf{SE}.K_{\mathsf{even}}, c_{\mathsf{even}}, \mathsf{h}_{i-1}, \rho_{\mathsf{h}_{i-1}}) = 1$
      b. $\mathsf{SE.Verify}(\mathsf{SE}.K_{\mathsf{odd}}, c_{\mathsf{odd}}, \mathsf{h}_i, \rho_{\mathsf{h}_i}) = 1$
      c. $\left\{\mathsf{SE.Verify}(\mathsf{SE}.K_{\mathsf{odd}}, c_{\mathsf{odd}}, b_i^j, \rho_{b_i^j}) = 1\right\}_{j\in[3]}$
      d. $\left\{\mathsf{SE.Verify}(\mathsf{SE}.K_{\mathsf{odd}}, c_{\mathsf{odd}}, \Pi_i^j, \rho_{\Pi_i^j}) = 1\right\}_{j\in[3]}$
      e. $\mathsf{SE.Verify}(\mathsf{SE}.K_{\mathsf{odd}}, c_{\mathsf{odd}}, \Pi'_i, \rho_{\Pi'_i}) = 1$
    * $\phi(\mathsf{h}_{i-1}, \mathsf{h}_i, \{b_i^j, \Pi_i^j\}_{j\in[3]}, \Pi'_i) = 1$
    * if $i = T$
      a. $\mathsf{HT.Hash}(\mathsf{dk}, y)$ has rt_T^3 as root.
      b. st_T indeed encodes the accept state.

Figure 6: Circuit C'_{index}

underlying SE hash, BARG, and read and write completeness of HT as seen in the completeness proof from the previous section. Clearly, the result will follow from the completeness of the underlying NIZK.

Efficiency. The following points follow from the above lemma, and the efficiency of the SE hash and hash tree construction.

- $|C'_{\mathsf{index}}| = \mathsf{poly}(\lambda, \log T)$.
- CRS Size: By the corresponding properties of the underlying primitives, $|\mathsf{crs}| = \mathsf{poly}(\lambda, \log T)$. The only addition here are the NIZK CRS and the commitment keys which are $\mathsf{poly}(\lambda)$.
- Proof Length: $|c.\mathsf{com}| + |\Pi.\mathsf{com}| + |\mathsf{NIZK}.\Pi| = \mathsf{poly}(\lambda, \log T) + \mathsf{poly}(\lambda, |C'_{\mathsf{index}}|) + \mathsf{poly}(\lambda) = \mathsf{poly}(\lambda, \log T)$.
- Verifier Time: Time taken to compute C'_{index} and verify the NIZK. This is $\mathsf{poly}(\lambda, \log T, |C'_{\mathsf{index}}|) = \mathsf{poly}(\lambda, \log T)$.

Soundness. Let us assume for the sake of contradiction that our scheme in Figure 5 is not sound, i.e., there exists a PPT adversary $\mathcal{A} := (\mathcal{A}_1, \mathcal{A}_2)$, a value T and a polynomial function $\mathsf{poly}(\lambda)$ such that for infinitely many values of $\lambda \in \mathbb{N}$,

$$\Pr[\mathsf{G}^{\mathcal{A}} = 1] \geq \frac{1}{\mathsf{poly}(\lambda)},$$

where $\mathcal{A}$ plays Game G described below,

Real Game G

- $\mathsf{crs} \leftarrow \mathsf{stSNARG.Setup}(1^\lambda, 1^T)$
- $(P, \mathsf{aux}) \leftarrow \mathcal{A}_1(1^\lambda, \mathsf{crs})$
- $(C_P, (\bar{H}_P, \bar{r_1})) \leftarrow \mathsf{stSNARG.TrustHash}(\mathsf{crs}, P)$
- $((x, y)(c.\mathsf{com}, \Pi.\mathsf{com}.\mathsf{NIZK}.\Pi)) \leftarrow \mathcal{A}_2(\mathsf{crs}, P, C_P, \bar{H}_P, \bar{r_1}, \mathsf{aux})$
- if $\mathsf{ZKstSNARG.V}(\mathsf{crs}, (x, y), \mathsf{POut} = C_P, c.\mathsf{com}, \Pi.\mathsf{com}, \mathsf{NIZK}.\Pi) = 1 \wedge (P, x, y, T, \mathsf{POut}, \mathsf{crs}) \notin \mathcal{L}_{\mathcal{TM}}$, return 1
- else return 0

We proceed by using a sequence of hybrids.

Hybrid Game G_1

- $\mathsf{crs}_1 \leftarrow \mathsf{stSNARG.Setup}(1^\lambda, 1^T)$
- $\mathsf{Com}_{\mathsf{bind}}.Key_1 \leftarrow \mathsf{Com.TGen}(1^\lambda)$
- $\mathsf{Com}_{\mathsf{bind}}.Key_2 \leftarrow \mathsf{Com.Gen}(1^\lambda)$
- $\mathsf{Com}_{\mathsf{bind}}.Key_3 \leftarrow \mathsf{Com.Gen}(1^\lambda)$
- $\mathsf{Com}_{\mathsf{bind}}.Key_4 \leftarrow \mathsf{Com.Gen}(1^\lambda), r_4 \leftarrow\!\!\$\ \{0,1\}^\lambda, z \leftarrow \mathsf{Com.C}(\mathsf{Com}.Key_4, 0; r_4)$
- $\mathsf{NIZK.crs} \leftarrow \mathsf{NIZK.Gen}(1^\lambda)$
- $\mathsf{crs} := (\mathsf{crs}_1, \mathsf{Com}_{\mathsf{bind}}.Key_1.\mathsf{Com}_{\mathsf{bind}}.Key_2, \mathsf{Com}_{\mathsf{bind}}.Key_3, z, \mathsf{NIZK.crs})$
- $(P, \mathsf{aux}) \leftarrow \mathcal{A}_1(1^\lambda, \mathsf{crs})$
- $(C_P, (\bar{H}_P, r)) \leftarrow \mathsf{stSNARG.TrustHash}(\mathsf{crs}, P)$
- $((x, y)(c.\mathsf{com}, \Pi.\mathsf{com}.\mathsf{NIZK}.\Pi)) \leftarrow \mathcal{A}_2(\mathsf{crs}, P, C_P, \bar{H}_P, \bar{r_1}, \mathsf{aux})$
- if $\mathsf{ZKstSNARG.V}(\mathsf{crs}, (x, y), \mathsf{POut} = C_P, c.\mathsf{com}, \Pi.\mathsf{com}, \mathsf{NIZK}.\Pi) = 1 \wedge (P, x, y, T, \mathsf{POut}, \mathsf{crs}) \notin \mathcal{L}_{\mathcal{TM}}$, return 1
- else return 0

Lemma 6.2. *Assuming CRS indistinguishability of* $\mathsf{Com}_{\mathsf{bind}}$, $|Pr[\mathsf{G}^{\mathcal{A}} = 1] - \Pr[\mathsf{G}_1^{\mathcal{A}} = 1]| \leq \mathsf{negl}(\lambda)$.

Proof. The only difference in Game G and G_1 is that the key generation algorithm of the commitment $\mathsf{Com}_{\mathsf{bind}}$ ($\mathsf{Com.Gen}$) is replaced by the trapdoor key generation ($\mathsf{Com.TGen}$).

If $\left|\Pr[\mathsf{G}^{\mathcal{A}} = 1] - \Pr[\mathsf{G}_1^{\mathcal{A}} = 1]\right| > \mathsf{negl}(\lambda)$, then one can construct a PPT adversary $\mathcal{B}$ that breaks the CRS indistinguishability of $\mathsf{Com}_{\mathsf{bind}}$ with Key as input from the key generation algorithm of the SE hash as follows:

Adversary $\mathcal{B}$ playing $\mathsf{Com}_{\mathsf{bind}}$ CRS indistinguishability game.

- $\mathsf{crs}_1 \leftarrow \mathsf{stSNARG.Setup}(1^\lambda, 1^T)$
- $\mathsf{Com}_{\mathsf{bind}}.Key_1 \leftarrow \mathsf{Key}$
- $\mathsf{Com}_{\mathsf{bind}}.Key_2 \leftarrow \mathsf{Com.Gen}(1^\lambda)$
- $\mathsf{Com}_{\mathsf{bind}}.Key_3 \leftarrow \mathsf{Com.Gen}(1^\lambda)$
- $\mathsf{Com}_{\mathsf{bind}}.Key_4 \leftarrow \mathsf{Com.Gen}(1^\lambda), r_4 \leftarrow\!\!\$\ \{0,1\}^\lambda, z \leftarrow \mathsf{Com.C}(\mathsf{Com}.Key_4, 0; r_4)$
- $\mathsf{NIZK.crs} \leftarrow \mathsf{NIZK.Gen}(1^\lambda)$
- $\mathsf{crs} := (\mathsf{crs}_1, \mathsf{Com}_{\mathsf{bind}}.Key_1.\mathsf{Com}_{\mathsf{bind}}.Key_2, \mathsf{Com}_{\mathsf{bind}}.Key_3, z, \mathsf{NIZK.crs})$
- $(P, \mathsf{aux}) \leftarrow \mathcal{A}_1(1^\lambda, \mathsf{crs})$
- $(C_P, (\bar{H}_P, \bar{r_1})) \leftarrow \mathsf{stSNARG.TrustHash}(\mathsf{crs}, P)$
- $((x, y)(c.\mathsf{com}, \Pi.\mathsf{com}.\mathsf{NIZK}.\Pi)) \leftarrow \mathcal{A}_2(\mathsf{crs}, P, C_P, \bar{H}_P, \bar{r_1}, \mathsf{aux})$

- if $\mathsf{ZKstSNARG.V}(\mathsf{crs}, (x, y), \mathsf{POut} = C_P, c.\mathsf{com}, \Pi.\mathsf{com}, \mathsf{NIZK}.\Pi) = 1 \wedge (P, x, y, T, \mathsf{POut}, \mathsf{crs}) \notin \mathcal{L}_{\mathcal{TM}}$, return 1
- else return 0

Here, Key is either $\mathsf{SE.Gen}(1^\lambda)$ or $\mathsf{SE.TGen}(1^\lambda)$ based on whether $\mathcal{A}$ is interacting with game G or G_a respectively. Adversary $\mathcal{B}$ here breaks the CRS indistinguishability of SE hash if the probability that it returns 1 is significantly different when $\mathcal{A}$ interacts with the key generation algorithm in normal mode vs trapdoor mode. Now, the probability that $\mathcal{B}$ returns 1 in either case is exactly equal to the probability that $\mathcal{A}$ wins the corresponding games, hence, $\mathcal{B}$ breaks if $\left|\Pr[\mathsf{G}^{\mathcal{A}} = 1] - \Pr[\mathsf{G}_a^{\mathcal{A}} = 1]\right| \geq \mathsf{negl}(\lambda)$. This leads to a contradiction of our assumption. □

Hybrid Game G_2

- $\mathsf{crs}_1 \leftarrow \mathsf{stSNARG.Setup}(1^\lambda, 1^T)$
- $\mathsf{Com}_{\mathsf{bind}}.Key_1 \leftarrow \mathsf{Com.TGen}(1^\lambda)$
- $\mathsf{Com}_{\mathsf{bind}}.Key_2 \leftarrow \mathsf{Com.TGen}(1^\lambda)$
- $\mathsf{Com}_{\mathsf{bind}}.Key_3 \leftarrow \mathsf{Com.TGen}(1^\lambda)$
- $\mathsf{Com}_{\mathsf{bind}}.Key_4 \leftarrow \mathsf{Com.TGen}(1^\lambda), r_4 \leftarrow\!\!\$\ \{0,1\}^\lambda, z \leftarrow \mathsf{Com.C}(\mathsf{Com}.Key_4, 0; r_4)$
- $\mathsf{NIZK.crs} \leftarrow \mathsf{NIZK.Gen}(1^\lambda)$
- $\mathsf{crs} := (\mathsf{crs}_1, \mathsf{Com}_{\mathsf{bind}}.Key_1.\mathsf{Com}_{\mathsf{bind}}.Key_2, \mathsf{Com}_{\mathsf{bind}}.Key_3, z, \mathsf{NIZK.crs})$
- $(P, \mathsf{aux}) \leftarrow \mathcal{A}_1(1^\lambda, \mathsf{crs})$
- $(C_P, (\bar{H}_P, \bar{r_1})) \leftarrow \mathsf{stSNARG.TrustHash}(\mathsf{crs}, P)$
- $((x, y)(c.\mathsf{com}, \Pi.\mathsf{com}.\mathsf{NIZK}.\Pi)) \leftarrow \mathcal{A}_2(\mathsf{crs}, P, C_P, \bar{H}_P, \bar{r_1}, \mathsf{aux})$
- if $\mathsf{ZKstSNARG.V}(\mathsf{crs}, (x, y), \mathsf{POut} = C_P, c.\mathsf{com}, \Pi.\mathsf{com}, \mathsf{NIZK}.\Pi) = 1 \wedge (P, x, y, T, \mathsf{POut}, \mathsf{crs}) \notin \mathcal{L}_{\mathcal{TM}}$, return 1
- else return 0

Lemma 6.3. *Assuming CRS indistinguishability of* $\mathsf{Com}_{\mathsf{bind}}$, $|Pr[\mathsf{G}_1^{\mathcal{A}} = 1] - \Pr[\mathsf{G}_2^{\mathcal{A}} = 1]| \leq \mathsf{negl}(\lambda)$.

One can define two intermediate hybrids by changing one $\mathsf{Com}_{\mathsf{bind}}$ key generation algorithm at a time. The proof then is straightforward from that of the previous lemma.

Hybrid Game G_3

- $\mathsf{crs}_1 \leftarrow \mathsf{stSNARG.Setup}(1^\lambda, 1^T)$
- $\mathsf{Com}_{\mathsf{bind}}.Key_1 \leftarrow \mathsf{Com.TGen}(1^\lambda)$
- $\mathsf{Com}_{\mathsf{bind}}.Key_2 \leftarrow \mathsf{Com.TGen}(1^\lambda)$
- $\mathsf{Com}_{\mathsf{bind}}.Key_3 \leftarrow \mathsf{Com.TGen}(1^\lambda)$
- $\mathsf{Com}_{\mathsf{bind}}.Key_4 \leftarrow \mathsf{Com.TGen}(1^\lambda), r_4 \leftarrow\!\!\$\ \{0,1\}^\lambda, z \leftarrow \mathsf{Com.C}(\mathsf{Com}.Key_4, 0; r_4)$
- $\mathsf{NIZK.crs} \leftarrow \mathsf{NIZK.Gen}(1^\lambda)$
- $\mathsf{crs} := (\mathsf{crs}_1, \mathsf{Com}_{\mathsf{bind}}.Key_1.\mathsf{Com}_{\mathsf{bind}}.Key_2, \mathsf{Com}_{\mathsf{bind}}.Key_3, z, \mathsf{NIZK.crs})$
- $(P, \mathsf{aux}) \leftarrow \mathcal{A}_1(1^\lambda, \mathsf{crs})$
- $(C_P, (\bar{H}_P, \bar{r_1})) \leftarrow \mathsf{stSNARG.TrustHash}(\mathsf{crs}, P)$
- $((x, y)(c.\mathsf{com}, \Pi.\mathsf{com}.\mathsf{NIZK}.\Pi)) \leftarrow \mathcal{A}_2(\mathsf{crs}, P, C_P, \bar{H}_P, \bar{r_1}, \mathsf{aux})$
- $\hat{c} \leftarrow \mathsf{Com.Ext}(\mathsf{Com}_{\mathsf{bind}}.Key_2, c.\mathsf{com})$
- $\hat{\Pi} \leftarrow \mathsf{Com.Ext}(\mathsf{Com}_{\mathsf{bind}}.Key_3, \Pi.\mathsf{com})$
- if $\mathsf{ZKstSNARG.V}(\mathsf{crs}, (x, y), \mathsf{POut} = C_P, c.\mathsf{com}, \Pi.\mathsf{com}, \mathsf{NIZK}.\Pi) = 1 \wedge (P, x, y, T, \mathsf{POut}, \mathsf{crs}) \notin \mathcal{L}_{\mathcal{TM}}$, return 1

– else return 0

Lemma 6.4. $Pr[\mathsf{G}_2^{\mathcal{A}} = 1] = \Pr[\mathsf{G}_3^{\mathcal{A}} = 1]$.

This lemma is obvious from the fact that the additional terms extracted have never been used in the hybrids.

Lemma 6.5. *Assuming extraction correctness/statistical binding property of* $\mathsf{Com}_{\mathsf{bind}}$*, and soundness of* stSNARG *we have that,*

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

Proof. For the sake of contradiction, let us say this is not the case. In other words, $(P, x, y, T, \mathsf{POut}, \mathsf{SOut}, \mathsf{crs}) \notin \mathcal{L}_{\mathcal{TM}}$ but $\mathsf{NIZK.V}(\mathsf{NIZK.crs}, (c.\mathsf{com}, \Pi.\mathsf{com}, (\mathsf{crs}, x, y, T), C_P), \mathsf{NIZK.}\Pi) = 1$. Assuming extraction correctness/binding property of $\mathsf{Com}_{\mathsf{bind}}$, we construct an adversary $\mathcal{B}$ which breaks the soundness of stSNARG as follows:

Adversary $\mathcal{B}$ playing stSNARG soundness game

– $\mathsf{crs}_1 \leftarrow \mathsf{stSNARG.Setup}(1^\lambda, 1^T)$
– $\mathsf{Com}_{\mathsf{bind}}.Key_1 \leftarrow \mathsf{Com.TGen}(1^\lambda)$
– $\mathsf{Com}_{\mathsf{bind}}.Key_2 \leftarrow \mathsf{Com.TGen}(1^\lambda)$
– $\mathsf{Com}_{\mathsf{bind}}.Key_3 \leftarrow \mathsf{Com.TGen}(1^\lambda)$
– $\mathsf{Com}_{\mathsf{bind}}.Key_4 \leftarrow \mathsf{Com.TGen}(1^\lambda), r_4 \leftarrow\!\!\$\ \{0,1\}^\lambda, z \leftarrow \mathsf{Com.C}(\mathsf{Com}.Key_4, 0; r_4)$
– $\mathsf{NIZK.crs} \leftarrow \mathsf{NIZK.Gen}(1^\lambda)$
– $\mathsf{crs} := (\mathsf{crs}_1, \mathsf{Com}_{\mathsf{bind}}.Key_1.\mathsf{Com}_{\mathsf{bind}}.Key_2, \mathsf{Com}_{\mathsf{bind}}.Key_3, z, \mathsf{NIZK.crs})$
– $(P, \mathsf{aux}) \leftarrow \mathcal{A}_1(1^\lambda, \mathsf{crs})$
– $(C_P, (\bar{H_P}, \bar{r_1})) \leftarrow \mathsf{stSNARG.TrustHash}(\mathsf{crs}, P)$
– $((x, y)(c.\mathsf{com}, \Pi.\mathsf{com.NIZK.}\Pi)) \leftarrow \mathcal{A}_2(\mathsf{crs}, P, C_P, \bar{H_P}, \bar{r_1}, \mathsf{aux})$
– $\hat{c} \leftarrow \mathsf{Com.Ext}(\mathsf{Com}_{\mathsf{bind}}.Key_2, c.\mathsf{com})$
– $\hat{\Pi} \leftarrow \mathsf{Com.Ext}(\mathsf{Com}_{\mathsf{bind}}.Key_3, \Pi.\mathsf{com})$
– return $\hat{c}, \hat{\Pi}$

We begin by pointing out that since crs contains a commitment to 0, a prover cannot produce a witness to show that crs has a commitment to 1 as it would violate the binding nature of $\mathsf{Com}_{\mathsf{bind}}$. Our assumption that the NIZK proof verifies implies that the prover produced witness $(r_1, r_2, r_3, c, \Pi, H_P)$ which were witnesses to the NP language $\mathcal{L}$ for the instance $(c.\mathsf{com}.\Pi.\mathsf{com}, (\mathsf{crs}, x, y, T), C_P)$. The extraction correctness of $\mathsf{Com}_{\mathsf{bind}}$ ensures it must be that $H_P = \bar{H_P}$. Thus, the prover indeed started the Turing Machine $\mathcal{TM}$ with the correct input $(\bar{H_P})$ in the second tape. This, along with the definition of C'_{index} implies that $\mathsf{stSNARGV}(\mathsf{crs}, ((x, y), T, \bar{H_P}), (c, \Pi)) = 1$ but, as per assumption, $\mathcal{TM}$ does not accept P, x, y in T steps. Furthermore, the extraction correctness $\mathsf{Com}_{\mathsf{bind}}$ also tells that $\hat{c} = c, \hat{\Pi} = \Pi$. Thus, $\mathsf{stSNARGV}(\mathsf{crs}, ((x, y), T, \bar{H_P}), (\hat{c}, \hat{\Pi})) = 1$ which clearly contradicts the soundness assumption of stSNARG. □

Combining these lemmas gives us a clear contradiction to our initial assumption that the ZKstSNARG scheme is not sound.

Non Interactive Zero Knowledge. For all $(P, x, y, T, \mathsf{POut}, \mathsf{crs}) \in \mathcal{L}_{\mathcal{TM}}$, there exists a PPT simulator $\mathsf{Sim} := (\mathsf{Sim}_1, \mathsf{Sim}_2, \mathsf{Sim}_3)$ such that the distributions of

$$(\mathsf{crs}, x, y, \mathsf{POut}, \Pi) \Big| (\mathsf{crs}, \mathsf{aux}) \leftarrow \mathsf{Sim}_1(1^\lambda, 1^T), (\mathsf{POut}, \mathsf{aux}') \leftarrow \mathsf{Sim}_2(\mathsf{crs}, \mathsf{aux}), \Pi \leftarrow \mathsf{Sim}_3(\mathsf{aux}', \mathsf{crs}, (x, y), \mathsf{POut})$$

Protocol 3 (NIZK Simulator $\mathsf{Sim} \coloneqq (\mathsf{Sim}_1, \mathsf{Sim}_2, \mathsf{Sim}_3)$)**.**

- $\mathsf{Sim}_1(1^\lambda, 1^T)$:
  1. $\mathsf{SE}.K_{\mathsf{even}} \leftarrow \mathsf{SE.Gen}(1^\lambda, 1^{M_{\lambda,T}}, 1^{L_\lambda})$
  2. $\mathsf{SE}.K_{\mathsf{odd}} \leftarrow \mathsf{SE.Gen}(1^\lambda, 1^M, 1^L)$
  3. $\mathsf{BARG.crs} \leftarrow \mathsf{BARG.Gen}(1^\lambda, 1^{T+1}, 1^{|C_{\mathsf{index}}|})$
  4. $\mathsf{dk} \leftarrow \mathsf{HT.Gen}(1^\lambda)$
  5. $\mathsf{Com}_{\mathsf{bind}}.Key_1 \leftarrow \mathsf{Com.Gen}(1^\lambda)$
  6. $\mathsf{Com}_{\mathsf{bind}}.Key_2 \leftarrow \mathsf{Com.Gen}(1^\lambda)$
  7. $\mathsf{Com}_{\mathsf{bind}}.Key_3 \leftarrow \mathsf{Com.Gen}(1^\lambda)$
  8. $\mathsf{Com}_{\mathsf{bind}}.Key_4 \leftarrow \mathsf{Com.Gen}(1^\lambda), r_4 \leftarrow\!\!\$\ \mathbb{N}, z \leftarrow \mathsf{Com.C}(\mathsf{Com}.Key_4, 1; r_4)$
  9. $\mathsf{NIZK.crs} \leftarrow \mathsf{NIZK.Gen}(1^\lambda)$
  10. return $\mathsf{crs} \coloneqq (\mathsf{SE}.K_{\mathsf{even}}, \mathsf{SE}.K_{\mathsf{odd}}, \mathsf{BARG.crs}, \mathsf{dk}, \mathsf{Com}_{\mathsf{bind}}.Key_1.\mathsf{Com}_{\mathsf{bind}}.Key_2, \mathsf{Com}_{\mathsf{bind}}.Key_3, z, \mathsf{NIZK.crs})$ and $\mathsf{aux} \coloneqq r_4$
- $\mathsf{Sim}_2(\mathsf{crs}, \mathsf{aux})$:
  1. $r_1 \leftarrow\!\!\$\ \{0,1\}^\lambda$, $C_P \leftarrow \mathsf{Com.C}(\mathsf{Com}_{\mathsf{bind}}.Key_1, 0; r_1)$ return $\mathsf{POut} \coloneqq C_P$.
  2. return $(\mathsf{POut}, \mathsf{aux}' \coloneqq \mathsf{aux})$
- $\mathsf{Sim}_3(\mathsf{crs}, \mathsf{aux}', (x, y), \mathsf{POut} \coloneqq C_P)$:
  1. $r_2 \leftarrow\!\!\$\ \{0,1\}^\lambda, c.\mathsf{com} \leftarrow \mathsf{Com.C}(\mathsf{Com}_{\mathsf{bind}}.Key_2, 0; r_2)$
  2. Generate a dummy proof $\hat{\Pi}$
  3. $r_3 \leftarrow\!\!\$\ \{0,1\}^\lambda, \Pi.\mathsf{com} \leftarrow \mathsf{Com.C}(\mathsf{Com}_{\mathsf{bind}}.Key_3, 0; r_3)$
  4. $\mathsf{NIZK}.\Pi \leftarrow \mathsf{NIZK.Prove}\left(\mathsf{NIZK.crs}, (c.\mathsf{com}, \Pi.\mathsf{com}, (\mathsf{crs}, x, y, T), C_P), (\perp, \perp, \perp, \mathsf{aux})\right)$
  5. return $(c.\mathsf{com}, \Pi.\mathsf{com}, \mathsf{NIZK}.\Pi)$.

Figure 7: Non Interactive Zero Knowledge Simulator

and

$$(\mathsf{crs}, x, y, \mathsf{POut}, \Pi) \Big| \begin{array}{l} \mathsf{crs} \leftarrow \mathsf{ZKstSNARG.Setup}(1^\lambda, 1^T), \\ (\mathsf{POut}, \mathsf{SOut}) \leftarrow \mathsf{ZKstSNARG.TrustHash}(\mathsf{crs}, P), \\ \Pi \leftarrow \mathsf{ZKstSNARG.P}(\mathsf{crs}, P, x, y, \mathsf{POut}, \mathsf{SOut}) \end{array}$$

are indistinguishable. For notational simplicity we denote these distributions by hyb_0 and hyb_1 respectively

- We define a game G' which is identical to G_0 except that crs has a commitment of 1 instead of 0. Note that an honest prover does not make use of this section of the crs in its proof. Consider hyb' as the output distribution of intermediate G'. All other algorithms in G' remains identical as G_0. hyb_0 must be indistinguishable from hyb', otherwise we can construct an efficient adversary that breaks the computational hiding property of $\mathsf{Com}_{\mathsf{bind}}$.
- The hybrid game G'' with output distribution hyb'' works like G' except stSNARG.P computes $(c.\mathsf{com}, \mathsf{NIZK}.\Pi)$ honestly and then ignores $c.\mathsf{com}$ and outputs $(c_1, \mathsf{NIZK}.\Pi)$ where c_1 is the statistical binding commitment to the 0 string using $\mathsf{Com}_{\mathsf{bind}}$. The indistinguishability of hyb' and hyb'' follows from the computational hiding property of $\mathsf{Com}_{\mathsf{bind}}$.
- We now define another hybrid game G''' where everything remains identical as G'' but the NIZK proof NIZK.P proves that crs has a commitment of 1 using randomness r as a witness. This is indeed a valid

witness for the same language $\mathcal{L}^*_{\mathsf{hyb}}$. Observe that G'' and G''' have identical CRS. However, $\mathsf{NIZK.P}$ in each case uses different witnesses, namely r and $((c, r_{\mathsf{com}_2}), \Pi)$ respectively. Thus, the Witness Indistinguishability of NIZK implies indistinguishability of G'' and G'''.

- In the next hybrid G'''', trusted commitment generator is replaced by Sim_2 which on input crs simply outputs a hiding commitment to the 0 string. Note that the output of Sim_2 is not used anywhere else in the proof and its output is identically distibuted to the public output of $\mathsf{ZKstSNARG.TrustHash}(\mathsf{crs}, P)$ because of the hiding property of commitment scheme.
- In the final game G_1, Sim_1 uses the same crs as the previous hybrid. Sim_3 ignores all operations performed by the prover and only outputs c_1 which is the statistical binding commitment to the 0 string using $\mathsf{Com}_{\mathsf{bind}}$ and sends a NIZK proof as G''''. The output distributions of G'''' and G_1 are indeed identical as the output of Sim_3 solely depends on the output of $\mathsf{Sim}_1, \mathsf{Sim}_2$ and the commitment of the 0 string c_1.

Combining all the hybrids, we prove that G_0 and G_1 have output distributions which are computationally indistinguishable.

Public Verifiable Non Interactive Succinct Delegation with Zero Knowledge A direct extension of Lemma 5.19 gives us the following corollary,

Corollary 6.6. *Assuming $T = \mathsf{poly}(m, n)$, $T, m, n \leq 2^\lambda$, the $\mathsf{ZKstSNARG}$ protocol in Figure 5 implies the unconditional existence of a publicly verifiable non interactive succinct delegation scheme with zero knowledge.*

The zero knowledge simulator for the delegation scheme $\mathsf{zk-sDel.Sim} := (\mathsf{zk-sDel.Sim}_1, \mathsf{zk-sDel.Sim}_2)$ can simply run the $\mathsf{stSNARG}$ ZK-simulator. More specifically, $\mathsf{zk-sDel.Sim}_1$ and $\mathsf{zk-sDel.Sim}_2$ call Sim_1 and Sim_2 respectively from Figure 7. The proof follows in a straightforward manner, hence we skip the details.

7 Acknowledgements

This research was supported in part from a Simons Investigator Award, DARPA SIEVE award, NTT Research, NSF Frontier Award 1413955, BSF grant 2012378, a Xerox Faculty Research Award, a Google Faculty Research Award, and an Okawa Foundation Research Grant. This material is based upon work supported by the Defense Advanced Research Projects Agency through Award HR00112020024.

8 References

[BCC+17] Nir Bitansky, Ran Canetti, Alessandro Chiesa, Shafi Goldwasser, Huijia Lin, Aviad Rubinstein, and Eran Tromer. The hunting of the snark. *Journal of Cryptology*, 30(4):989–1066, 2017.

[BCCT13] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for SNARKS and proof-carrying data. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 111–120. ACM Press, June 2013.

[BCI+13] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Rafail Ostrovsky, and Omer Paneth. Succinct non-interactive arguments via linear interactive proofs. In Amit Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 315–333. Springer, Heidelberg, March 2013.

[BHK17] Zvika Brakerski, Justin Holmgren, and Yael Tauman Kalai. Non-interactive delegation and batch NP verification from standard computational assumptions. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *49th ACM STOC*, pages 474–482. ACM Press, June 2017.

[BK20] Zvika Brakerski and Yael Kalai. Witness indistinguishability for any single-round argument with applications to access control. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020, Part II*, volume 12111 of *LNCS*, pages 97–123. Springer, Heidelberg, May 2020.

[BKK+18] Saikrishna Badrinarayanan, Yael Tauman Kalai, Dakshita Khurana, Amit Sahai, and Daniel Wichs. Succinct delegation for low-space non-deterministic computation. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *50th ACM STOC*, pages 709–721. ACM Press, June 2018.

[BKP18] Nir Bitansky, Yael Tauman Kalai, and Omer Paneth. Multi-collision resistance: a paradigm for keyless hash functions. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *50th ACM STOC*, pages 671–684. ACM Press, June 2018.

[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 Moses Charikar and Edith Cohen, editors, *51st ACM STOC*, pages 1082–1090. ACM Press, June 2019.

[CJJ21a] Arka Rai Choudhuri, Abhishek Jain, and Zhengzhong Jin. Non-interactive batch arguments for NP from standard assumptions. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part IV*, volume 12828 of *LNCS*, pages 394–423, Virtual Event, August 2021. Springer, Heidelberg.

[CJJ21b] Arka Rai Choudhuri, Abhishek Jain, and Zhengzhong Jin. Snargs for $\mathcal{P}$ from *lwe*. *Cryptology ePrint Archive*, 2021.

[CKV10] Kai-Min Chung, Yael Kalai, and Salil P. Vadhan. Improved delegation of computation using fully homomorphic encryption. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 483–501. Springer, Heidelberg, August 2010.

[CMT12] Graham Cormode, Michael Mitzenmacher, and Justin Thaler. Practical verified computation with streaming interactive proofs. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, pages 90–112, 2012.

[DFH12] Ivan Damgård, Sebastian Faust, and Carmit Hazay. Secure two-party computation with low communication. In Ronald Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 54–74. Springer, Heidelberg, March 2012.

[GGP10] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 465–482. Springer, Heidelberg, August 2010.

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 626–645. Springer, Heidelberg, May 2013.

[GKR15] Shafi Goldwasser, Yael Tauman Kalai, and Guy N Rothblum. Delegating computation: interactive proofs for muggles. *Journal of the ACM (JACM)*, 62(4):1–64, 2015.

[Gro10] Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 321–340. Springer, Heidelberg, December 2010.

[HLR21] Justin Holmgren, Alex Lombardi, and Ron D Rothblum. Fiat–shamir via list-recoverable codes (or: parallel repetition of gmw is not zero-knowledge). In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 750–760, 2021.

[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.

[Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments. In *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, pages 723–732, 1992.

[KP16] Yael Tauman Kalai and Omer Paneth. Delegating RAM computations. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part II*, volume 9986 of *LNCS*, pages 91–118. Springer, Heidelberg, October / November 2016.

[KPY18] Yael Kalai, Omer Paneth, and Lisa Yang. On publicly verifiable delegation from standard assumptions. *Cryptology ePrint Archive*, 2018.

[KPY19] Yael Tauman Kalai, Omer Paneth, and Lisa Yang. How to delegate computations publicly. In Moses Charikar and Edith Cohen, editors, *51st ACM STOC*, pages 1115–1124. ACM Press, June 2019.

[KRR13] Yael Tauman Kalai, Ran Raz, and Ron D. Rothblum. Delegation for bounded space. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 565–574. ACM Press, June 2013.

[KRR14] Yael Tauman Kalai, Ran Raz, and Ron D. Rothblum. How to delegate computations: the power of no-signaling proofs. In David B. Shmoys, editor, *46th ACM STOC*, pages 485–494. ACM Press, May / June 2014.

[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.

[Lip12] Helger Lipmaa. Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments. In Ronald Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 169–189. Springer, Heidelberg, March 2012.

[LS19] Alex Lombardi and Luke Schaeffer. A note on key agreement and non-interactive commitments. *Cryptology ePrint Archive*, 2019.

[Mer88] Ralph C. Merkle. A digital signature based on a conventional encryption function. In Carl Pomerance, editor, *CRYPTO'87*, volume 293 of *LNCS*, pages 369–378. Springer, Heidelberg, August 1988.

[Mic00] Silvio Micali. Computationally sound proofs. *SIAM Journal on Computing*, 30(4):1253–1298, 2000.

[PHGR13] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *2013 IEEE Symposium on Security and Privacy*, pages 238–252. IEEE, 2013.

[PR17] Omer Paneth and Guy N. Rothblum. On zero-testable homomorphic encryption and publicly verifiable non-interactive arguments. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part II*, volume 10678 of *LNCS*, pages 283–315. Springer, Heidelberg, November 2017.

[PRV12] Bryan Parno, Mariana Raykova, and Vinod Vaikuntanathan. How to delegate and verify in public: Verifiable computation from attribute-based encryption. In Ronald Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 422–439. Springer, Heidelberg, March 2012.

[PS19] Chris Peikert and Sina Shiehian. Noninteractive zero knowledge for NP from (plain) learning with errors. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part I*, volume 11692 of *LNCS*, pages 89–114. Springer, Heidelberg, August 2019.

[RRR19] Omer Reingold, Guy N Rothblum, and Ron D Rothblum. Constant-round interactive proofs for delegating computation. *SIAM Journal on Computing*, 50(3):STOC16–255, 2019.

[SMBW12] Srinath TV Setty, Richard McPherson, Andrew J Blumberg, and Michael Walfish. Making argument systems for outsourced computation practical (sometimes). In *NDSS*, volume 1, page 17, 2012.

[SVP+12] Srinath Setty, Victor Vu, Nikhil Panpalia, Benjamin Braun, Andrew J Blumberg, and Michael Walfish. Taking {Proof-Based} verified computation a few steps closer to practicality. In *21st USENIX Security Symposium (USENIX Security 12)*, pages 253–268, 2012.

[TKRR13] Yael Tauman Kalai, Ran Raz, and Ron D Rothblum. Delegation for bounded space. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 565–574, 2013.

[TRMP12] Justin Thaler, Mike Roberts, Michael Mitzenmacher, and Hanspeter Pfister. {Verifiable} computation with massively parallel interactive proofs. In *4th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 12)*, 2012.