Black-Box Non-Black-Box Zero Knowledge

Vipul Goyal
Microsoft Research
INDIA
vipul@microsoft.com

Rafail Ostrovsky
UCLA
USA
rafail@cs.ucla.edu

Alessandra Scafuro
UCLA
USA
scafuro@cs.ucla.edu

Ivan Visconti
University of Salerno
ITALY
visconti@unisa.it

Abstract

Motivated by theoretical and practical interest, the challenging task of designing cryptographic protocols having only black-box access to primitives has generated various breakthroughs in the last decade. Despite such positive results, even though nowadays we know black-box constructions for secure two-party and multi-party computation even in constant rounds, there still are in Cryptography several constructions that critically require non-black-box use of primitives in order to securely realize some fundamental tasks. As such, the study of the gap between black-box and non-black-box constructions still includes major open questions.

In this work we make progress towards filling the above gap. We consider the case of black-box constructions for computations requiring that even the *size* of the input of a player remains hidden. We show how to commit to a string of arbitrary size and to prove statements over the bits of the string. Both the commitment and the proof are succinct, hide the input size and use standard primitives in a black-box way. We achieve such a result by giving a black-box construction of an *extendable* Merkle tree that relies on a novel use of the "MPC in the head" paradigm of Ishai et al. [STOC 2007].

We show the power of our new techniques by giving the first black-box constant-round public-coin zero knowledge argument for NP.

To achieve this result we use the non-black-box simulation technique introduced by Barak [FOCS 2001], the PCP of Proximity introduced by Ben-Sasson et al. [STOC 2004], together with a black-box public-coin witness indistinguishable universal argument that we construct along the way.

Additionally we show the first black-box construction of a generalization of zero-knowledge sets introduced by Micali et al. [FOCS 2003]. The generalization that we propose is a strengthening that requires both the size of the set and the size of the elements of the set to remain private.

1 Introduction

A fundamental question in Cryptography of both theoretical and practical interest is whether a task can be securely realized by using primitives only in a black-box (BB) way. The question is of interest for theoreticians because, despite recent advances, there still are several fundamental tasks (e.g., CCA encryption, non-interactive zero-knowledge) that so far have been realized only

by critically using in a non-black-box (NBB) fashion some underlying primitives. For such tasks it is still unknown whether BB constructions are possible at all. Moreover, recently proposed BB constructions include ingenious and elegant techniques that make the study of the above gap fascinating and challenging. The question is also of practical interest since BB constructions avoid NP reductions involving circuits of primitives. Such reductions are often very expensive and categorize the NBB constructions as mere feasibility results. Another advantage of BB constructions is that one can instantiate the underlying primitive with an arbitrary implementation, for example even with a physical implementation (i.e., hardware tokens). As we shall see later, one can also replace a collision-resistant (CR) hash function with a random oracle.

Related work. The seminal work of Impagliazzo and Rudich [IR89] studied relations amongst cryptographic primitives with respect to BB constructions. While for secure signature schemes and commitment schemes, BB constructions based on any one-way function (OWF) were already known long time ago [Rom90, Nao91], the case of interactive protocols remained obscure until some recent breakthroughs.

In [IKLP06], Ishai et al. showed the first BB construction for oblivious transfer (OT) and then for multi-party computation (MPC), by pairing their result with the one of Kilian [Kil88]. Later on in [Hai08], Haitner showed how to get oblivious transfer through BB calls to semi-honest oblivious transfer, therefore improving on the generality of the underlying primitives.

Even though the existence of BB constructions for MPC might look like the end of the story, the state of affairs is instead much more complicated. Indeed, things change completely when fundamental properties of cryptographic protocols (e.g., round complexity, security in a concurrent setting, complexity of the players, underlying assumptions) are taken into account. Constant-round constructions were considered in [PW09], where Pass and Wee showed BB constructions based on OWFs for various building blocks such as trapdoor commitments, coin tossing and zero-knowledge (ZK) arguments of knowledge. The above building blocks combined with previous work produced the first constant-round constructions for two-party computation starting from any constant-round semi-honest OT protocol. In [CDSMW09], Choi et al. showed how to obtain similar results w.r.t. adaptive adversaries. In [Wee10], Wee showed the first BB constructions with sub-linear round complexity for MPC, and later on Goyal in [Goy11] obtained constant-round constructions based on the BB use of any OWF. In [GLOV12] the BB use of OWFs has been shown to be sufficient to construct constant-round concurrent non-malleable commitments. Other BB constructions for commitment schemes have been considered w.r.t. selective opening attacks in [Xia11, ORSV13]. In [LP12] Lin and Pass showed the first BB construction for MPC in the standard model that satisfies a non-trivial form of concurrent security and requires a non-constant number of rounds. Very recently, Kiyoshima et al. in [KMO14] improved on the round complexity providing a constant-round construction for the same result.

The tough case of hiding the input size. While the above results are encouraging towards avoiding NBB constructions, all the known techniques fail spectacularly when a cryptographic task aims at hiding the size of the input used during the computation. Such a requirement is for instance critical in constructions of witness indistinguishable universal arguments [BG02, BG08] (WIUARGs) and in turn in the breakthrough of Barak [Bar01] that shows how to get constant-round public-coin ZK, by means of new NBB simulation technique.

Input-size hiding constructions typically rely on the use of Merkle trees[1] to succinctly commit to an a priori *unbounded* number of elements, and to later reveal only selected elements. The key requirement is that in order to protect the size of the input, the length of a path between the root and a leaf of the tree *must* remain hidden. This requirement invalidates any straightforward black-box solution based on combining a CR hash function and a black-box commitment scheme, as it would require to unfold a path and therefore reveal the size of the tree.

Difficulties on achieving BB input-size hiding protocols become more evident when even more sophisticated known techniques fail. The reason is again that they crucially reveal the size of the input. Consider for instance one of the most powerful techniques developed so far, namely: the "MPC in the head" paradigm of Ishai et al. [IKOS07], subsequently extended in [GLOV12]. The use of this technique would require the virtual execution of an n-party protocol where players perform a computation over the committed input. It can be trivially observed by using information-theoretic arguments that since when implementing this paradigm some views of those virtual players are shown to the verifier, the size of the views of those player would reveal the size of the input.

In a concurrent and independent work, Ishai and Weiss [IW14] show a commit-and-prove scheme based on the MPC-in-the head technique that is succinct and makes only BB use of a hash function. Their construction is not size-hiding: their techniques crucially rely on the fact that the size of the input is known to all players.

This leads us to the first question.: "Is it possible to have black-box constructions of input-size hiding proofs"?

The Black-box non-black-box question. The NBB simulation paradigm of Barak has significantly influenced the landscape of cryptographic protocols as it yields results that are impossible via BB simulation (e.g., constant-round public-coin ZK). Despite being very influential, the downside is the established folklore that protocols proved secure through NBB simulation *must* be inefficient. This is in part due to the heavy NBB usage of cryptographic primitives. Does this mean that the constructions of Barak along with all other subsequent similar constructions would *forever remain only in papers* and can not move towards being realized in practice? If indeed this is true, it would be quite unfortunate. Getting a BB construction of a cryptographic protocol is generally considered to be the "first step" towards understanding how efficiently it can be implemented in practice.

This leads us to the second question. "Is it possible to have NBB simulation even without the protocol making NBB use of cryptographic primitives?", and therefore, "is there a BB construction for constant-round public-coin ZK?"

We remark that recent results based on non-black-box simulation techniques [BP12b, BP12a, BP13, CPS13, COPV13, COP+14] managed to obtain some important results on security under reset attacks without relying on Barak's approach, but their constructions are not public coin.

1.1 Our Results and Techniques

In this work we solve the above open problems by introducing new techniques to obtain BB input-size hiding proofs. Such proofs do not reveal anything about the size of the witnesses and use cryptographic primitives in a BB manner. We construct a BB size-hiding commit-and-prove protocol

[1] Since this work focuses on standard security notions, we will not consider the so called knowledge of exponent assumption [Dam91] and its generalizations as they are non-falsifiable [Nao03] (and therefore non-standard) assumptions.

that we use to implement the first BB construction of a WIUARG and of a constant-round public-coin ZK argument. Moreover, we provide a BB implementations of a generalized version of ZK sets.

More details on our techniques and applications follow below.

Black-Box input-size hiding commit-and-prove. We put forth the notion of a *black-box input-size hiding commit-and-prove* protocol, in which there is a prover P that commits to an arbitrarily long string, and later proves a predicate about this string so that: (1) both the commitment and the proof are succinct *and* hide the input size (note that a proof can be succinct but not size-hiding, e.g., UARG of [Kil92]); (2) the primitives are accessed in a BB manner.

Constructing a succinct BB commitment can be easily done via a Merkle tree. A Merkle tree is built by arranging the bits of the string on the leaves of a binary tree, and computing each internal node as the hash of the children. The root of such tree is of fixed size, independent of the size of the tree. The commitment then is simply a BB commitment of the root.

Achieving succinct and *size-hiding* BB proofs is instead much more problematic. The crucial problem is to use the hash function in a BB manner while guaranteeing input-size hiding. To see why, consider a prover who wants to prove a predicate about a bit of the committed string (hence, about a leaf of the Merkle tree). Soundness demands that first the prover proves that the leaf is consistent with the committed root. Consistency means that there exists a path from the leaf to the committed root, and each node along the path corresponds to the hash of its children. How to prove to the verifier that a path is consistent, without using the code of the hash function? The only way we can think of doing this, is by exhibiting a path and letting the verifier check the hash consistency of each node in such path. Neglecting for a moment how to reveal a path while keeping the values of the nodes along the path secret (which is only slightly less challenging), the main problem here is that the length of the path itself reveals the size of the string! Using an upper-bound on the size of the tree is not a solution, as in order to enable the prover to commit to any polynomially long string, such upper bound should be super-polynomial.

We solve the problem by introducing *extendable* Merkle trees, in which the prover can extend any real path to an arbitrary long imaginary path, *on-the-fly*. The main idea behind our extendable Merkle tree is a new representation of the nodes and their connection. A node is split in **two parts:** the usual label, that similarly to the standard Merkle tree, is computed as the hash of the children; and a redundant *representation of the label*, which is used to generate the next level of the tree. The consistency between the two parts of a node (namely, the label and its representation) is proved via a BB witness indistinguishable (WI) proof. This proof introduces a *soft link* between the nodes that can be broken with the right witness. The main gain from this new representation is twofold: (1) the redundant representation of the label allows the prover to reveal only parts of a node and let the verifier check the hash consistency of such parts on its own; (2) it allows the honest prover to arbitrary extend a path by cheating in the BB WI proof (when some conditions are satistied).

Concretely, we implement this idea using VSS (Verifiable Secret Sharing [BOGW88]) and MPC-in-the-head [IKOS07]. Namely, the redundant representation of the label is implemented using VSS: each label is represented as a vector of n imaginary VSS shares. The tree is built so that a label is the hash of the VSS *shares* of the children (instead of the hash of the labels of the children).

The BB WI proof is implemented using MPC-in-the-head on top of the VSS shares [GLOV12]. Due to the privacy of VSS the prover can reveal parts of the label, and the verifier can compute the hash on such shares on its own. Due to the correctness of MPC-in-the-head, the verifier is convinced

of the soundness of the BB WI proof, again by only looking at few views.

For convenience, in the rest of the paper we shall write size-hiding instead of input-size hiding.

Black-box WIUARGs. Witness Indistinguishable Universal Arguments (for short, WIUARGs) [BG08] are interactive arguments that allow to prove statements for languages in $\mathsf{NTIME}(\mathtt{t})$ (the tuple $(M, x, \mathtt{t}) \in \mathsf{NTIME}(\mathtt{t})$ if $M(x)$ outputs 1 in at most $\mathtt{t}$ steps) with the verification time which is polylogarithmic in $\mathtt{t}$. Because the verification must be polylogarithmic in $\mathtt{t}$, the proof itself must be succinct, namely, polylogarithmic in the size of the witness of maximum length. We follow the implementation of Barak and Goldreich [BG08] where a prover commits to a PCP oracle, and after receiving the queries from the verifier, it proves that, had it opened the bits of the oracle selected by the PCP queries, the PCP Verifier would have accepted.

Our BB commit-and-prove protocol directly yields a BB constant-round public-coin WIUARG: in the commitment phase, the prover commits to the PCP π; in the proof phase, the verifier provides the positions of π to be checked, and then the prover proves the predicate: "the PCP verifier would have accepted the bits of π selected by the verifier's queries". This result is presented in App. D.

Black-box constant-round public-coin ZK. As a main result of our work, we show a black-box construction for constant-round public-coin ZK, following the ideas proposed by Barak in [Bar01]. Barak's ZK protocol for an NP language L consists of two phases. In the first phase, called the trapdoor generation phase, the prover commits to a trapdoor. The trapdoor is a string which is supposed to be the code of a machine predicting the next message function of the verifier. This trapdoor is used by the non-black-box simulator who knows the code of the malicious verifier. The verifier replies to the commitment with a random string r.

In the second phase, called proof phase, P must prove that: (Theorem 1) either the value committed in the first phase is a Turing Machine M that predicts r in less then $|r|^{\log^2 n}$ steps (which happens only with negligible probability without knowledge of the code of V), or (Theorem 2) $x \in L$. There are two observations about Theorem 1.

First, Theorem 1 is not an NP statement, and as such must be proved using a WIUARG. Second, the theorem itself uses the code of the commitment scheme and of the hash function. The latter observation implies that even the improved implementations of Barak's protocol given in [PR08][2] still require both prover and verifier to access the code of the hash function.

As mentioned above, we know how to construct the black-box WIUARG that we can use to prove Theorem 1. Also, we know how to get a BB version of Theorem 1, by computing the commitment of M using our BB size-hiding commitment stage (via extendable Merkle trees). However, the two tools together are not enough to obtain a size-hiding proof for Theorem 1, for the following reason.

The machine M is the theorem on which a PCP proof π is computed. Although it is true that the PCP verifier can decide to accept a proof by reading only few bits of π, it still needs to read the *entire* theorem M in order to compute this decision. Therefore the view of a PCP verifier - which will be run in the head of the prover - depends on the size of the theorem being proved, and this immediately invalidates the zero-knowledge property. To solve this problem we use PCPs of Proximity instead of standard PCPs. In a PCP of Proximity the verifier has only oracle access to

[2] In such implementation, the honest prover is not required to compute a PCP since it can just use the witness corresponding to x in order to prove that either $x \in L$ or he sent a committed transcript of an accepting UA. Still both prover and verifier use the code of the hash function in their computations. We remark that also in our construction the prover is not required to actually compute a PCP proof.

the theorem, and thus is able to make its decision by looking at few bits of the theorem and few bits of the proof. This result is presented in Sec. 4.2.

Black-box public-coin protocols and the Random Oracle Model. Our black-box use of a CR hash function allows us to instantiate the hash function with a random oracle (our constructions also use statistically-hiding commitments, but they can be constructed in a BB manner from any CR hash function [NY89]). Therefore, in the random oracle model (ROM) [BR93] we obtain the first information-theoretically secure WIUARG. Applying the Fiat-Shamir heuristic [FS86], we obtain the first *non-interactive* information-theoretically secure WIUARG in the ROM. Note that CS proofs of [Mic94] are non-interactive UARGs but they are not witness indistinguishable. Similarly, our public-coin ZK protocol can be made non-interactive and information-theoretically secure in the ROM. Interestingly, this seems to be the first example of an information-theoretically secure protocol in the ROM for which the Fiat-Shamir heuristic would fail.

We remark that the soundness of such information-theoretic succinct proofs is still "computational" in the sense that a false proof can exist and can be found by a prover that has unbounded access to the random oracle.

1.1.1 Generalized BB Input-Size Hiding Proofs of Set Membership

We use some ideas from the extendable Merkle tree to construct *generalized* BB input-size hiding proofs of set membership. In such proofs there is a prover that commits to a sparse dataset, and a verifier that is allowed to make set membership queries. The commitment and the proofs must not reveal any information about the set except the membership of the element. In particular, the proof should not even leak the size of elements that the dataset supports. Namely, suppose that the largest element contained in the dataset is k-bit long. Our definition requires that even k is hidden to the verifier. This is a natural generalization of ZK sets introduced by Micali et el. [MRK03], where the prover instead has to declare the upper bound k in advance.

Since in generalized proofs of set membership a verifier can query an element $i \in 2^{poly(n)}$ for any polynomial *poly* chosen by the verifier, with the extendable Merkle tree in hands, one can think that this problem is easily solved for any element described by a k'-bit string with $k' > k$. Indeed, for such elements the prover can just extend the path on the fly, and provide a proof of non-membership in the set. Unfortunately, this is not true for elements of size i which are shorter than k. The reason is that in the standard way of building Merkle's trees, all the elements are arranged in the leaves, therefore at level k. Thus, providing any proof about an index of size $i < k$ requires to open a path of at least k nodes. Consequently, the size of the path depends on the upperbounded size of an element of the set of the prover. We solve this problem by building the Merkle tree in a new way. Namely, we arrange the elements so that, an element of size i lies on the i-th level of the tree. In this way, the length of a proof depends only on the size of the element queried and not on a fixed upperbound. We give our definition and construction of generalized BB input-size hiding proofs of set membership in App. F.

2 Definitions

In this section we provide informal definitions of some of the tools that we use in our constructions. We give more details and other definitions in App. A.

Probabilistically Checkable Proofs. Informally, a PCP system for a language L consists of a proof π written in a redundant form for a statement "$x \in L$", and a PPT verifier, which is able to decide the truthfulness of the statement by reading only few bits of the proof.

A PCP verifier V can be decomposed into a pair of algorithms: the query algorithm $\mathsf{Q}_{\mathsf{pcp}}$ and the decision algorithm $\mathsf{D}_{\mathsf{pcp}}$. $\mathsf{Q}_{\mathsf{pcp}}$ on input x and random tape r, outputs positions $q_1 = \mathsf{Q}_{\mathsf{pcp}}(x, r, 1)$, $\ldots$, $q_{p(|x|)} = \mathsf{Q}_{\mathsf{pcp}}(x, r, p(|x|))$ for some polynomial p. V accepts if $\mathsf{D}_{\mathsf{pcp}}(x, r, \pi[q_1], \ldots, \pi[q_{p(|x|)}])$ outputs 1. The formal definition of a PCP is provided in App. A.5.

For later, it is useful to see algorithm $\mathsf{D}_{\mathsf{pcp}}(\cdot)$ as a predicate defined over a string π which is tested on few positions.

Probabilistically Checkable Proofs of Proximity. The standard PCP verifier decides whether to accept the statement "$x \in L$" by probing few bits of the proof π and reading the entire statement x.

A "PCP of proximity" (PCPP) [BSGH+06] is a relaxation of PCP in which the verifier is able to make a decision without even reading the entire statement, but only few bits of it. More specifically, in a PCPP the theorem is divided in two parts (a, y). A public string a, which is read entirely by the verifier, a private string y, of which the verifier has only *oracle* access. Consequently, PCPP is defined for pair languages $L \subseteq \{0,1\}^* \times \{0,1\}^*$. For every $a \in \{0,1\}^*$ we denote $L_a = \{y \in \{0,1\}^* : (a, y) \in L\}$.

The soundness requirement of PCPP is relaxed in the sense that V can only verify that the input is *close* to an element of the language. The PCP Verifier can be seen as a pair of algorithms $(\mathsf{Q}_{\mathsf{pcpx}}, \mathsf{D}_{\mathsf{pcpx}})$, where $\mathsf{Q}_{\mathsf{pcpx}}(a, r, i)$ outputs a pair of positions (q_i, p_i): q_i denotes a position in the theorem y, p_i denotes a position in the proof π. $\mathsf{D}_{\mathsf{pcpx}}$ decides whether to accept (a, y) by looking at the public theorem a, and at positions $y[q_i]$, $\pi[p_i]$. More details on PCPP are provided in App. A.5.1.

For later, it is useful to see algorithm $\mathsf{D}_{\mathsf{pcpx}}(\cdot)$ as a predicate defined over two strings y, π, testing few positions of each string.

Definition 1 (PCPP verifier)**.** *For functions $s, \delta : \mathbb{N} \to [0,1]$, a verifier V is a probabilistically checkable proof of proximity (PCPP) system for a pair language L with proximity parameter δ and soundness error s, if the following two conditions hold for every pair of strings (a, y):*

- *Completeness: If $(a, y) \in L$ then there exists π such that $V(a)$ accepts oracle $y \circ \pi$ with probability 1.*
- *Soundness: If y is $\delta(|a|)$-far from L_a, then for every π, the verifier $V(a)$ accepts oracle $y \circ \pi$ with probability strictly less than $s(|a|)$. Formally:*

$$\forall \pi \;\; \mathrm{Prob}_{(\mathsf{Q}_{\mathsf{pcpx}}, \mathsf{D}_{\mathsf{pcpx}}) \leftarrow V(a)}[\mathsf{D}_{\mathsf{pcpx}}((y \circ \pi)|_{\mathsf{Q}_{\mathsf{pcpx}}})] = 1 < s(|a|).$$

The query complexity of the verifier depends only on the public input a.

Secure Multiparty Computation. A secure multi-party computation (MPC) [BOGW88, AL11] scheme allows n players to jointly, privately and correctly compute an n-ary function based on their private inputs, even in the presence of t corrupted players. [BOGW88] shows that for every n-ary function $\mathcal{F} : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$, there exists a t-secure MPC protocol $\mathsf{MPC}\text{-}\mathcal{F}$ that securely computes $\mathcal{F}$ in the semi-honest model for any $t < n/2$, and in the malicious model for any $t < n/3$,

with perfect completeness and security. That is, given the private input w_i of player P_i, after running the protocol $\mathsf{MPC}\text{-}\mathcal{F}$, an honest P_i receives in output the i-th component of the result of the function $\mathcal{F}$ applied to the inputs of the players, as long as the adversary corrupts less than t players. In addition, nothing is learnt by the adversary from the execution of $\mathsf{MPC}\text{-}\mathcal{F}$ other than the outputs of corrupted players.

In an MPC protocol, the view of a player includes all messages received by that player during the execution of the protocol, and the private inputs and the randomness used by the player. The views of two players are consistent if they satisfy the following definition.

Definition 2 (View Consistency)**.** *The view of an honest player during an MPC computation contains input and randomness used in the computation, and all messages received/sent from/to the communication tapes. We have that two views $(view_i, view_j)$ are consistent with each other if, (a) both the players P_i and P_j individually computed each outgoing message honestly by using the random tapes, inputs and incoming messages specified in $view_i$ and $view_j$ respectively, and, (b) all output messages of P_i to P_j appearing in $view_i$ are consistent with incoming messages of P_j received from P_i appearing in $view_j$, and vice versa.*

Verifiable Secret Sharing (VSS). A verifiable secret sharing (VSS) [CGMA85] scheme is a two-stage secret sharing protocol for implementing the following functionality. In the first stage, denoted by $\texttt{Share}(s)$, a special player referred to as dealer, shares a secret s among n players, in the presence of at most t corrupted players. In the second stage, denoted by $\texttt{Recon}$, players exchange their views of the share stage, and reconstruct the value s. We use notation $\texttt{Recon}(P_i^{\mathsf{VSS}}, \ldots, P_n^{\mathsf{VSS}})$ to refer to this procedure. The functionality ensures that when the dealer is honest, before the second stage begins, the t corrupted players have no information about the secret. Moreover, when the dealer is dishonest, at the end of the share phase the honest players would have realized it through an accusation mechanism that disqualifies the dealer.

A VSS scheme can tolerate errors on malicious dealer and players on distributing inconsistent or incorrect shares, indeed the critical property is that even in case the dealer is dishonest but has not been disqualified, still the second stage always reconstructs the same string among the honest players. In this paper, we use a (n, t)-perfectly secure VSS scheme with a deterministic reconstruction procedure [GIKR01]. The formal definition of VSS and MPC are provided in A.6.

MPC-in-the-head. MPC-in-the-head is a breakthrough technique introduced by Ishai at al. in [IKOS07] to construct a BB zero-knowledge protocol. Let $\mathcal{F}_{ZK}$ be the zero-knowledge functionality for an NP language L, that takes as public input x and one share from each player P_i, and outputs 1 iff the secret reconstructed from the shares is a valid witness. Let $\mathsf{MPC}\text{-}\mathsf{ZK}$ be a perfect t-secure MPC protocol implementing $\mathcal{F}_{ZK}$.

Very roughly, the "MPC-in-the-head" idea is the following. The prover runs *in his head* an execution of $\mathsf{MPC}\text{-}\mathsf{ZK}$ among n imaginary players, each one participating in the protocol with a share of the witness. Then it commits to the view of each player separately. The verifier obtains t randomly chosen views, and checks that such views are consistent (according to Def. 2) and accepts if the output of every player is 1. Clearly P^* decides the randomness and the input of each player so it can cheat at any point and make players output 1. However, the crucial observation is that in order to do so, it must be the case that a constant fraction of the views committed are not consistent. Thus by selecting the t views at random, V will catch inconsistent views whp.

One can extend this technique further (as in [GLOV12]), to prove a general predicate ϕ about arbitrary values. Namely, one can consider the functionality $\mathcal{F}_\phi$ in which every player i participates with input a VSS share P_i^{vss}. $\mathcal{F}_\phi$ collects all such shares, and outputs 1 if and only if $\phi(\mathtt{Recon}(P_1^{\mathsf{vss}}, \ldots, P_n^{\mathsf{vss}})) = 1$. We crucially use this idea in our constructions.

3 Size-Hiding Commit-and-Prove

We now define the primitive *size-hiding BB commit-and-prove*. This is a two-stage functionality parameterized by an *upper bound* $d = n^{\log n}$ on the string size, which captures any polynomially long string. In the first stage, the prover P commits to a string s. In the second stage, called proof stage, the verifier V challenges P with a set of positions I in the range $[d]$ and P proves that a predicate ϕ is satisfied in those positions. Because the real size of the string is unknown to V, V will choose one set of positions for each possible size: $I_1, \ldots, I_{\ell_d}$, with $\ell_d = \log d$.

The soundness requirement is that V accepts the proof iff ϕ is satisfied in the set of position $I_{\log |s|}$ where s is the unique string committed in the first stage. The privacy requirement is defined via the witness indistinguishability property: for every predicate ϕ, for every pair of strings w_0, w_1 of possibly *different* sizes which satisfy ϕ, at the end of the proof phase, any malicious verifier cannot distinguish which of the two strings was committed. The formal definition is provided in App. A.1.

For simplicity of explanation, in the following we assume that V queries a single position q_j, instead of a set of positions I_j, for each $j \in \{1, \ldots, \ell_d\}$.

3.1 Warm-Up: A Non-Black-Box Approach

Achieving a black-box construction for the above primitive requires several ideas and techniques. We start by showing a non-black-box construction which explains the rationale behind the final protocol. Then we show how to make this protocol black-box.

Merkle tree. Merkle trees are used to succinctly commit to an arbitrarily long string and to later selectively open single bits of the string. Given a string s and a CR hash function $h : \{0,1\}^{2n} \to \{0,1\}^n$, let $\ell_s = \log(|s|)$. The Merkle tree is a binary tree constructed as follows.

- A leaf l^γ is set as the γ-th bit of s, where $\gamma \in \{0,1\}^{\ell_s}$.
- An internal node l^γ is set as $h(l^{\gamma 0} | l^{\gamma 1})$, where $\gamma \in \cup_{i=\ell_s-1}^{i=0} \{0,1\}^i$. We denote by l^λ the root of the tree.

We shall refer to l^γ as the *label* of the node.

Commitment. The commitment of s is the root of the tree. *Opening.* To reveal a bit in position $\gamma \in \{0,1\}^{\ell_s}$, the prover sends s_γ together with an authentication path consisting of the nodes from the leaf γ to the root, and their siblings. Formally, for a leaf $\gamma \in \{0,1\}^{\ell_s}$, the authentication path consists of the labels $\mathsf{path}_\gamma = ((l^0, l^1), (l^{\gamma_1 0}, l^{\gamma_1 1}), \ldots, (l^{\gamma_{\ell_s-1} 0}, l^{\gamma_{\ell_s-1} 1}))$; where $\gamma_1, \ldots, \gamma_{\ell_s}$ is the bit expansion of index γ. *Verification.* V accepts s_γ if for any $i \in [\ell_s]$, $l^{\gamma_i} = h(l^{\gamma_i 0} | l^{\gamma_i 1})$.

We start by showing a straightforward commit-and-prove protocol based on Merkle trees which is succinct, but is *not size-hiding* and uses commitment scheme and hash function in a non-black-box manner.

- Commitment. P constructs the Merkle tree for a string $s \in \{0,1\}^*$. P sends the root l^λ and the depth of the tree ℓ_s to V.
- V sends predicate ϕ to be proved, and the index $q \in \{0,1\}^{\ell_s}$.
- Proof. On input (ϕ, q), P sends commitment of the authentication path for s_q, namely $\mathsf{cp} = \mathsf{com}(\mathsf{path}_q)$. Then it proves in zero-knowledge that: 1) (**path consistency**) the path committed in cp is consistent with l^λ and opens to a value s_q. Namely, there exists a valid opening of the commitment cp to a path path_q for a leaf l^q that is consistent with the root l^λ *under the hash function* h. This proof uses the code com and h. 2) (**predicate**) $\phi(l^q) = 1$.

Size-hiding **NBB commit-and-prove.** In the above construction P reveals the size ℓ_s of the committed string. ℓ_s is used by V to compute the index q on which the predicate should be evaluated. We now show a construction that is also size-hiding. Recall that the size-hiding construction works with the upper bound $d = n^{\log n}$, and that $\ell_d = \log d$.

The first change is that now the verifier will query an index q for each possible level of the tree representing the string: $q_1, \ldots, q_{\ell_d}$. Consequently, the prover is expected to provide a path for each query q_j. However, the prover has built a real tree of depth ℓ_s, and does not know how to open consistent paths which are longer than ℓ_s (note that P cannot build a tree of depth ℓ_d as it would be of exponential size). Additionally, for any $j \neq \ell_s$, the prover cannot prove that the predicate ϕ is true.

The observation here is that the prover is not really required to do so. The only thing we require from the prover is to provide a consistent path for the query q_{ℓ_s} (the only query that hits the real depth) and that $\phi(s_{q_{\ell_s}}) = 1$. For any other query q_j, with $j \neq \ell_s$, the prover should not give any proof.

This turns out to be very easy to achieve using the code of the commitment scheme and of the hash function in the proof. The idea is to use the depth of the real tree as a **trapdoor**, and allow the prover to cheat when answering to any query outside the real depth.

Thus, we require P to commit to the depth ℓ_s of the real tree (the trapdoor) already in the commitment phase. This binds P to a tree of depth ℓ_s and enables P to prove only predicates about leaves lying at level ℓ_s. In the proof phase, P answers to each query that does not hit the real tree by computing a fake path and then computing the ZK proof using as witness the trapdoor, that is, the depth of the tree. More in details, the size-hiding NBB commit-and-prove is the following.

- Commitment. P constructs the Merkle tree for a string $s \in \{0,1\}^*$. P sends $\mathsf{cr} = l^\lambda$ and $\mathsf{cd} = \mathsf{com}(\ell_\mathsf{s})$ to V.
- Challenge. V sends predicate ϕ to be proved, and indexes $q_1, \ldots, q_{\ell_d}$ with $q_j \in \{0,1\}^j$ and $j = 1, \ldots, \ell_d$.
- Proof. (**Commitment of paths**) For $j \neq \ell_s$, P set path_{q_j} to be all zeros and $\mathsf{cp}_{q_j} = \mathsf{com}(\mathsf{path}_{\mathsf{q_j}})$. For q_{ℓ_s}, P sets $\mathsf{path}_{q_{\ell_s}}$ to be the real path from a leaf labelled as $l^{q_{\ell_s}}$ to the root committed in cr. (**Proof of consistency**) For any j, P proves in ZK that either 1) cp_{q_j} is the commitment of a path from leaf l^{q_j} consistent to the root committed in cr, *under hash function* h and $j = \ell_s$ and $\phi(l^{q_j}) = 1$, or 2) $j \neq \ell_s$.

The NBB use of the primitives allows to cheat straightforwardly: for the queries that are not hitting the real tree, the prover can just make up fake paths and use the trapdoor condition to

compute the ZK proof. In fact, *the prover is doing all the checks in his head*, using the code of the primitives. The verifier only sees commitments and ZK proofs therefore cannot distinguish which is the real path.

The difficulty comes when the prover cannot prove statements about committed values and hashed values using the code of such primitives. In particular, how can P prove the consistency of the path with the root without using the code of the hash function?

3.2 Our Black-Box Protocol

The main difficulty when moving to a BB construction is how to prove consistency of a path without revealing any information about the real tree, and without using the code of the hash function and the commitment scheme.

In order to use the hash function in a BB manner, the prover should unfold each path and let the verifier check its consistency by directly applying the hash function. This is clearly problematic for two reasons: (1) for any query lying beyond the real tree, the prover has not constructed any path, hence for such queries the consistency check always fails, thus revealing the size of the real tree; (2) the prover should reveal a path without revealing the values of the nodes. This might require the prover to work with committed values, therefore requiring the ZK proof to use the code of the commitment.

3.2.1 The High-Level Idea

Before going into the details of the construction, we describe the main idea that allows us to finally implement a size-hiding commit-and-prove protocol in a fully black-box manner. Our idea is to construct the tree differently.

A standard Merkle tree is a chain of hash values. Any node is directly connected with its children and directly connected with its parent via hash.

Our idea is to break this chain by splitting the node in two parts, one part is connected with the children only, we call this *label* part, one part is connected with the parent only, let us call this *meta-label* part. The meta-label is just a redundant representation of the label, namely, given the meta-label one can uniquely reconstruct the label. Obviously a node is consistent if the two parts are connected, namely, if it holds that the meta-label and the label express exactly the same information. The consistency of a node can be proved by revealing both the label and the meta-label, and let the verifier compute and verify the reconstruction. However, looking ahead we do not want to reveal any information about the labels. Therefore, we let the prover prove this consistency via black-box ZK proof[3]. In order to allow BB ZK proof, we have to choose the format of the meta-label appropriately.

Thus every triple (label, left child, right child) can be seen as a LEGO (see Fig. 1). Inside a LEGO there is an hard connection between the label and the children, as the label is the hash of the (meta-label of the) children. However, between two LEGOs there is only a virtual connection, namely two LEGOs are connected iff the meta-label is consistent with the label.

Now, to make the path extendable it is sufficient to break the virtual connection between label and meta-label when necessary. In such cases, the prover/simulator will possess a witness that allows him to cheat in the proof of consistency.

[3]Actually a witness indistinguishable proof.

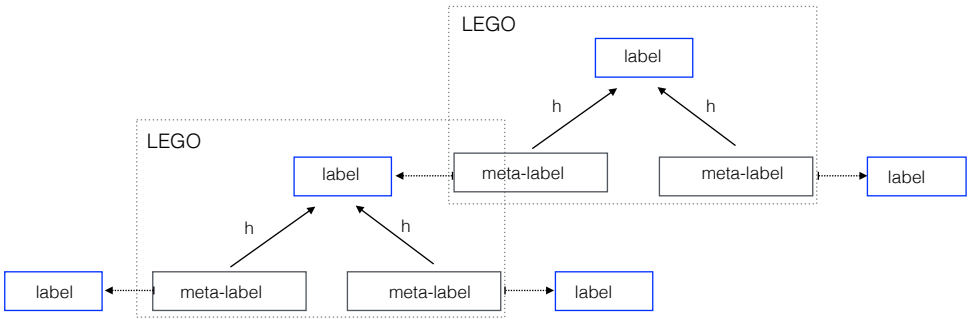


Figure 1: High-level idea behind an extendable Merkle Tree.

The final ingredient that allows to provide a BB proof of consistency, is to represent *any* value that will be involved in the proof using the same representation used for the meta-label.

3.2.2 The implementation

We now provide an implementation of the above high-level idea.

In a nutshell, the representation that we use for the *meta-label* is VSS. The BB ZK proof to prove consistency of a node (i.e., to prove that a label is uniquely reconstructed from its *meta-label*), is implemented using MPC-in-the-head on top of VSS [GLOV12].

Meta-label representation. We represent any value s involved in the proof in a redundant manner, namely, as a vector of views of VSS players. More specifically, let s be any string that P will use in the proof. Instead of working directly with s, P will execute (in his head) a VSS protocol among $n+1$ players in which the dealer shares the string s. The result of this process is a vector of n views of VSS players, with the property that $(n-t)$ views allow the reconstruction of value s, but any subset of only t views does not reveal anything about s. We call this vector of views as the VSS representation of s.

Definition 3 (VSS representation)**.** *The VSS representation of a string $s \in \{0,1\}^*$, is the vector $\mathsf{VSS} = [P_1^{\mathsf{VSS}}, \ldots, P_n^{\mathsf{VSS}}]$, where P_i^{VSS} is the view of player i, participating in the protocol $\mathtt{Share}(s)$. In the paper we use VSS^γ to denote the VSS representation of the label l^γ, where γ is the index the node, or simply $\mathsf{VSS}^{\mathsf{string}}$ to denote the VSS representation of the string string.*

Following this approach, the prover computes the Merkle tree by "extending" each label with its VSS representation.

In our tree, a node in position γ is split in the pair $[l^\gamma, \mathsf{VSS}^\gamma]$, where l^γ is the label and VSS^γ is the vector of n views of VSS players which are secret sharing l^γ. We shall refer to l^γ as the *label* part of a node, and to VSS^γ as the VSS part. Each node $[l^\gamma, \mathsf{VSS}^\gamma]$ satisfies the condition that that $\mathtt{Recon}(\mathsf{VSS}^\gamma) = l^\gamma$. We call this constraint the "innode property".

Similarly, the value representing the depth of the real tree is replaced with its VSS representation, that we denote by $\mathsf{VSS}^{\mathsf{depth}}$.

The VSS representation allows us to prove the consistency of a path and the validity of the predicate using hash and commitment in a BB manner, for the following reasons. First, the t-privacy of VSS allows P to partially reveal a node and let the verifier check the hash consistency

with the parent. Indeed, P can reveal up to t views of the VSS part of each node, while still preserving the hiding of the *label*, and the verifier can compute the hash of such values (we explain this more formally later). Second, the VSS representation allows us to prove any predicate about the value reconstructed by the VSS views, in a BB manner using MPC-in-the-head: due to the t-correctness of VSS and the MPC protocol used, the verifier is convinced about the validity of the proof by looking at only t views of the VSS and the MPC.

More specifically, assume P wants to prove a predicate ϕ about a value s, and let $\mathsf{VSS} = [P_1^{\mathsf{VSS}}, \ldots, P_n^{\mathsf{VSS}}]$ be the VSS representation of s. P can run an MPC-in-the-head for a functionality F_ϕ played among n players. A player P_i gives in input a view P_i^{VSS}, F_ϕ reconstructs the value s from such views, and outputs 1 to all players iff $\phi(s) = 1$. The output of this process is again a vector of n views. Later, V will probe t views of the MPC and VSS, and is convinced if the views are consistent and the output in every revealed view is 1.

Remark 1. [On the need of VSS.] One might ask why we use the VSS representation instead of a simpler secret sharing algorithm, e.g., just xor of n string, or Shamir's secret sharing. Indeed, in both cases, revealing t views will not reveal any information about the secret. The reason is that such secret sharing schemes are not t robust: in the case of the xor, it is sufficient for P^* to cheat in only one view to change the secret shared, Shamir's secret sharing instead is not robust against a malicious dealer.

Node connection. In a standard Merkle tree, the label of a node is computed as the hash of the labels of the children. We connect the nodes differently. In our tree, the label of a parent node is computed as the hash of only the VSS part of the children, discarding the *label* part. The innode property guarantees that the label of the parent is still *virtually* connected to the *label* part of the children (see Fig. 2). This virtual connection introduces a *soft* link between labels, that can be broken when necessary.

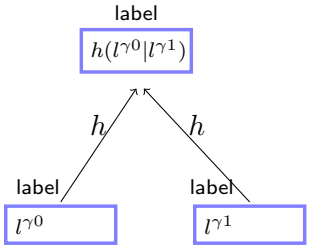
Definition 4 (Node Connection)**.** *Let* $\mathsf{VSS}^{\gamma 0} = [P_i^{\mathsf{VSS}^{\gamma 0}}]_{i\in[n]}$ *and* $\mathsf{VSS}^{\gamma 1} = [P_i^{\mathsf{VSS}^{\gamma 1}}]_{i\in[n]}$ *then the label for the parent node* γ *is the vector:* $l^\gamma = [h(P_1^{\mathsf{VSS}^{\gamma 0}}) \mid \cdots \mid h(P_n^{\mathsf{VSS}^{\gamma 0}}) \mid h(P_1^{\mathsf{VSS}^{\gamma 1}}) \mid \cdots \mid h(P_n^{\mathsf{VSS}^{\gamma 1}})]$*. Note that each VSS view is* hashed separately*.*

As there is no direct connection between labels $l^\gamma, l^{\gamma 0}, l^{\gamma 1}$, the consistency of a link in a tree is conditioned on the fact that a node is consistent, that is, $[l^{\gamma b}, \mathsf{VSS}^{\gamma b}]$ satisfies the innode property, for $b = 0, 1$.

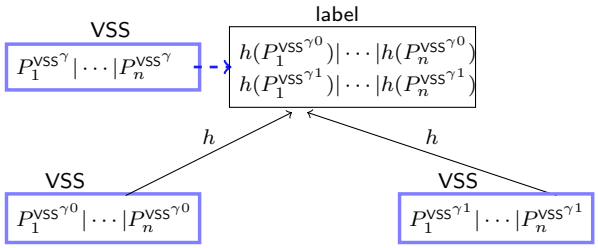
Two key observations follow. First, the innode condition is crucial for the tree to be binding. If the innode condition is not enforced, then P can replace the label of a node with a string which is the root of some freshly generated path. Second, the innode condition cannot be checked directly by V, as it would require V to see more than t views and reconstruct the value of each node. Instead, it is proved by P via MPC-in-the-head. Looking ahead, in this MPC, P will prove that either the innode condition holds, or the node is on a path of a query which is outside the real depth. For this proof P will need the VSS part of a node and the VSS representation of the depth of the real tree.

As we shall see later, these two facts together enable the prover to compute fake paths for the queries outside the real tree.

Summing up, for each level of our Merkle tree we have a two-tier connection. A hard connection between a parent label and the children: each label l^γ is the concatenation of the hash values of $\mathsf{VSS}^{\gamma 0}, \mathsf{VSS}^{\gamma 1}$. This connection is checked directly by V applying hash function upon t views



(a) A node in the Merkle Tree.



(b) A node in the Extendable Merkle Tree.

Figure 2: Standard Merkle Tree Vs. Extendable Merkle Tree

revealed (**hash consistency**). A soft connection, within a node $[l^\gamma, \mathsf{VSS}^\gamma]$ (**node consistency**). This is checked indirectly by the prover using MPC-in-the-head on top of VSS^γ and $\mathsf{VSS}^{\mathsf{depth}}$. P runs an MPC protocol for the functionality $\mathcal{F}_{\mathsf{innode}}$ that takes as input from each player i, the i-th view of VSS^γ, and $\mathsf{VSS}^{\mathsf{depth}}$ and the i-th and $(i+n)$-th hash values from l^γ, and outputs 1 to every player iff, either $\mathtt{Recon}(\mathsf{VSS}^\gamma) = l^\gamma$ or $\gamma > \mathtt{Recon}(\mathsf{VSS}^{\mathsf{depth}})$. We denote this protocol by $\mathsf{MPC\text{-}in}$.

Fig. 3 illustrates how our new Merkle tree is built, and the commitment stage of our protocol.

Black-box proof. We now have all the ingredients to describe how we obtain a proof phase which is size-hiding and completely black-box.

Recall that in the proof phase, V sends queries $q_1, \ldots, q_{\ell_d}$, one for each possible depth of the real tree. The prover is expected to **prepare paths** for each query, even for those that are outside the real tree, and **prove** that paths are consistent and that predicate ϕ is true for the query hitting the committed depth. For queries outside the real tree P computes fake paths for which it is not required to honestly prove consistency. In the non-black-box protocol, fake paths were just a sequence of zeros. This was enough because paths were never opened.

In our case, we will need the prover to partially open each node of the path, so that the verifier can directly check the hash consistency of each node. Therefore here the prover has to compute fake paths that still look consistent with the real tree. We are able to prepare such paths by exploiting the *soft* connection in our extendable Merkle tree.

Our black-box proof phase proceeds as follows. P **prepares paths.** For each query q_j, P retrieves the nodes along the path between q_j and the root. If q_j is below the real tree, P will just add fake nodes from a virtual leaf in position q_j up to the first node hitting the real tree, in a way that **hash consistency** is preserved. P commits to the paths so constructed. **Proof.** For the path for query q_j, P prepares the following proofs: **path consistency**, for each node of the path, P computes $\mathsf{MPC\text{-}in}$ using the trapdoor condition iff $j > \ell_s$; **predicate**: for the leaf q_j, P runs an

Commitment Stage: $\mathsf{BBCom}(\cdot)$
Input to P: String $s \in \{0,1\}^*$. Let $\ell_s = \log |s|$

V sends h to P.

P builds the *extendable* Merkle tree for s.

- Leaf: the leaf in position $\gamma \in \{0,1\}^{\ell_s}$ is the pair: $[l^\gamma, \mathsf{VSS}^\gamma]$. Where $l^\gamma = s_\gamma$ and $\mathsf{VSS}^\gamma = [P_1^{\mathsf{vss}^\gamma} | \ldots | P_n^{\mathsf{vss}^\gamma}] = \mathtt{Share}(l^\gamma)$.
- Internal Node: for $\gamma \in \cup_{i=\ell_s-1}^{i=0} \{0,1\}^i$, $l^\gamma = [h_1^{\gamma 0} | \ldots | h_n^{\gamma 1}] = [h(P_1^{\mathsf{vss}^{\gamma 0}}) | \ \ldots \ | h(P_n^{\mathsf{vss}^{\gamma 1}})]$ and $\mathsf{VSS}^\gamma = \mathtt{Share}(l^\gamma)$.
- Root: the label of the root is $l^\lambda = [h_1^0 | \ldots | h_n^1]$.

P runs $\mathtt{Share}(\ell_s)$ and obtains $[P_1^{\mathsf{depth}}, \ldots, P_n^{\mathsf{depth}}]$.

P commits to the root and to the depth of the real tree sending: $\bar{\mathsf{C}}_\lambda = [\mathsf{SHCom}(h_1^0) \mid \ldots \mid \mathsf{SHCom}(h_n^1)]$ and $\mathsf{C}_i = \mathsf{SHCom}(P_i^{\mathsf{depth}})$, for $i = 1, \ldots, n$.

P sends $\bar{\mathsf{C}}_\lambda$, C_1, $\ldots$, C_n to V.

Figure 3: Commitment via Extendable Merkle Tree

MPC-in-the-head on top of VSS^{q_j} and $\mathsf{VSS}^{\mathsf{depth}}$, to prove that: either $\phi(\mathtt{Recon}(\mathsf{VSS}^{q_j})) = 1$ and $j = \mathtt{Recon}(\mathsf{VSS}^{\mathsf{depth}})$, or the trapdoor condition holds: $j \neq \mathtt{Recon}(\mathsf{VSS}^{\mathsf{depth}})$. We denote this MPC as $\mathsf{MPC}\text{-}\phi$.

Now, observe that this approach can be generalized further. Namely, we can test more than one trapdoor condition on different strings, just by representing such strings as VSS. Looking ahead, in our ZK protocol, we allow the prover to cheat if he knows the witness. We ask the prover to commit to VSS^w, where w is the witness, and then in the MPC-in-the-head $\mathsf{MPC\text{-}in}$ and $\mathsf{MPC}\text{-}\phi$, we add a second trapdoor condition which is satisfied iff $\mathcal{R}_L(\mathtt{Recon}(\mathsf{VSS}^w), x) = 1$.

Verification. In order to verify the proof, V sends t positions $p_1, \ldots, p_t$ to P. For each node, P reveals the views in such positions. V is then able to verify hash consistency, node consistency and the predicate condition. The details of the proof phase are illustrated in Fig. 4.

Our size-hiding black-box commit-and-prove protocol is shown in Fig. 3 (Commitment) and Fig. 4 (Proof). For simplicity of exposition we have shown a construction that works with a single query q_j for each level of the virtual tree. The same protocol can be easily adapted to work with a set $I_j = \{q_{j1}, \ldots, q_{jk}\}$ of positions.

The commitment scheme SHCom used in the protocol must be statistically-hiding, as it is also secure under selective opening attacks (under the indistinguishability definition [Hof11]).

Observations. In the commitment stage of our protocol, P commits to the string s by committing to the root of the extendable Merkle tree, and to the depth ℓ_s, by committing to the VSS representation of ℓ_s. The depth is used as trapdoor condition. In the proof phase P is actually proving a predicate about both s and ℓ_s.

This approach can be generalized further. P can commit to several strings, committing either to their VSS representation (in case the size can be revealed) or to the root of an extendable Merkle tree. Then it can prove arbitrary relations among such strings just by setting the condition to be checked in protocols $\mathsf{MPC\text{-}in}$ and $\mathsf{MPC}\text{-}\phi$ accordingly. We use this ability of proving statements

Proof Stage: $\mathsf{BBProve}(\phi, (q_1, \ldots, q_d))$

P **computes a path for each query** q_j**.**

Case $j > \ell_s$**: queries lying below the real depth.** Compute a fake path from position j to position ℓ_s. Denote by q_j^i the first i bits of the string q_j. **Leaf.** Compute $\mathsf{VSS}^{q_j^{(j-1)b}} = \mathtt{Share}(0)$, for $b = 0, 1$. **Nodes.** For $i = j-1, \ldots, \ell_s$. If both children q_j^{i0}, q_j^{i1} exist, compute label $l^{q_j^i} = [h(P_1^{\mathsf{VSS}^{q_j^{i0}}}), \cdots, h(P_n^{\mathsf{VSS}^{q_j^{i1}}})]$. Else, first compute the VSS of the missing child $\mathsf{VSS}^{q^{ib}} = \mathtt{Share}(0)$ and proceed as above. For the remaining nodes, use the ones of the real tree.

For each fake node $[l^{q_j^i}, \mathsf{VSS}^{q_j^i}]$ run $\mathsf{MPC\text{-}in}$ using trapdoor condition $i \geq \mathtt{Recon}(\mathsf{VSS}^{\mathsf{depth}})$. Obtain views $\mathsf{MPC\text{-}in}^{q_j^i} = [P_1^{\mathsf{in}^{q_l^i}}, \ldots, P_n^{\mathsf{in}^{q_l^i}}]$. For the leaf q_j, run MPC-ϕ using as witness the condition $\mathtt{Recon}(\mathsf{VSS}^{\mathsf{depth}}) \neq j$. Obtain views MPC-$\phi^{q_j} = [P_1^{\phi^{q_i}}, \ldots, P_n^{\phi^{q_i}}]$.

Case $j = \ell_s$**: queries hitting the real depth.** Retrieve the real path from the root to leaf q_{ℓ_s}. For each node, compute $\mathsf{MPC\text{-}in}$ proving that innode property is satisfied. For the leaf q_{ℓ_s}, compute MPC-ϕ to prove that $\phi(\mathtt{Recon}(\mathsf{VSS}^{q_{\ell_s}}) = 1)$ **and** $\mathtt{Recon}(\mathsf{VSS}^{\mathsf{depth}}) = j$.

Case $j < \ell_s$**: queries lying on the real depth.** Retrieve the path from the real tree, from the root to leaf q_l. Compute proof of consistency and proof of predicate using the trapdoor condition $\mathtt{Recon}(\mathsf{VSS}^{\mathsf{depth}}) \neq j$.

Commitment of the paths. For $\gamma = q_1, \ldots, q_{\ell_d}$, P has obtained paths: $\mathsf{path}_\gamma = [l^0, \mathsf{VSS}^0, \mathsf{MPC\text{-}in}^0], [l^1, \mathsf{VSS}^1, \mathsf{MPC\text{-}in}^1], \ldots, [l^\gamma, \mathsf{VSS}^\gamma, \mathsf{MPC\text{-}in}^\gamma]$ and MPC-ϕ^γ. P commits to each node by committing to each view separately and hash value separately via SHCom.

Verification.

V sends to P randomly selected positions $p_1, \ldots, p_t$.

P opens views corresponding to players in position $p_1, \ldots, p_t$ for all the commitments above.

V performs the following checks:

Hash consistency. For each node i, check that $l^i[p_j] = h(P_{p_j}^{\mathsf{VSS}^{i0}})$ and $l^i[p_j + n] = h(P_{p_j}^{\mathsf{VSS}^{i1}})$ (for $j = 1, \ldots, t$).

Nodes consistency: innode property. For each node i, check that view $P_{p_j}^{\mathsf{in}^i}$ of $\mathsf{MPC\text{-}in}$ contains $l^i[p_j]$, $l^i[p_j + n]$, $P_{p_j}^{\mathsf{VSS}^i}$, $P_{p_j}^{\mathsf{depth}}$ as input, that it is consistent with the other views and the output is 1.

Predicate. Check the views $P_{p_j}^{\phi^{q_j}}$, for $j = 1, \ldots, \ell_d$ and $p_j = p_1, \ldots, p_t$, as in the previous step. If any of these checks fails, abort.

Figure 4: Proof Stage

over multiple strings committed using either VSS or extendable merkle tree in our public-coin ZK protocol.

4 Applications

In this section we show how our size-hiding commit-and-prove protocol can be used to obtain a WIUARG, and a public-coin BB ZK argument.

4.1 Black-Box WIUARG

A WIUARG for a language $\mathcal{L}_{\mathcal{U}}$ can be seen as a commit-and-prove protocol: the prover P commits to a PCP proof π, the PCP verifier V challenges P with the positions computed via $\mathsf{Q}_{\mathsf{pcp}}$, and P proves the predicate: "$\mathsf{D}_{\mathsf{pcp}}$ outputs 1 on the selected positions".

More formally, the predicate proved by P is denoted by $\phi\text{-}\mathsf{D}_{\mathsf{pcp}}(y, \cdot)$ and is defined over instances $y \in \mathcal{L}_{\mathcal{U}}$ and a set of indexes I, and it is true iff $\mathsf{D}_{\mathsf{pcp}}(y, \{\pi_i\}_{i \in I}) = 1$. The details of the construction are shown in Prot. 5. The commitment scheme used in Prot. 5 is an *extractable* SH commitment scheme (instead of just SH). This is required to obtain the weak-proof-of-knowledge property.

Protocol 1. *Black-Box Witness Indistinguishable Universal Argument.*
Common Input: $y = (M, x, \mathtt{t}) \in \mathcal{L}_{\mathcal{U}}$. *Let* $d = poly(\mathtt{t})$.
Auxiliary Input to P**:** w *such that* $(y, w) \in \mathcal{R}_{\mathcal{U}}$.

Commitment of the PCP.

1. P *runs* M *on input* (w, x). *Let* $\mathtt{t}' = T_M(x, w)$. *Let* $w' = (w, \mathtt{t}')$ *such that* $(y, w') \in \mathcal{R}'_{\mathcal{U}}$. P *invokes* $\mathsf{P}_{\mathsf{pcp}}$ *on input* (y, w') *and obtains the proof* π.
2. P *runs* $\mathsf{BBCom}(\pi)$ *(Fig. 3) to commit to* π.

Queries. V *sends random tapes* $r_1, \ldots, r_{\ell_d}$ *to* P. P *and* V *run* $I_j \leftarrow \{\mathsf{Q}_{\mathsf{pcp}}(y, r_j, i)\}_{i \in [k]}$ *for* $j \in [\ell_d]$. *They obtain queries* $\{I_1, \ldots, I_{\ell_d}\}$.

Proof. P *runs* $\mathsf{BBProve}(\phi\text{-}\mathsf{D}_{\mathsf{pcp}}, \{I_1, \ldots, I_{\ell_d}\})$ *(Fig. 4).* V *accepts iff the verifier of* $\mathsf{BBProve}$ *accepts.*

4.2 Black-box Constant-Round Public-Coin Zero Knowledge

We provide a black-box implementation of Barak's zero-knowledge protocol by extending our commit-and-prove protocol to several strings.

In the first stage of Barak's protocol, P starts by sending a commitment z, that is supposed to be a commitment of a machine M predicting the next message function of the verifier. The size of M cannot be upper bounded by any fixed polynomial, as one has to include any possible polynomial time machine. Therefore commitment z is computed using our size-hiding commitment of Fig. 3. The honest prover simply commit to the string 0^n. Next, V sends a random string r. This concludes the trapdoor generation phase, and defines the trapdoor theorem (M, r): "M is a TM that predicts r in at most $\mathtt{t}$ steps", which will be proved using PCP of Proximity. Here M is the private theorem, and $(r, \mathtt{t})$ is the public theorem.

In the second phase, P has to prove that either the trapdoor theorem is true or $x \in L$. It sends two commitments: a commitment to a string which is supposed to be a PCPP proof π for the trapdoor theorem, and a commitment to the witness for theorem "$x \in L$". The PCPP proof π is committed again via our size-hiding commitment stage (Fig. 3). For the witness w, P commits to its VSS representation. We denote this commitment by VSSCom. This concludes the commitment stage of the commit-and-prove protocol.

The verifier then generates the queries for the oracles M, π by running algorithm $\mathsf{Q}_{\mathsf{pcpx}}$ on input the public theorem $(r, \mathtt{t})$. $\mathsf{Q}_{\mathsf{pcpx}}$ outputs indexes q_i for private theorem M and indexes p_i for the proof π.

We now use the full power of our novel black-box size-hiding commit-and-prove protocol. We define the predicate ϕ-ZK to be checked over the strings M, π, w, as follows: ϕ-ZK is true iff the decision algorithm $\mathsf{D}_{\mathsf{pcpx}}$, executed on the selected positions of M and π outputs 1, or if w is a valid witness for $x \in L$. Needless to say, P will not commit to a valid PCPP proof, and will cheat in every proof using the trapdoor theorem $x \in L$.

Additionally, in protocol $\mathsf{MPC\text{-}in}$, we introduce another trapdoor condition that allows the prover to cheat in proving the consistency of the tree. The trapdoor condition is the knowledge of the witness for $x \in L$.

The formal specification of the protocol follows. We use PCP of Proximity with proximity parameter δ, for the following pair language: $\mathcal{L}_{\mathcal{P}} = \{(a = (r, \mathtt{t}), (Y)),\ \exists M \in \{0,1\}^*$ s.t. $Y \leftarrow \mathsf{ECC}(M),\ M(z) = r$ within $\mathtt{t}$ steps$\}$ (where $\mathsf{ECC}(\cdot)$ is a binary error correcting code tolerating a constant fraction $\delta > 0$ of errors).

Protocol 2. *Black-Box Constant-Round Public-coin ZK*
Common Input: x.
Auxiliary Input to P: *w such that $(x, w) \in \mathcal{R}_L$.*

Trapdoor-generation phase.

- *P runs $\mathsf{BBCom}(0^n)$ with V. Let z be the commitment so obtained.*
- *V sends a random value $r \in \{0,1\}^n$ to P. The public theorem a is defined as: $a = (r, \mathtt{t})$.*

Proof phase.

- Commitment of the PCPP. *P runs $\mathsf{BBCom}(0^n)$ and $\mathsf{VSSCom}(w)$ and sends the commitments to V.*
- PCPP Queries. *V sends random tapes $r_1, \ldots, r_{\ell_d}$ to P. P and V compute $(q_i^j, p_i^j) = \mathsf{Q}_{\mathsf{pcpx}}(a, r_j, i)$ with $i \in [k]$, where k is the security parameter for the PCPP. Let $I_j^M = \{q_1^j, \ldots, q_k^j\}$ and $I_j^\pi = \{p_1^j, \ldots, p_k^j\}$.*
- Proof. *P runs $\mathsf{BBProve}(\phi\text{-}\mathsf{ZK}, \mathcal{I}^M, \mathcal{I}^\pi)$. V accepts iff the verifier of $\mathsf{BBProve}$ accepts.*

5 Acknowledgment

We thank the anonymous reviewers for many very helpful comments. We thank Adam Smith and Rafael Pass for suggesting the connection with the Fiat-Shamir heuristic.
Work supported in part by NSF grants 09165174, 1065276, 1118126 and 1136174, US-Israel BSF grant 2008411, OKAWA Foundation Research Award, IBM Faculty Research Award, Xerox Faculty

Research Award, B. John Garrick Foundation Award, Teradata Research Award, and Lockheed-Martin Corporation Research Award. This material is based upon work supported by the Defense Advanced Research Projects Agency through the U.S. Office of Naval Research under Contract N00014 -11 -1-0392. The views expressed are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

References

[AL11] Gilad Asharov and Yehuda Lindell. A Full Proof of the BGW Protocol for Perfectly-Secure Multiparty Computation. *Electronic Colloquium on Computational Complexity (ECCC)*, 18:36, 2011.

[Bar01] Boaz Barak. How to go beyond the black-box simulation barrier. In *FOCS*, pages 106–115. IEEE Computer Society, 2001.

[BG02] Boaz Barak and Oded Goldreich. Universal arguments and their applications. In *Computational Complexity*, pages 162–171, 2002.

[BG08] Boaz Barak and Oded Goldreich. Universal arguments and their applications. *SIAM J. Comput.*, 38(5):1661–1694, 2008.

[BHY09] Mihir Bellare, Dennis Hofheinz, and Scott Yilek. Possibility and impossibility results for encryption and commitment secure under selective opening. In Antoine Joux, editor, *EUROCRYPT*, volume 5479 of *Lecture Notes in Computer Science*, pages 1–35. Springer, 2009.

[BOGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In Janos Simon, editor, *STOC*, pages 1–10. ACM, 1988.

[BP12a] Nir Bitansky and Omer Paneth. From the impossibility of obfuscation to a new non-black-box simulation technique. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 223–232. IEEE Computer Society, 2012.

[BP12b] Nir Bitansky and Omer Paneth. Point obfuscation and 3-round zero-knowledge. In *Theory of Cryptography - 9th Theory of Cryptography Conference, TCC 2012, Taormina, Sicily, Italy, March 19-21, 2012. Proceedings*, volume 7194 of *Lecture Notes in Computer Science*, pages 190–208. Springer, 2012.

[BP13] Nir Bitansky and Omer Paneth. On the impossibility of approximate obfuscation and applications to resettable cryptography. In *STOC*, 2013.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM Conference on Computer and Communications Security*, pages 62–73, 1993.

[BSCGT12] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, and Eran Tromer. On the concrete-efficiency threshold of probabilistically-checkable proofs. *Electronic Colloquium on Computational Complexity (ECCC)*, 19:45, 2012.

[BSGH+06] Eli Ben-Sasson, Oded Goldreich, Prahladh Harsha, Madhu Sudan, and Salil P. Vadhan. Robust pcps of proximity, shorter pcps, and applications to coding. *SIAM J. Comput.*, 36(4):889–974, 2006.

[CDD+99] Ronald Cramer, Ivan Damgrard, Stefan Dziembowski, Martin Hirt, and Tal Rabin. Efficient Multiparty Computations Secure Against an Adaptive Adversary. In *Advances in Cryptology — EUROCRYPT '99*, volume 1592 of *Lecture Notes in Computer Science*, pages 311–326. Springer, 1999.

[CDSMW09] Seung Geol Choi, Dana Dachman-Soled, Tal Malkin, and Hoeteck Wee. Simple, black-box constructions of adaptively secure protocols. In *Theory of Cryptography, 6th Theory of Cryptography Conference, TCC 2009, San Francisco, CA, USA, March 15-17, 2009. Proceedings*, volume 5444 of *Lecture Notes in Computer Science*, pages 387–402. Springer, 2009.

[CGMA85] Benny Chor, Shafi Goldwasser, Silvio Micali, and Baruch Awerbuch. Verifiable Secret Sharing and Achieving Simultaneity in the Presence of Faults (Extended Abstract). In *Proceedings of the 26th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '85, pages 383–395, 1985.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In John H. Reif, editor, *STOC*, pages 494–503. ACM, 2002.

[COP+14] Kai-Min Chung, Rafail Ostrovsky, Rafael Pass, Muthuramakrishnan Venkitasubramaniam, and Ivan Visconti. 4-round resettably-sound zero knowledge. In *TCC*, pages 192–216, 2014.

[COPV13] Kai-Min Chung, Rafail Ostrosky, Rafael Pass, and Ivan Visconti. Simultaneous resettability from one-way functions. In *FOCS*, 2013.

[CPS13] Kai-Min Chung, Rafael Pass, and Karn Seth. Non-black-box simulation from one-way functions and applications to resettable security. In *STOC*, 2013.

[Dam91] Ivan Damgrard. Towards practical public key systems secure against chosen ciphertext attacks. In Joan Feigenbaum, editor, *CRYPTO*, volume 576 of *Lecture Notes in Computer Science*, pages 445–456. Springer, 1991.

[DNRS03] Cynthia Dwork, Moni Naor, Omer Reingold, and Larry J. Stockmeyer. Magic functions. *J. ACM*, 50(6):852–921, 2003.

[DSK12] Dana Dachman-Soled and Yael Tauman Kalai. Securing circuits against constant-rate tampering. In *CRYPTO*, volume 7417 of *LNCS*, pages 533–551. Springer, 2012.

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO*, 1986.

[GIKR01] Rosario Gennaro, Yuval Ishai, Eyal Kushilevitz, and Tal Rabin. The Round Complexity of Verifiable Secret Sharing and Secure Multicast. In *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing*, STOC '01, pages 580–589. ACM, 2001.

[GLOV12] Vipul Goyal, Chen-Kuei Lee, Rafail Ostrovsky, and Ivan Visconti. Constructing non-malleable commitments: A black-box approach. In *FOCS*, pages 51–60. IEEE Computer Society, 2012.

[Goy11] Vipul Goyal. Constant round non-malleable protocols using one way functions. In Lance Fortnow and Salil P. Vadhan, editors, *STOC*, pages 695–704. ACM, 2011.

[Hai08] Iftach Haitner. Semi-honest to malicious oblivious transfer - the black-box way. In *Theory of Cryptography, Fifth Theory of Cryptography Conference, TCC 2008, New York, USA, March 19-21, 2008*, volume 4948 of *Lecture Notes in Computer Science*, pages 412–426. Springer, 2008.

[HM96] Shai Halevi and Silvio Micali. Practical and Provably-Secure Commitment Schemes from Collision-Free Hashing. In *Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '96, pages 201–215. Springer-Verlag, 1996.

[HNO+09] Iftach Haitner, Minh-Huyen Nguyen, Shien Jin Ong, Omer Reingold, and Salil P. Vadhan. Statistically Hiding Commitments and Statistical Zero-Knowledge Arguments from Any One-Way Function. *SIAM J. Comput.*, 39(3):1153–1218, 2009.

[Hof11] Dennis Hofheinz. Possibility and impossibility results for selective decommitments. *J. Cryptology*, 24(3):470–516, 2011.

[HR07] Iftach Haitner and Omer Reingold. Statistically-hiding commitment from any one-way function. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, STOC '07, pages 1–10. ACM, 2007.

[IKLP06] Yuval Ishai, Eyal Kushilevitz, Yehuda Lindell, and Erez Petrank. Black-box constructions for secure computation. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing, Seattle, WA, USA, May 21-23, 2006*, pages 99–108. ACM, 2006.

[IKOS07] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge from secure multiparty computation. In David S. Johnson and Uriel Feige, editors, *STOC*, pages 21–30. ACM, 2007.

[IR89] Russell Impagliazzo and Steven Rudich. Limits on the provable consequences of one-way permutations. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing, May 14-17, 1989, Seattle, Washigton, USA*, pages 44–61. ACM, 1989.

[IW14] Yuval Ishai and Mor Weiss. Probabilistically checkable proofs of proximity with zero-knowledge. In *TCC*, 2014.

[Kil88] Joe Kilian. Founding cryptography on oblivious transfer. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 20–31. ACM, 1988.

[Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In S. Rao Kosaraju, Mike Fellows, Avi Wigderson, and John A. Ellis, editors, *STOC*, pages 723–732. ACM, 1992.

[KMO14] Susumu Kiyoshima, Yoshifumi Manabe, and Tatsuaki Okamoto. Constant-round black-box construction of composable multi-party computation protocol. In *TCC*, pages 343–367, 2014.

[LP12] Huijia Lin and Rafael Pass. Black-box constructions of composable protocols without set-up. In Safavi-Naini and Canetti [SNC12], pages 461–478.

[Mic94] Silvio Micali. Cs proofs (extended abstracts). In *FOCS*, pages 436–453. IEEE Computer Society, 1994.

[MRK03] Silvio Micali, Michael O. Rabin, and Joe Kilian. Zero-knowledge sets. In *FOCS*, pages 80–91. IEEE Computer Society, 2003.

[Nao91] Moni Naor. Bit commitment using pseudorandomness. *J. Cryptology*, 4(2):151–158, 1991.

[Nao03] Moni Naor. On cryptographic assumptions and challenges. In Dan Boneh, editor, *CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 96–109. Springer, 2003.

[NY89] Moni Naor and Moti Yung. Universal one-way hash functions and their cryptographic applications. In *STOC*, 1989.

[ORSV13] Rafail Ostrovsky, Vanishree Rao, Alessandra Scafuro, and Ivan Visconti. Revisiting lower and upper bounds for selective decommitments. In *TCC*, pages 559–578, 2013.

[PR08] Rafael Pass and Alon Rosen. Concurrent nonmalleable commitments. *SIAM J. Comput.*, 37(6):1891–1925, 2008.

[PW09] Rafael Pass and Hoeteck Wee. Black-box constructions of two-party protocols from one-way functions. In Omer Reingold, editor, *TCC*, volume 5444 of *Lecture Notes in Computer Science*, pages 403–418. Springer, 2009.

[Rom90] John Rompel. One-way functions are necessary and sufficient for secure signatures. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA*, pages 387–394. ACM, 1990.

[SNC12] Reihaneh Safavi-Naini and Ran Canetti, editors. *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, volume 7417 of *Lecture Notes in Computer Science*. Springer, 2012.

[Wee10] Hoeteck Wee. Black-box, round-efficient secure computation via non-malleability amplification. In *FOCS*, pages 531–540. IEEE Computer Society, 2010.

[Xia11] David Xiao. (nearly) round-optimal black-box constructions of commitments secure against selective opening attacks. In Yuval Ishai, editor, *TCC*, volume 6597 of *Lecture Notes in Computer Science*, pages 541–558. Springer, 2011.

A Definitions

We assume familiarity with interactive arguments and argument of knowledge.

Notation. We denote the security parameter by n. A *negligible* function $\nu(n)$ is a non-negative function such that for any constant $c < 0$ and for all sufficiently large n, $\nu(n) < n^c$. We call a positive function *overwhelming* if it can be described as $1 - \nu(n)$ for some negligible function ν. We denote by $[n]$ the sequence $\{1, \ldots, n\}$ and by $poly(\cdot)$ a generic polynomial.

A.1 Input-Size Hiding Black-Box Proofs

Here we provide a formal definition of our size-hiding black-box proof systems.

Definition 5 (Input-Size hiding black-box commit-and-prove system[4])**.** *A pair of PPT algorithm $\langle P, V\rangle$ is a* **size-hiding black-box** *commitment and prove system, with parameters $d = n^{\log(n)}$, if it consists of two stages, commitment and proof stage. In the commitment stage, P interacts with V to commit to a string s of size $|s| \leq d$. In the proof phase, V probes a set of indexes $\mathcal{I} \subset \left(\{0,1\}^i\right)^\kappa$ with $i = 1, \ldots, \ell_d$ (with $\ell_d = \log d$). P then proves that for any subset of indexes $I \in \mathcal{I} \cap \{0,1\}^{\log(|s|)}$, $\phi(\{s_i\}_{i\in I})$ is true, for a predicate ϕ. $\langle P, V\rangle$ must satisfy the following conditions.*

Completeness. *If P and V are honest, than V accepts the proof with probability* 1*.*

Black-box. *Both P and V make only oracle calls to cryptographic primitives.*

Efficient Verification. *The time spent by V to verify the proof is $poly(|d|, \kappa)$.*

Witness Indistinguishability. *For any malicious verifier V^*, for any pair of strings w^0, w^1 (of possibly different size), for any pair of (possibly identical) sets $I_0, I_1 \in \mathcal{I}$ such that $\phi(\{w_i^0\}_{i\in I_0}) = \phi(\{w_i^1\}_{i\in I_1})$, where $|I_0| = |I_1| = \kappa$, for any auxiliary input z, the probability ensembles $\{\langle P(w^0), V^*(z)\rangle\}$ and $\{\langle P(w^1), V^*(z)\rangle\}$ are indistinguishable, where $\{\langle P(s), V^*(z)\}$ is the view of V^* at the end of the protocol.*

Soundness. *For any PPT malicious prover P^*, for a commitment transcript τ, except with negligible probability, two conditions must hold: 1) (binding) there exists at most one string s that P^* can use in the proof; 2) (*soundness) for any set $I \in \mathcal{I}$ chosen by V, P^* convinces the verifier iff $\phi(\{s_i\}_{i\in I})$ is true for $I \in \mathcal{I} \cap \{0,1\}^{\log(|s|)}$.

In the following definition we use the word *index* in place of element. Namely, we say that the verifier sends index i, to mean that the verifier wants to check if the element i is not empty in the database. This nomenclature is helpful for us when describing the implementation of BB input size-hiding proofs of membership using extendable Merkle Trees.

Definition 6 (Generalized Input-Size hiding black-box proof of set membership system)**.** *A database D is a map: $\{0,1\}^* \to \{0,1\}^* \cup \perp$ such that $D[i] \neq \perp$ for finitely many values. The domain of D is called* indexes *and the codomain is called* values*. We say that an index i is a YES-instance if $D[i] \neq \perp$. Otherwise we say that i is a NO-instance.*

A pair of PPT algorithm $\langle P, V\rangle$ is a **generalized size-hiding black-box** *proof of set membership system if it consists of two stages, commitment and proof phase. In the commitment stage, P interacts with V to commit to a set D which domain is $\{0,1\}^{\mathsf{maxIndSize}}$. Let* cmDb *be the transcript of*

[4]Note that our commit and prove primitive differs from the commit and prove functionality of [CLOS02] in the fact that our security definition is not based on the ideal/real world paradigm, and does not require a simulator. Furthermore in our definition black-box and size hiding property are explicit requirements that are not needed in [CLOS02].

this commitment. In the proof phase, the verifier probes indexes $i \in \{0,1\}^{|poly(n)|}$ *for a polynomial poly chosen by* V*, and for each* i*,* P *provides a possibly interactive proof* (π, v) *for the predicate* $D[i] = v$*.*

It satisfies the following conditions.

Completeness. *If* P *and* V *are honest, than* V *accepts the proof with probability* 1*.*

Black-box. *Both* P *and* V *make only oracle calls to cryptographic primitives.*

Efficient Verification. *For a proof associated to index* i*, the time spent by* V *to verify the proof is* $poly(i, n)$*.*

Soundness. *For every index* i*, and for every PPT malicious prover* P^**, let* cmDb^* *be the commitment computed by* P^* *in the commitment stage. Probability that* P^* *computes accepting proofs* (π_i^1, v_i^1) *and* (π_i^2, v_i^2) *for index* i *and* $v_i^1 \neq v_i^2$*, is negligible.*

Zero-Knowledge. *There exists a PPT simulator* Sim*, such that for any* V^**, any auxiliary information* z*, the following two probability ensembles are computational indistinguishable:* $\{\langle P(D), V^*(z)\rangle\}$ *and* $\{\langle \mathsf{Sim}^{D(\cdot)}(1^{|d|}), V^*(z)\rangle\}$*.*

Where $\{\langle \mathsf{Sim}^{D(\cdot)}(1^{|d|}), V^*(z)\rangle\}$ *corresponds to the following experiment. In the commitment phase* Sim *generate a "commitment" without receiving any information about the database. In the proof phase,* Sim *has oracle access to* D *and it is allowed to query it only for indexes requested by* V^**.*

A.2 Commitments Secure under Selective Opening Attacks

Commitment scheme. A commitment scheme is a two-phase protocol between a sender C and a receiver R. In the former phase, called the *commitment phase*, C commits to a secret bit b to R. Let c be the transcript of the interaction. In the later phase, called the *decommitment phase*, C reveals a bit b' and proves that it is the same as b that was hidden in the transcript c of the commitment phase. Typically, there are two security properties w.r.t. a commitment scheme. The *binding* property requires that after the commitment phase, a malicious sender cannot decommitment c to two different values in the decommitment phase. The *hiding* property guarantees that a malicious receiver learns nothing about b in the commitment phase. A commitment scheme can be either Statistically Binding (but Computationally Hiding) or Statistically Hiding (but Computationally Binding).

Definition 7 (Commitment Scheme)**.** *A (bit) commitment scheme* $\mathsf{Com} = (\mathsf{C}, \mathsf{R})$ *is a two-phase protocol consists of a pair of* PPT *Turing machines* C *and* R*. In the commit phase,* C *runs on a private input* $b \in \{0,1\}$ *and a transcript* $c = \langle \mathsf{C}(b), \mathsf{R}\rangle$ *is obtained after interacting with* R*. In the decommitment phase,* C *reveals a bit* b' *and* R *accepts the value committed to be* b' *if and only if* C *can convince* R *that* $b' = b$*. In a commitment scheme, the following security properties hold for any* PPT *adversary* A*.*

- **Correctness:** *if sender and receiver both follow the protocol, then for all* $b \in \{0,1\}$*, when the sender commits and opens to* b*,* R *outputs* b*.*
- **Hiding:** *for every* PPT *adversary* A *and auxiliary input* z*, the probability ensembles* $\{\langle \mathsf{C}(0), A(z)\rangle\}$ *and* $\{\langle \mathsf{C}(1), A(z)\rangle\}$ *are indistinguishable, where* $\{\langle \mathsf{C}(b), A(z)\rangle\}$ *denote the random variable describing the output of* A *running on auxiliary input* z*, with a honest sender committing to a bit* b *by running* Com*.*

Binding: *for every* PPT *adversary A, and for all but a negligible probability over the coins of* R*, after the commitment phase, the probability that A can successfully open the commitment both as 0 and 1 is negligible.*

A commitment scheme is statistically biding (resp. statistically hiding) if its binding (resp. hiding) property is secure against any unbounded adversary A. In our protocol we use statistically hiding commitments. It is known how to construct a two-round statistically hiding commitment scheme from any family of collision-resistant hash functions [HM96] or an $O(n/\log n)$-round statistically hiding commitment from any one-way function [HR07, HNO+09].

Selective Opening Security. In our constructions we have a prover that commits to a bunch of views, and then reveals only some of them, and the security of our constructions relies on the fact that the hiding of the remaining *unopened* commitments holds. This security requirement is defined as *hiding in presence of selective openings attack*. Similarly to Witness Indistinguishability and Zero Knowledge, there exist two definitions of Selective Opening security. The Indistinguishability based definition, due to [Hof11], guarantees indistinguishability of the unrevealed messages. The simulation based definition, introduced in [DNRS03], guarantees the existence of a simulator which is able to commit to random messages and later equivocate some of the messages in the opening phase. In the following we provide the formal definitions.

Definition 8 (Indistinguishability under Selective Openings [Hof11])**.** *Let $N = N(n) > 0$ be polynomially bounded, and let $\mathcal{I} = (\mathcal{I}_N)_N$ be a family of sets such that each $\mathcal{I}_N$ is a set of subsets of $[N]$. A commitment scheme $\mathsf{Com} = (\mathsf{C}, \mathsf{R})$ is indistinguishable under selective openings (short IND-SO-COM secure) iff for every PPT n-message distribution $\mathcal{M}$ and every PPT adversary A, and any auxiliary input z to A and $\mathcal{M}$, we have that $\mathbf{Adv}^{\mathsf{ind\text{-}so}}_{\mathsf{Com},\mathcal{M},A}$ is negligible. Here:*

$$\mathbf{Adv}^{\mathsf{ind\text{-}so}}_{\mathsf{Com},\mathcal{M},A} = \left|\mathrm{Prob}[\mathbf{Exp}^{\mathsf{ind\text{-}so\text{-}real}}_{\mathsf{Com},\mathcal{M},A}(N) = 1] - \mathrm{Prob}[\mathbf{Exp}^{\mathsf{ind\text{-}so\text{-}ideal}}_{\mathsf{Com},\mathcal{M},A}(N) = 1]\right| \leq \nu(N)$$

The probability is taken over the choice of the random coins of the sampling algorithm $\mathcal{M}$, and the adversary A. Experiments $\mathbf{Exp}^{\mathsf{ind\text{-}so\text{-}real}}_{\mathsf{Com},\mathcal{M},A}$ and $\mathbf{Exp}^{\mathsf{ind\text{-}so\text{-}real}}_{\mathsf{Com},\mathcal{M},A}$ are defined below.

Experiment $\mathbf{Exp}^{\mathsf{ind\text{-}so\text{-}real}}_{\mathsf{Com},\mathcal{M},A}(z,N)$*:*	*Experiment* $\mathbf{Exp}^{\mathsf{ind\text{-}so\text{-}ideal}}_{\mathsf{Com},\mathcal{M},A}$ *: (z,N)*
$\mathbf{M} \leftarrow \mathcal{M}(z)$	$\mathbf{M} \leftarrow \mathcal{M}(z)$
$I \leftarrow \langle \mathsf{C}_i(\mathtt{com}, M_i)_{i\in[n]}, A(\mathtt{recv}, z)\rangle$	$I \leftarrow \langle \mathsf{C}_i(\mathtt{com}, M_i)_{i\in[n]}, A(\mathtt{recv}, z)\rangle$
$\langle \mathsf{C}_i(\mathtt{open})_{i\in I}, A(\mathtt{open})\rangle$	$\langle \mathsf{C}_i(\mathtt{open})_{i\in I}, A(\mathtt{open})\rangle$
	$\mathbf{M}' \leftarrow \mathcal{M}\|M_I(z)$
return $A(\mathtt{guess},\mathbf{M})$	*return* $A(\mathtt{guess},\mathbf{M}')$

Where by $\langle \mathsf{C}_i(\mathtt{com}, M_i)_{i\in[n]}, A(\mathtt{recv})\rangle$ we denote the interaction between the adversary A and a sender for the commitment phase, and by $\langle \mathsf{C}_i(\mathtt{open})_{i\in I}, A(\mathtt{open})\rangle$ we denote such interaction in the opening phase. $\mathcal{M}|M_I$ denotes the message distribution $\mathcal{M}$ conditioned on the revealed values M_I.

Definition 9 (Simulation under Selective Opening Attacks [BHY09, Hof11])**.** *A commitment scheme $\mathsf{Com} = (\mathsf{C}, \mathsf{R})$ is secure against selective opening attacks if for all N, all sets $I \in \mathcal{I}$, all N-bit message distributions $\mathcal{M}$, all PPT relations $\mathcal{R}$, there exists an expected PPT machine Sim such that for any PPT malicious receiver R^* there exists a negl. function ϵ such that:*

$$\mathbf{Adv}^{\mathsf{sim\text{-}soa}}_{\mathsf{Com},\mathcal{M},A} = \left|\mathrm{Prob}[\mathbf{Exp}^{\mathsf{sim\text{-}soa\text{-}real}}_{\mathsf{Com},\mathcal{M},A}(N)=1] - \mathrm{Prob}[\mathbf{Exp}^{\mathsf{sim\text{-}soa\text{-}ideal}}_{\mathsf{Com},\mathcal{M},A}(N)=1]\right| \leq \nu(N)$$

The probability is taken over the choice of the random coins of the parties.

Experiment $\mathbf{Exp}^{\mathrm{real}}_{\mathsf{Com},\mathsf{C},\mathsf{R}^*}(n)$*:*	*Experiment* $\mathbf{Exp}^{\mathrm{ideal}}_{\mathsf{Com},\mathsf{Sim},\mathsf{R}^*}(n)$*:*
$pk \xleftarrow{\$} \mathsf{R}^*(1^n)$*;*	$pk \xleftarrow{\$} \mathsf{R}^*(1^n)$*;*
$\mathbf{b} \xleftarrow{\$} \mathcal{M}$*;*	$\mathbf{b} \xleftarrow{\$} \mathcal{M}$*;*
$I \xleftarrow{\$} \langle \mathsf{C}_i(pk, \mathtt{com}, \mathbf{b}[i])_{i\in[N]}, \mathsf{R}^*(pk, \mathtt{recv})\rangle$*;*	$I \xleftarrow{\$} \mathsf{Sim}^{\mathsf{R}^*}(pk)$*;*
$(\cdot, \tau) \xleftarrow{\$} \langle \mathsf{C}_i(\mathtt{open})_{i\in I}, \mathsf{R}^*(\mathtt{open})\rangle$*;*	$\tau \xleftarrow{\$} \mathsf{Sim}^{\mathsf{R}^*}(\mathbf{b}[i])_{i\in I}$*;*
output $\mathcal{R}(I, \mathbf{b}, \tau)$*.*	*output* $\mathcal{R}(I, \mathbf{b}, \tau)$*.*

A.3 Zero Knowledge

Definition 10 (Zero Knowledge)**.** *An interactive protocol (P, V) for a language L is* zero knowledge *if for every PPT adversary V^*, there exists a PPT simulator S such that the probability ensembles $\{\langle P, V^*(z)\rangle(x)\}_{x\in L, z\in\{0,1\}^*}$ and $\{S(x,z)\}_{x\in L, z\in\{0,1\}^*}$ are computationally indistinguishable, where $\langle P, V^*(z)\rangle(x)$ denotes the output of V^* when interacting with P on common input x and auxiliary input z.*

A.4 Universal Arguments

We recall the definition provided in [BG08]. Let $\mathcal{L}_{\mathcal{U}} = \{(M, x, \mathtt{t})\text{: } \exists w \text{ s.t. } ((M, x, \mathtt{t}), w) \in \mathcal{R}_{\mathcal{U}}\}$, where $((M, x, \mathtt{t}), w) \in \mathcal{R}_{\mathcal{U}}$ if M accepts (x, w) within $\mathtt{t}$ steps. Let $T_M(x, w)$ denote the number of steps made by M on input (x, w). Recall that $|(M, x, \mathtt{t})| = O(|M| + |x| + \log \mathtt{t})$; that is, $\mathtt{t}$ is given in binary.

Definition 11 (Universal argument [BG08])**.** *A universal argument system is an interactive protocol (P, V) that satisfies the following properties:*

- Efficient Verification. *There exists a polynomial p such that for any $y = (M, x, \mathtt{t})$, the total time spent by the (probabilistic) verifier strategy V, on the common input y, is at most $p(|y|)$. In particular, all messages exchanged in the protocol have a length smaller than $p(|y|)$.*

- Completeness via a relatively efficient prover. *For every $((M, x, \mathtt{t}), w) \in \mathcal{R}_{\mathcal{U}}$, $Pr[\langle P(w), V\rangle(M, x, \mathtt{t}) = 1] = 1$. Furthermore, there exists a polynomial p such that for every $((M, x, \mathtt{t}), w) \in \mathcal{R}_{\mathcal{U}}$ the total time spent by $P(w)$, on input $(M, x, \mathtt{t})$ is upper bounded by $p(|M| + T_M(x, w)) \leq p(|M| + \mathtt{t})$.*

- Computational Soundness. *For every polynomial-size circuit family $\{P^*_n\}_{n\in N}$ and $(M, x, \mathtt{t}) \in \{0,1\}^n \backslash \mathcal{L}_{\mathcal{U}}$:*

$$Pr\left[\langle P^*, V\rangle(M, x, \mathtt{t})\right] = 1 \leq \nu(n)$$

  where $\nu : \mathbb{N} \to [0, 1]$ is a negligible function.

- A weak proof-of-knowledge property. *For every positive polynomial p there exists a positive polynomial p' and a probabilistic polynomial-time oracle machine E such that the following holds: for every polynomial-size circuit family $\{P^*_n\}_{n\in N}$ and every sufficiently long y =*

$(M, x, \mathtt{t}) \in \{0,1\}^*$, *if* $Pr[\langle P_n^*, V\rangle(y) = 1] > 1/p(|y|)$, *then:*

$$Pr_r[\exists w = w_1, \cdots, w_{\mathtt{t}} \in \mathcal{R}_{\mathcal{U}}(y), \forall i \in [\mathtt{t}] \\ E_r^{P_n}(y, i) = w_i] > \frac{1}{p'(|y|)}$$

where $\mathcal{R}_{\mathcal{U}}(y) = \{w : (y, w) \in \mathcal{R}_{\mathcal{U}}\}$ *and* $E_r^{P_n^*}(.,.)$ *denotes the function defined by fixing the random tape of* E *to equal* r *and providing the resulting* E_r *with oracle access to* P_n^**. The oracle machine* E *is called a (knowledge) extractor.*

Definition 12 (Witness Indistinguishable UARG)**.** *A universal argument [BG08]* (P, V) *is called witness indistinguishable (WI) if, for every polynomial* p*, every polynomial size circuit family* $\{V_n^*\}_n \in \mathbb{N}$*, and every three sequences* $\langle y_n = (M_n, x_n, \mathtt{t}_n) : n \in \mathbb{N}\rangle$*,* $\langle w_n^1 : n \in \mathbb{N}\rangle$ *and* $\langle w_n^2 : n \in \mathbb{N}\rangle$ *such that* $|y_n| = n$ *and* $\mathtt{t}_n \leq p(|x_n|)$ *and* (y_n, w_n^1)*,* $(y_n, w_n^1) \in \mathcal{R}_{\mathcal{U}}$*, the ensembles* $\{\langle P(w_n^1), V^*\rangle(y_n)\}_{n\in\mathbb{N}}$ *and* $\{\langle P(w_n^2), V^*\rangle(y_n)\}_{n\in\mathbb{N}}$ *are computationally indistinguishable.*

A.5 Probabilistically Checkable Proof (PCP)

Here we provide a more formal definition of PCP.

Definition 13 (PCP - basic definition.)**.** *A probabilistically checkable proof (PCP) system with error bound* $\epsilon : \mathbb{N} \to [0, 1]$ *for a set* S *is a probabilistic polynomial-time oracle machine (called verifier), denoted by* V *satisfying the following:*

Completeness. *For every* $x \in S$ *there exists an oracle* π_x *such that* V*, on input* x *and access to oracle* π_x*, always accepts* x*.*

Soundness. *For every* $x \notin S$ *and every oracle* π*, machine* V*, on input* x *and access to oracle* π*, rejects* x *with probability at least* $1 - \epsilon(|x|)$*.*

Let r and q be integer functions. The complexity class $PCP_\epsilon[r(\cdot), q(\cdot)]$ consists of sets having a PCP system with error bound ϵ in which the verifier, on any input of length n, makes at most $r(n)$ coin tosses and at most $q(n)$ oracle queries.

Definition 14 (PCP – auxiliary properties).)**.** *Let* V *be a PCP verifier with error* $\epsilon : \mathbb{N} \to [0, 1]$ *for a set* $S \in NEXP$*, and let* R *be a corresponding witness relation. That is, if* $S \in Ntime(t(\cdot))$*, then we refer to a polynomial-time decidable relation* R *satisfying* $x \in S$ *if and only if there exists* w *of length at most* $t(|x|)$ *such that* $(x, w) \in R$*. We consider the following auxiliary properties:*

Relatively efficient oracle construction. *This property holds if there exists a polynomial-time algorithm* P *such that, given any* $(x, w) \in R$*, algorithm* P *outputs an oracle* $\frac{1}{2}_x$ *that makes* V *always accept (i.e., as in the completeness condition).*

Nonadaptive verifier. *This property holds if the verifier's queries are determined based only on the input and its internal coin tosses, independently of the answers given to previous queries. That is,* V *can be decomposed into a pair of algorithms* $\mathsf{Q}_{\mathsf{pcp}}$ *and* $\mathsf{D}_{\mathsf{pcp}}$ *that on input* x *and random tape* r*, the verifier makes the query sequence* $\mathsf{Q}_{\mathsf{pcp}}(x, r, 1), \mathsf{Q}_{\mathsf{pcp}}(x, r, 2), \ldots, \mathsf{Q}_{\mathsf{pcp}}(x, r, p(|x|))$*, obtains the answers* $b_1, \ldots, b_{p(|x|)}$*, and decides according to* $\mathsf{D}_{\mathsf{pcp}}(x, r, b_1, \ldots, b_{p(|x|)})$*, where* p *is some fixed polynomial.*

Efficient reverse sampling. *This property holds if there exists a probabilistic polynomial-time algorithm S such that, given any string x and integers i and j, algorithm S outputs a uniformly distributed r that satisfies $\mathsf{Q}_{\mathsf{pcp}}(x, r, i) = j$, where $\mathsf{Q}_{\mathsf{pcp}}$ is as defined in the previous item.*

A proof-of-knowledge property. *This property holds if there exists a probabilistic polynomial-time oracle machine E such that the following holds: for every x and π, if $Pr[V^{\pi}(x) = 1] > \epsilon(|x|)$, then there exists $w = w_1, \cdots, w_t$ such that $(x, w) \in R$ and $Pr[E^{\pi}(x, i) = w_i] > 2/3$ holds for every $i \in [t]$.*

It can be verified that any $S \in NEXP$ has a PCP system that satisfies all of the above properties [BG08].

A.5.1 Probabilistically Checkable Proofs of Proximity

A "PCP of proximity" (PCPP, for short) proof [BSGH+06] is a relaxation of a standard PCP, that only verifies that the input is *close* to an element of the language. The advantage of such relaxation is that the verifier can check the validity of a proof without having to read the entire theorem, but just poking few bits. More specifically, in PCPP the theorem is divided in two parts. A public part, which is read entirely by the verifier, and a private part, for which the verifier has only *oracle* access to. Therefore, PCPP is defined for pair languages, where the theorem is in the form (a, y) and the verifier knows a and has only oracle access to y.

Definition 15 (Pair language [DSK12].)**.** *A pair language L is simply a subset of the set of string pairs $L \subseteq \{0,1\}^* \times \{0,1\}^*$. For every $a \in \{0,1\}^*$ we denote $L_a = \{y \in \{0,1\}^* : (a, y) \in L\}$.*

Definition 16 (Relative Hamming distance)**.** *Let $y, y' \in \{0,1\}^K$ be two strings. The relative Hamming distance between y and y' is defined as $\Delta(y, y') = |\{i \in [K] : y_i \neq y'_i\}|/K$. We say that y is δ-far from y' if $\Delta(y, y') > \delta$. More generally, let $S \subseteq \{0,1\}^K$; we say y is δ-far from S if $\Delta(y, y') > \delta$ for every $y' \in S$.*

Definition 17 (PCPP verifier for a pair language)**.** *For functions $s, \delta : \mathbb{N} \to [0, 1]$, a verifier V is a probabilistically checkable proof of proximity (PCPP) system for a pair language L with proximity parameter δ and soundness error s, if the following two conditions hold for every pair of strings (a, y):*

- *Completeness: If $(a, z) \in L$ then there exists π such that $V(a)$ accepts oracle $y \circ \pi$ with probability 1. Formally:*

$$\exists \pi \mathrm{Prob}_{(\mathsf{Q},\mathsf{D}) \leftarrow V(a)}[\mathsf{D}((y \circ \pi)|_{\mathsf{Q}}) = 1] = 1.$$

- *Soundness: If y is $\delta(|a|)$-far from $L(a)$, then for every π, the verifier $V(a)$ accepts oracle $y \circ \pi$ with probability strictly less than $s(|a|)$. Formally:*

$$\forall \pi \mathrm{Prob}_{(\mathsf{Q},\mathsf{D}) \leftarrow V(a)}[\mathsf{D}((y \circ \pi)|_{\mathsf{Q}})] = 1 < s(|a|).$$

It is important to note that the query complexity of the verifier depends only on the public input a.

Theorem 1 ((Theorem 1 of [DSK12])). *Let $T : \mathbb{N} \to \mathbb{N}$ be a non-decreasing function, and let L be a pair language. If L can be decided in time T then for every constant $\rho \in (0,1)$ there exists a PCP of proximity for L with randomness complexity $O(\log T)$, query complexity $q = O(1/\rho)$, perfect completeness, soundness error $1/2$, and proximity parameter ρ.*

We note that the soundness error $\frac{1}{2}$ can be reduced to $\frac{1}{2^u}$ by running V with independent coins u times. This blows up the randomness, query, and time complexity of V by a (multiplicative) factor of u (but does not increase the proof length).

We also assume that our PCP of Proximity satisfies the efficient-resampling property [BSCGT12].

A.6 Perfectly secure multi-party computation underline the MPC-in-the-head

Under the assumption that there exists a synchronous network over secure point-to-point channels, [BOGW88] shows that for every n-ary function $f : (\{0,1\}*)^n \to (\{0,1\}*)^n$, there exists a t-secure MPC protocol MPC-$\mathcal{F}$ that securely computes f in the semi-honest model for any $t < n/2$, and in the malicious model for any $t < n/3$, with perfect completeness and security.

More formally, we denote by A the real-world adversary running on auxiliary input z, and by SimMpc the ideal-world adversary. We then denote by $REAL_{\pi,A(z),I}(\bar{x})$ the random variable consisting of the output of A controlling the corrupted parties in I and the outputs of the honest parties. Following a real execution of π where for any $i \in [n]$, party P_i has input x_i and $\bar{x} = (x_1, \ldots, x_n)$. We denote by $IDEAL_{f,S(z),I}(\bar{x})$ the analogous output of S and honest parties after an ideal execution with a trusted party computing f.

Definition 18. *Let $f : (\{0,1\}*)^n \to (\{0,1\}*)^n$ be an n-ary functionality and let Π be a protocol. We say that Π (n,t)-perfectly securely computes f if for every probabilistic adversary A in the real model, there exists a probabilistic adversary SimMpc of comparable complexity[5] in the ideal model, such that for every $I \subset [n]$ of cardinality at most t, every $\bar{x} = (x_1, \ldots, x_n) \in (\{0,1\}^*)^n$ where $|x_1| = \ldots = |x_n|$, and every $z \in \{0,1\}^*$, it holds that: $\{IDEAL_{f,\mathsf{SimMpc}(z),I}(\bar{x})\} \equiv_s \{REAL_{\pi,A(z),I}(\bar{x})\}$.*

Theorem 2 (BGW88). *Consider a synchronous network with pairwise private channels. Then, for every n-ary functionality f, there exists a protocol π_f that (n,t)-perfectly securely computes f in the presence of a static semi-honest adversary for any $t < n/2$, and there exists a protocol that (n,t)-perfectly securely computes f in the presence of a static malicious adversary for any $t < n/3$.*

Notice that all the above communication requirements to run the MPC protocol will not result in communication requirements for our commitment scheme, since we will use virtual executions of MPC that will be run only locally by players.

In our protocols, we need the following definition of *correctness in presence of malicious parties*, called robustness in [IKOS07].

Definition 19 (t-correctness). *We say that protocol Π realizes a deterministic n party functionality $f(v_1, \ldots, v_n)$ with perfect t-correctness if for all inputs $v_1, \ldots, v_n$ and for all randomness to $P_1, \ldots, P_n$, the probability that the output of one players is different from the output of f is 0.*

Definition 20 (t-robustness). *Let $T \subset \{P_1, \ldots, P_n\}$. We say that protocol Π realizes f with perfect t-robustness if it is perfectly correct in the presence of a semi-honest adversary, and for every malicious adversary corrupting parties in T the following holds. Fix a tuple $v_{\bar{T}}$ of inputs of the*

[5] Comparable complexity means that S runs in time that is polynomial in the running time of A.

honest parties. If f evaluates to 0 to all choices of the input v consistent with $v_{\bar{T}}$, then all honest parties are guaranteed to output 0.

Remark 2. We remark that this definition of robustness differs from the one provided in [IKOS07]. The difference is the following. In [IKOS07] the functionality f is tailored to verify the truthfulness of a statement. Namely f takes as public input a theorem "$x \in L$", and private inputs $w_1, \ldots, w_n$ and outputs 1 iff $w = \bigoplus_i w_i$ is a valid witness for $x \in L$. Now, if $x \notin L$, then there exists no witness, therefore it holds that *for all* $w_1, \ldots, w_n$ the function f on input x must output 0.

In our setting we need functionalities that check if the *secret* inputs of the players satisfy some predicate. As such, there might always exist a tuple of inputs such that the predicate is satisfied. For example, consider the functionality that takes in input VSS views and check if they reconstruct to a specific value v: clearly it is always true that there exists a set of VSS views $v_1, \ldots, v_n$ that reconstruct to v. Therefore, the question makes sense only if one *fixes* a set of views $v_1^*, \ldots, v_n^*$ and ask whether such set correctly reconstructs to the value v. (Contrast this with the function f that checks that $x \in L$, if $x \notin L$ then there exist no inputs that make f output 1).

Therefore, our definition of robustness says the following. Fix inputs $v_{\bar{T}}$ for the honest parties, then if for all possible inputs of the malicious parties v_T it holds that $f(v_T, v_{\bar{T}}) = 0$ then at the end of the protocol the honest parties will output 0.

Use of perfect t-private MPC Protocols in our Security Proofs. We use MPC protocols with perfect completeness and security. For our proofs it will be sufficient to have perfect t-privacy wrt to a semi-honest adversary. The reason is that, in all our constructions a malicious verifier can only *observe* t views of the MPC-in-the-head played by P. We will denote by SimMpc the simulator associated to the MPC protocol. SimMpc takes t inputs from the corrupted parties, say $x_{p_1}, \ldots, x_{p_t}$ and the output out from the ideal functionality and gives in output the views of players $P_{p_1}, \ldots, P_{p_t}$ consistent with an honest execution of the protocol Π with input $x_{p_1}, \ldots, x_{p_t}$ and outputs out.

Verifiable Secret Sharing (VSS) functionality. The formal definition of VSS follows.

Definition 21 (VSS Scheme)**.** *An $(n+1, t)$-perfectly secure VSS scheme consists of a pair of protocols $\mathtt{VSS} = \langle \mathtt{Share}, \mathtt{Recon} \rangle$ that implement respectively the sharing and reconstruction phases as follows.*

`Share`**.** *Player P_{n+1} referred to as dealer runs on input a secret s and randomness r_{n+1}, while any other player P_i, $1 \leq i \leq n$, runs on input a randomness r_i. During this phase players can send (both private and broadcast) messages in multiple rounds.*

`Recon`**.** *Each shareholder sends its view v_i of the sharing phase to each other player, and on input the views of all players (that can include bad or empty views) each player outputs a reconstruction of the secret s.*

All computations performed by honest players are efficient. The computationally unbounded adversary can corrupt up to t players that can deviate from the above procedures. The following security properties hold.

Commitment: *if the dealer is dishonest then one of the following two cases happen: 1) during the sharing phase honest players disqualify the dealer, therefore they output a special value $\perp$ and will refuse to play the reconstruction phase; 2) during the sharing phase honest players do not disqualify the dealer, therefore such a phase determines a unique value s^* that belongs to the set of possible legal values that does not include $\perp$, which will be reconstructed by the honest players during the reconstruction phase.*

Secrecy: *if the dealer is honest then the adversary obtains no information about the shared secret before running the protocol* `Recon`.

Correctness: *if the dealer is honest throughout the protocols then each honest player will output the shared secret s at the end of protocol* `Recon`.

Direct implementations of $(n+1, \lfloor n/3 \rfloor)$-perfectly secure VSS schemes can be found in [BOGW88, CDD+99]. In this paper we use VSS with a deterministic reconstruction procedure, as provided in [GIKR01] that implements an $(n+1, \lfloor n/4 \rfloor)$-perfectly secure VSS scheme.

A.7 Other Definitions

In this section we provide the definition of other tools used in our constructions.

Collision Resilient Hash Function (CRHFs) A collection of (uniformly polynomial-time computable) functions $\{\mathcal{H} : \{0,1\}^* \to \{0,1\}^\theta\}$ such that for every (nonuniform) family of polynomial-size circuits $\{C_n\}_{n\in\mathbb{N}}$ is a collision-resistant hashing if:

$$Pr_{h\leftarrow\mathcal{H}}[C_n(h) = (x,y) \text{ s.t. } x \neq y \text{ and } h(x) = h(y)] = \nu(n)$$

where ν is a negligible function, and the probability is taken over the uniform choice of h.

Extractable commitment schemes. Informally, a commitment scheme is said to be extractable if there exists an efficient extractor that having black-box access to any efficient malicious sender ExCom^* that successfully performs the commitment phase, is able to efficiently extract the committed string. We first recall the formal definition from [PW09] in the following.

Definition 22 (Extractable Commitment Scheme)**.** *A commitment scheme* $\mathsf{ExCom} = (\mathsf{ExCom}, \mathsf{ExRec})$ *is an extractable commitment scheme if given an oracle access to any* PPT *malicious sender* ExCom^**, committing to a string, there exists an expected* PPT *extractor* Ext *that outputs a pair* (τ, σ^*)*, where* τ *is the transcript of the commitment phase, and* σ^* *is the value extracted, such that the following properties hold:*

Simulatability: *the simulated view* τ *is identically distributed to the view of* ExCom^* *(when interacting with an honest* ExRec*) in the commitment phase.*

Extractability: *the probability that* τ *is accepting and* σ^* *correspond to* $\perp$ *is negligible. Moreover the probability that* ExCom^* *opens* τ *to a value different than* σ^* *is negligible.*

One can construct an extractable commitment scheme $\mathsf{ExCom} = (\mathsf{ExCom}, \mathsf{ExRec})$ with non-interactive opening from any commitment scheme $\mathsf{Com} = (\mathsf{C}, \mathsf{R})$ in a black-box manner as follows. Let ExCom be the sender, ExRec be the receiver, and $\mathsf{C}(\sigma;\omega)$ denote the commitment to a message σ computed using randomness ω. We will now show the steps of a statistically-hiding extractable commitment scheme by assuming that if at any time the received message is inconsistent with the protocol specification then the honest player aborts (e.g., the receiver would output $\perp$).

Commitment Phase:

1. ExCom on input a message σ, generates k random strings $\{r_i^0\}_{i\in[k]}$ of the same length as σ, and computes $\{r_i^1 = \sigma \oplus r_i^0\}_{i\in[k]}$, therefore $\{\sigma = r_i^0 \oplus r_i^1\}_{i\in[k]}$. Then ExCom uses Com to commit to the k pairs $\{(r_i^0, r_i^1)\}_{i\in[k]}$. That is, ExCom and ExRec produce $\{c_i^0 = \langle \mathsf{C}(r_i^0, \omega_i^0), \mathsf{R}\rangle, c_i^1 = \langle \mathsf{C}(r_i^1, \omega_i^1), \mathsf{R}\rangle\}_{i\in[k]}$.

2. ExRec responses to ExCom by sending a random k-bit challenge string $r' = (r'_1, \ldots, r'_k)$.
3. ExCom decommits $\{c_i^{r'_i}\}_{i\in[k]}$ (i.e., non-interactively opens k of previous commitments, one per pair).
4. ExRec verifies that commitments have been opened correctly.

Decommitment Phase:

1. ExCom sends σ and non-interactively decommits the other k commitments $\{c_i^{\bar{r}'_i}\}_{i\in[k]}$, where $\bar{r}'_i = 1 - r'_i$.
2. ExRec checks that all k pairs of random strings $\{r_i^0, r_i^1\}_{i\in[k]}$ satisfy $\{\sigma = r_i^0 \oplus r_i^1\}_{i\in[k]}$. If so, ExRec takes the value committed to be σ and $\perp$ otherwise.

The extractor can simply run as a receiver, and if any of the k commitments is not accepting, it outputs $\sigma^* = \perp$. Otherwise, it rewinds (Step 2) and changes the challenge until another k well formed decommitments are obtained. Then it verifies that for each decommitment, the XOR of all pairs corresponds to the same string. Then the extractor can extract a value from the responses of these two distinct challenges. Due to the binding of the underlying commitment, the value extracted by the extractor will correspond to the value opened later by the malicious committer. The extractor by playing random challenges in each execution of Step 2 is perfectly simulating the behavior of the receiver. The running time of the extractor is polynomial (as can be argued following arguments similar to [PW09])

Error-correcting codes. A pair $(\mathsf{ECC}, \mathsf{DEC})$ is an error-correcting code with polynomial expansion if ECC and DEC are polynomial-time computable functions that satisfy the following three properties.

Correctness: $\mathsf{DEC}(\mathsf{ECC}(x), r) = x$ where r is a sufficiently long random string.

Polynomial expansion: there exists some function $\ell(\cdot)$ such that $\ell(k) < k^c$ for some constant $c > 0$, and for every string $x \in \{0,1\}^*, |\mathsf{ECC}(x)| = \ell(|x|)$. Constant distance: there exists some constant $\delta > 0$ such that for every $x \neq x' \in \{0,1\}^k, |\{i|\mathsf{ECC}(x)_i \neq \mathsf{ECC}(x')_i\}| \geq \delta\ell(k)$.

B Details on BB Commit-and-prove Protocol via Extendable Merkle Tree

In this section we provide the specification of our implementation of BB input-size hiding commit-and-prove protocol via extendable Merkle Tree.

Notation and Parameters. The size of the string shared by the players in the VSS is m bits. The size of the view of each player is $v = poly(m, n)$. We use a family of hash functions $\mathcal{H} : \{0,1\}^v \to \{0,1\}^{\frac{m}{2n}}$.

B.1 Building an Extendable Merkle Tree

We formally describe the procedure used by P to construct an extendable Merkle Tree. We denote such procedure by BuildRealTree. BuildRealTree takes as input a string s and a CRHF h.

Procedure BuildRealTree**.**
On input a string s, let $\ell_s = \log |s|$, and the hash function h. The tree is built as follows.

- (Leaves). Level ℓ_s. For each $\gamma \in \{0,1\}^{\ell_s}$ do:
  - (Label) $l^\gamma = s_\gamma$.
  - (VSS) Run VSS in-the-head among $n+1$ players where player P_{n+1} is the dealer and secret shares the string l^γ. Namely, obtain $\{P_1^{\mathsf{VSS}^\gamma}|\cdots|P_{n+1}^{\mathsf{VSS}^\gamma}\} \leftarrow \mathtt{Share}(l^\gamma)$. Note that the view of the dealer P_{n+1} is not included in the views committed.
    The node at position γ is the pair: $[l^\gamma, (P_1^{\mathsf{VSS}^\gamma}|\cdots|\ P_n^{\mathsf{VSS}^\gamma})]$. For short, we denote it by: $[l^\gamma, \mathsf{VSS}^\gamma]$.
- (Internal Node) For $j = \ell_s - 1$ to 1. For each $\gamma \in \{0,1\}^j$
  - (Label) The label here is the hash of the VSS of the children. Namely:
$$l^\gamma = h(P_1^{\mathsf{VSS}^{\gamma 0}})|,\ldots,h(P_n^{\mathsf{VSS}^{\gamma 0}})|h(P_1^{\mathsf{VSS}^{\gamma 1}})|,\ldots,h(P_n^{\mathsf{VSS}^{\gamma 1}})$$
  - (VSS) Compute $[P_1^{\mathsf{VSS}^\gamma}|\cdots|P_{n+1}^{\mathsf{VSS}^\gamma}] \leftarrow \mathtt{Share}(l^\gamma)$.
- (Root). The root is the last label l^λ.
- (Level). Run VSS in-the-head to secret share the depth ℓ_s. Obtain views $[P_1^{\mathsf{depth}},\ldots,P_n^{\mathsf{depth}}] \leftarrow \mathtt{Share}(\ell_s)$. For short, we denote the concatenation of such views by $\mathsf{VSS}^{\mathsf{depth}}$.

Return the root l^λ and the VSS-representation of the length of the real tree: $P_1^{\mathsf{depth}},\ldots,P_n^{\mathsf{depth}}$.

B.2 Size-hiding Black-box Commit-and-Prove Protocol

Here we provide the details of the commit and prove protocol outlined in Section 3.2.2. The protocol consists in 3 stages. In the first stage, the prover commits to a string s by constructing the extendable Merkle tree, using procedure $\mathsf{BuildRealTree}$. The commitment consists in the commitment of the root, and of the VSS of the depth of the tree. We denote this stage by $\mathsf{BBCom}(s)$.

In the second stage, the verifier selects indexes to be checked. V selects a subset of indexes I_ℓ for each possible depth $\ell = 1,\ldots,\ell_d$. Namely, V selects $\mathcal{I} = I_1,\ldots,I_{\ell_d}$.

In the third stage, called proof phase, P proves that the bits lying on position I_{ℓ_s} satisfy property ϕ. Such proof goes as follows. First, P commits to one path for each query in $\mathcal{I}$. The length of each path depends on the query. Namely, a path for a query in I_ℓ has depth ℓ. For queries in I_ℓ with $\ell < \ell_s$, P will just retrieve the real paths computed above. For queries in I_ℓ with $\ell > \ell_s$, P will extend the real path with fake nodes. Such node are computed so that the hash consistency is always verified, while the node consistency (i.e., the innode property) is not. For each path so obtained, P computes: 1) one $\mathsf{MPC\text{-}in}$ for each node of the path (to prove node consistency); 2) one MPC-ϕ for the leaves (for a set I_ℓ, MPC-ϕ is computed only for the nodes at level ℓ). Then, V sends positions $p_1,\ldots,p_t$ of players to be opened.

P then opens the views of players $P_{p_1},\ldots,P_{p_t}$ for all MPC-in-the-head protocols: VSS, $\mathsf{MPC\text{-}in}$, MPC-ϕ. Additionally P opens the substrings of the label part l corresponding to positions $p_1,\ldots,p_t$. Finally, V is convinced if all the opened views are consistent.

We denote the third stage as $\mathsf{BBProve}(\cdot)$. $\mathsf{BBProve}$ takes as input the queries $I_1,\ldots,I_{\ell_d}$ and the property ϕ to be tested.

Protocol 3. *Commit-and-prove protocol via extendable Merkle tree.*

Commitment. $\mathsf{BBCom}(s)$

- *($V \to P$) V sends $h \in \mathcal{H}$ to P*
- *P runs* $\mathsf{BuildRealTree}(s, h)$ *and obtains the root* l^λ *and the views* $[P_1^{\mathsf{depth}}, \ldots, P_n^{\mathsf{depth}}]$ *. Compute commitments* $\mathsf{C}_\lambda = \mathsf{SHCom}(l^\lambda)$ *and* $\mathsf{C}_i = \mathsf{SHCom}(P_i^{\mathsf{depth}})$.
- *($P \to V$). Send* C_λ, C_1, $\ldots$, C_n *to* V.

Queries

- *V computes sets $\mathcal{I} = \{I_1, \ldots, I_{\ell_d}\}$ of queries for each possible size of s using an arbitrary (possibly randomized) algorithm that we denote as* $\mathsf{GetIndex}$*. Namely, for $j = 1$ to ℓ_d V computes $I_j \leftarrow \mathsf{GetIndex}(j)$.*
- *($V \to P$) V sends $I_1, \ldots, I_{\ell_d}$ to P.*
  For easiness of explanation we now assume that each set I_j contains one query q_j only. Hence, we consider $\mathcal{I} = \{q_1, \ldots, q_{\ell_d}\}$.

Proof: $\mathsf{BBProve}(q_1, \ldots, q_{\ell_d})$ *for property ϕ.*

1. *P* **computes a path for each query.** *For a query q_j, P constructs a path of length j as follows. Let $\ell_s = \mathtt{Recon}(\mathsf{VSS}^{\mathsf{depth}})$.*

   Case $j > \ell_s$: queries hitting a depth greater than the depth of the real tree.
   For each $j \in \{\ell_d, \ell_d - 1, \ldots, \ell_s\}$, compute a path of length j as follows.

   (a) *Compute fake leaves:*
   Set l^{q_j} to a random m-bit string. Run $\mathtt{Share}(l^{q_j})$ to obtain views $\mathsf{VSS}^{q_j} = [P_1^{\mathsf{VSS}^{q_j}}, \cdots, P_n^{\mathsf{VSS}^{q_l}}]$.

   (b) *Compute fake nodes.*
   Let q_j^i be the i-bit prefix of query q_j, for $i = \ell_{d-1}$ to $\ell_s + 1$. If q_j^i has two children q_l^{i0}, q_j^{i1}, compute label $l^{q_j^i} = h(P_1^{\mathsf{VSS}^{q_j^{i0}}}), \cdots, h(P_n^{\mathsf{VSS}^{q_j^{i1}}})$. Note that labels are computed only for nodes having children. Run $\mathtt{Share}(l^{q_j^i})$ to obtain views $\mathsf{VSS}^{q_j^i} = [P_1^{\mathsf{VSS}^{q_j^i}}, \ldots, P_n^{\mathsf{VSS}^{q_j^i}}]$.

   (c) *Compute proof of consistency.*
   For each fake node $[l^{q_j^i}, [P_1^{\mathsf{VSS}^{q_j^i}} | \cdots | P_n^{\mathsf{VSS}^{q_j^i}}]]$ run $\mathsf{MPC\text{-}in}$ *using trapdoor condition $j > \ell_s$.*

   (d) *Add Real Nodes/Labels.*
   Let q_j^i be the i-bit prefix of query q_j, for $i = \ell_s, \ldots, 1$. Retrieve node $[l^{q_j^i}, \mathsf{VSS}^{q_j^i}]$ from the real tree and compute $\mathsf{MPC\text{-}in}$ *proving that the* innode *property is satisfied.*

   (e) *Compute proof of predicate (proving* **trapdoor condition** *$j \neq \ell_s$), for the leaf q_j. Consider the node in position q_j, namely the VSS: $[P_1^{\mathsf{VSS}^{q_j}} | \cdots | P_n^{\mathsf{VSS}^{q_j}}]$. Run* $\mathsf{MPC\text{-}}\phi$ *using as witness the condition $j \neq \ell_s$ (namely, the query does not lie on the leaf level of the real tree). The proof of predicate is computed using the witness $j \neq \mathtt{Recon}(P_1^{\mathsf{depth}}, \ldots, P_n^{\mathsf{depth}})$.*

Case $j = \ell_s$: queries hitting the real depth. *Retrieve the path from the real tree, from the root to leaf q_{ℓ_s}. Then, for $i = \ell_s$ to 1, let $q^i_{\ell_s}$ be the i-bit prefix of the position q_{ℓ_s}.*

(a) Compute proof of consistency. Consider each node $[l^{q^i_j}, [P_1^{\mathsf{VSS}^{q^i_j}} | \cdots | P_n^{\mathsf{VSS}^{q^i_j}}]]$ and run MPC-in *to prove that the* innode *property is satisfied.*

(b) Compute proof of predicate. Consider the node in position q_{ℓ_s}, namely the VSS $[P_1^{\mathsf{VSS}^{q_{\ell_s}}} | \cdots | P_n^{\mathsf{VSS}^{q_{\ell_s}}}]$. Run MPC-$\phi$ *using to prove that $\phi(v)$ is true and $j = \mathtt{Recon}(P_1^{\mathsf{depth}}, \ldots, P_n^{\mathsf{depth}})$, where $v = \mathtt{Recon}(P_1^{\mathsf{VSS}^{q_{\ell_s}}}, |\cdots|, P_n^{\mathsf{VSS}^{q_{\ell_s}}})$.*

Case $j < \ell_s$: queries lying on the real tree. *Retrieve the path from the real tree, from the root to leaf q_j. Then, for $i = \ell_s - 1$ to 1, let q^i_j be the i-bit prefix of the position q_j.*

(a) Compute proof of consistency. Consider each node $[l^{q^i_j}, [P_1^{\mathsf{VSS}^{q^i_j}} | \cdots | P_n^{\mathsf{VSS}^{q^i_j}}]]$ and run MPC-in *to prove the consistency between the VSS and the label.*

(b) Compute proof of predicate. Consider the node in position q_j, namely the VSS $[P_1^{\mathsf{VSS}^{q_j}} | \cdots | P_{n+1}^{\mathsf{VSS}^{q_j}}]$. Run MPC-$\phi$ *using the trapdoor condition $j \neq \mathtt{Recon}(P_1^{\mathsf{depth}}, \ldots, P_n^{\mathsf{depth}})$.*

Commitment of the paths. *Commit to each node, label,* MPC-in *and* MPC-ϕ *computed above, by committing* separately *to each view of MPC players, and to each hash of the label. Send commitments to the verifier.*

2. **Verification.**

($V \to P$). Pick randomly players $p_1, \ldots, p_t$ and send them to P.

($P \to V$). P opens views corresponding to players in position $p_1, \ldots, p_t$ for all the commitments above.

(Verification Steps)

(a) **Check Hash Consistency** *(parent property). For each node γ, check that $l^\gamma[p_j] = h(P_{p_j}^{\mathsf{VSS}^{\gamma 0}})$ and $l^\gamma[p_j + n] = h(P_{p_j}^{\mathsf{VSS}^{\gamma 1}})$ (for $j = 1, \ldots, t$).*

(b) **Check Consistency of Nodes** *(*innode *property). This property is checked for every node γ opened (except for the leaves). For each player $P_{p_j}^{\mathsf{in}^\gamma}$ of the MPC-in-the-head* MPC-in *check:*

i. Input consistency: check that player $P_{p_j}^{\mathsf{in}^\gamma}$ plays with inputs $P_{p_j}^{\mathsf{VSS}^\gamma}$, and $P_{p_j}^{\mathsf{depth}}$, namely the p_j-th VSS views for node γ and for the VSS of the depth. Additionally with inputs $l^\gamma[p_j]$, $l^\gamma[p_j + n]$, namely the hash values of the VSS views of the children. Namely, the input of $P_{p_j}^{\mathsf{in}^\gamma}$ is the tuple: $l^\gamma[p_j]$, $l^\gamma[p_j + n]$, $P_{p_j}^{\mathsf{VSS}^\gamma}$, $P_{p_j}^{\mathsf{depth}}$.

ii. Views consistency. Check that the opened views are consistent with the honest execution of the protocol and with each other.

iii. Output correctness. Check that the output of all the revealed views is 1.

(c) **Check Proof of Predicate.** *This property is checked only for the nodes that lie on the leaf level. Namely, only for the nodes at position $q_1, \ldots, q_{\ell_d}$.*
Check view of player $P_j^{\varphi^{q_l}}$, for $l = 1, \ldots, \ell_d$ and $j = p_1, \ldots, p_t$, as in the previous step.
If any of this check fails, abort.

In the above protocol, SHCom is a statistically-hiding commitment scheme, which is also IND-SO-secure under the indistinguishability definition of [Hof11] (see Def. 8). Protocols MPC-in, MPC-ϕ are perfectly t-private and t-correct MPC protocols [BOGW88].

Extractability. Protocol 3 can be made *extractable* by replacing the SH commitment SHCom with a SH extractable commitment ExCom. We denote by BBExtCom, BBExtProve the modified version. Extractability here means that one can extract the bits for which V asks the property to be tested, in time polynomial in $|d|$. Note that this does not mean that the *entire* string can be extracted in polynomial time. We use extractable version BBExtCom, BBExtProve in our WIUA protocol in order to achieve the weak-proof-of-knowledge property.

Security properties. The formal proof of the following properties is provided in Sec. C. In the following we just provide intuitions behind the proof. *Witness Indistinguishability.* Throughout the protocol, V sees only t views of many instances of MPC protocols. The remaining $(n-t)$ views remain committed. Due to the perfect t-privacy of the MPC protocols used, t views do not reveal any information about the inputs of the remaining players. Due to the SOA-security of the commitment scheme, the hiding of the remaining $(n-t)$ views is preserved. *Soundness.* Due to the t-correctness of protocols VSS, MPC-in, MPC-ϕ, the only way for a malicious prover to provide a false proof, is to cheat in the opening of the views asked by the verifier. Due to the collision resistance of the hash function, and to the binding of the commitment scheme, once the root of the tree has been fixed, there exists only one accepting opening for each node.

B.2.1 Protocol MPC-in

Protocol MPC-in is executed by P in order to prove consistency of a node $[l, \mathsf{VSS}]$. MPC-in is a (n,t)-perfect MPC protocol implementing the functionality $\mathcal{F}_{\mathsf{innode}}$ depicted in Fig. 5.

$\mathcal{F}_{\mathsf{innode}}$ takes as public input the index γ of the node that is being tested. $\mathcal{F}_{\mathsf{innode}}$ takes as private input the VSS part and the label of the node γ from the players.

The secret input of a player P_i is : $\left(P_i^{\mathsf{VSS}^\gamma}, P_i^{\mathsf{depth}}, l^\gamma[i], l^\gamma[i+n]\right)$. The functionality is the following. $\mathcal{F}_{\mathsf{innode}}$ first reconstructs the value from views VSS^γ sent in input by each player, let v be the string obtained. Then it computes the value of the label l^γ by concatenating values $l^\gamma[i], l^\gamma[i+n]$ received from each P_i. If $v = l^\gamma$ then it outputs 1 to all players. Otherwise, $\mathcal{F}_{\mathsf{innode}}$ reconstructs the depth of the tree ℓ_s from views $\{P_i^{\mathsf{depth}}\}_{i\in[n]}$. If the level of node γ is below the real tree (i.e., $|\gamma| > \ell_s$), then γ is a fake node. As such the innode property can be violated. Thus $\mathcal{F}_{\mathsf{innode}}$ outputs 1 to all players. If both checks fail, $\mathcal{F}_{\mathsf{innode}}$ outputs 0 to all players.

Properties of Protocol MPC-in. Here we state the properties that we require for MPC-in implementing functionality $\mathcal{F}_{\mathsf{innode}}$.

Definition 23 (t-robustness)**.** . *We say that protocol* MPC-in *realizes f with perfect t-robustness if it is perfectly correct in the presence of a semi-honest adversary, and furthermore for any computationally unbounded malicious adversary corrupting a set T of at most t players, and for any inputs $(v_1, \ldots, v_n)$, the following robustness property holds. If $\mathcal{F}_{\mathsf{innode}}(v_1, \ldots, v_n) = 0$ then the probability that some uncorrupted player outputs 1 in an execution of $\mathcal{F}_{\mathsf{innode}}$ in which the inputs of the honest players are consistent with $(v_1, \ldots, v_n)$ is 0.*

Functionality $\mathcal{F}_{\mathsf{innode}}$.
Public Input: Index of the node: γ.

- for $i = 1$ to n: obtain the secret input $P_i^{\mathsf{VSS}^\gamma}, P_i^{\mathsf{depth}}$, $l^\gamma[i], l^\gamma[i+n]$ from player P_i^{innode}.
- run $v \leftarrow \mathtt{Recon}(P_1^{\mathsf{VSS}^\gamma}, |\cdots|, P_n^{\mathsf{VSS}^\gamma})$. If the reconstruction fails output 0 to all players and halt.
- $\ell'_s = \mathtt{Recon}(P_1^{\mathsf{depth}}, |\cdots|, P_n^{\mathsf{depth}})$
- set $l^\gamma = l^\gamma[1]||\ldots||l^\gamma[2n]$.
- if $v = l^\gamma$ and $|\gamma| \leq \ell'_s$ output 1 to all players and halt.
- else if $|\gamma| > \ell'_s$, output 1 to all players and halt.
- Else output 0 and halt.

Figure 5: The $\mathcal{F}_{\mathsf{innode}}$ functionality.

B.2.2 Protocol MPC-ϕ

Protocol MPC-ϕ is executed by P in order to prove that a leaf (or a set of leaves) satisfies a certain predicate ϕ. MPC-ϕ is a (n,t)-perfect MPC protocol implementing the functionality $\mathcal{F}_\phi$ depicted in Fig. 6.

$\mathcal{F}_\phi$ takes as public input the index γ of the node that is being tested, and the property ϕ that must be tested. $\mathcal{F}_\phi$ takes as private input the VSS part and the label of the leaf γ from the players.

The secret input of a player P_i is : $\left(P_i^{\mathsf{VSS}^\gamma}, P_i^{\mathsf{depth}}\right)$. The functionality is the following. $\mathcal{F}_\phi$ reconstructs the value from views VSS^γ sent in input by each player, let v be the string obtained, and the value from views $\mathsf{VSS}^{\mathsf{depth}}$, let this value be ℓ_s. First, it computes $\phi(v)$ and output 1 if $\phi(v) = 1$ and the $\ell_s = |\gamma|$. Otherwise, it outputs 1, iff $\gamma \neq \ell_s$. This means that the leaf being tested is not a leaf of the real tree (but of a shorter/longer tree).

Otherwise, if both checks fail, $\mathcal{F}_\phi$ outputs 0 to all players.

$\mathcal{F}_\phi$ over multiple leaves. When ϕ must be checked over multiple leaves, the only change in $\mathcal{F}_{\mathsf{innode}}$ is the fact that each players participates with a VSS's view for each leaf, and that ϕ is computed on multiple values. For completeness we write $\mathcal{F}_\phi$ for multiple values in Fig 7.

Functionality $\mathcal{F}_\phi$ for property ϕ-$\mathsf{D}_{\mathsf{pcp}}$ In Fig. 8 we describe in details how $\mathcal{F}_\phi$ is instantiated for property ϕ-$\mathsf{D}_{\mathsf{pcp}}$. This functionality is used in our WIUA protocol shown in Sec. D.

Functionality $\mathcal{F}_\phi$ for property ϕ-ZK. In the public-coin ZK protocol shown in 4.2 the property to be tested is whether the PCPP verifier would accept the proof. Such property must be tested over two oracles, the theorem and the PCPP proof, which are committed using two distinct extendable merkle trees. We describe how the $\mathcal{F}_\phi$ is adapted to the ϕ-ZK in Fig. 9

Functionality $\mathcal{F}_\phi$. **Public Input:** Index of the leaf: γ, property to be tested ϕ.

- for $i = 1$ to n: obtain the secret input $P_i^{\mathsf{VSS}^\gamma}, P_i^{\mathsf{depth}}$, from player P_i^ϕ.
- run $v \leftarrow \mathtt{Recon}(P_1^{\mathsf{VSS}^\gamma}, |\cdots|, P_n^{\mathsf{VSS}^\gamma})$. If the reconstruction fails output 0 to all players and halt.
- $\ell'_s = \mathtt{Recon}(P_1^{\mathsf{depth}}, |\cdots|, P_n^{\mathsf{depth}})$
- if $\phi(v)$ is true and $\ell'_s = |\gamma|$, output 1 to all players and halt.
- Else if $|\gamma| \neq \ell'_s$, output 1 to all players and halt.
- Else output 0 and halt.

Figure 6: The $\mathcal{F}_\phi$ functionality.

Functionality $\mathcal{F}_\phi$ computed over multiple leaves.
Public Input: Index of the leaves: $\gamma_1, \ldots, \gamma_k$ (having same bit length), property to be tested ϕ.

- for $i = 1$ to n: obtain the secret input $P_i^{\mathsf{VSS}^{\gamma_1}}, \ldots, P_i^{\mathsf{VSS}^{\gamma_k}}, P_i^{\mathsf{depth}}$, from player P_i^ϕ.
- run $b_j \leftarrow \mathtt{Recon}(P_1^{\mathsf{VSS}^{\gamma_j}}, |\cdots|, P_n^{\mathsf{VSS}^{\gamma_j}})$, for $j = 1, \ldots, k$. If the reconstruction fails output 0 to all players and halt.
- $\ell'_s = \mathtt{Recon}(P_1^{\mathsf{depth}}, |\cdots|, P_n^{\mathsf{depth}})$
- if $\phi(b_1, \ldots, b_k)$ is true and $\ell'_s = |\gamma_1|$ output 1 to all players and halt.
- Else if $|\gamma_1| \neq \ell'_s$, output 1 to all players and halt.
- Else output 0 and halt.

Figure 7: The $\mathcal{F}_\phi$ functionality over multiple leaves.

Functionality $\mathcal{F}_\phi$ for property ϕ-$\mathsf{D}_{\mathsf{pcp}}$.
Public Input: Index of the leaves: $\gamma_1, \ldots, \gamma_k$ (having same bit length); theorem X, property to be tested $\mathsf{D}_{\mathsf{pcp}}(x, b_1, \ldots, b_k) = 1$.

- for $i = 1$ to n: obtain the secret input $P_i^{\mathsf{VSS}^{\gamma_1}}, \ldots, P_i^{\mathsf{VSS}^{\gamma_k}}, P_i^{\mathsf{depth}}$, from player P_i^ϕ.
- run $b_j \leftarrow \mathtt{Recon}(P_1^{\mathsf{VSS}^{\gamma_j}}, |\cdots|, P_n^{\mathsf{VSS}^{\gamma_j}})$, for $j = 1, \ldots, k$. If the reconstruction fails output 0 to all players and halt.
- $\ell'_s = \mathtt{Recon}(P_1^{\mathsf{depth}}, |\cdots|, P_n^{\mathsf{depth}})$
- if $\mathsf{D}_{\mathsf{pcp}}(X, b_1, \ldots, b_k)$ and $\ell'_s = |\gamma_1|$ is true, output 1 to all players and halt.
- Else if $|\gamma_1| \neq \ell'_s$, output 1 to all players and halt.
- Else output 0 and halt.

Figure 8: The $\mathcal{F}_\phi$ functionality for property ϕ-$\mathsf{D}_{\mathsf{pcp}}$

Functionality $\mathcal{F}_{\phi\text{-}\mathsf{ZK}}$.
Public Input:

1. Indexes for leaves of the first tree: $\gamma_1, \ldots, \gamma_k$.
2. Indexes for leaves of the second tree : $\delta_1, \ldots, \delta_k$.
3. Public theorem a for PCP of Proximity.
4. Property to be satisfied $\mathsf{D}_{\mathsf{pcpx}}(a, \alpha_1, \ldots, \alpha_k, \beta_1, \ldots, \beta_k) = 1$.

- for $i = 1$ to n: obtain from player player P_i^ϕ. the secret inputs:
  1. views from the nodes of the first tree: $P_i^{\mathsf{VSS}^{\gamma_1}}, \ldots, P_i^{\mathsf{VSS}^{\gamma_k}}, P_i^{\mathsf{depth1}}$.
  2. views from the nodes of the first tree: $P_i^{\mathsf{VSS}^{\delta_1}}, \ldots, P_i^{\mathsf{VSS}^{\delta_k}}, P_i^{\mathsf{depth2}}$.
- run $\alpha_j \leftarrow \mathtt{Recon}(P_1^{\mathsf{VSS}^{\gamma_j}}, |\cdots|, P_n^{\mathsf{VSS}^{\gamma_j}})$, for $j = 1, \ldots, k$. If the reconstruction fails output 0 to all players and halt.
- run $\beta_j \leftarrow \mathtt{Recon}(P_1^{\mathsf{VSS}^{\delta_j}}, |\cdots|, P_n^{\mathsf{VSS}^{\delta_j}})$, for $j = 1, \ldots, k$. If the reconstruction fails output 0 to all players and halt.
- $\ell'_{s1} = \mathtt{Recon}(P_1^{\mathsf{depth1}}, |\cdots|, P_n^{\mathsf{depth1}})$
- $\ell'_{s2} = \mathtt{Recon}(P_1^{\mathsf{depth2}}, |\cdots|, P_n^{\mathsf{depth2}})$
- if $\mathsf{D}_{\mathsf{pcpx}}(a, \alpha_1, \ldots, \alpha_k, \beta_1, \ldots, \beta_k) = 1$ and $\ell'_{s1} = |\gamma_1|$ output 1 to all players and halt.
- Else if $|\gamma_1| > \ell'_s$, output 1 to all players and halt.
- Else output 0 and halt.

Figure 9: The $\mathcal{F}_{\phi\text{-}\mathsf{ZK}}$ functionality.

B.3 Extensions

In this section we show how Protocol 3 can be extended to allow proofs over multiple strings.

Assume a prover wants to commit to several strings using BBCom (if size hiding must be ensured) or VSSCom (if the size can be revealed) and prove a property ϕ over *all* such strings. For instance, this case can happen when the prover cannot commit to all the strings at once because some of the strings are generated during the protocol execution (this is precisely the scenario of Barak's ZK protocol, in which the PCPP proof can be committed only after the trapdoor theorem has been generated). In this kind of scenarios, the prover might run $\mathsf{BBCom}(A)$ and then $\mathsf{BBCom}(B)$ and $\mathsf{VSSCom}(C)$, and later prove that a property $\phi(A, B, C)$ is satisfied. More specifically, ϕ is defined over (A_{I_A}, B_{I_B}, C) where (I_A, I_B) are jointly computed by $\mathsf{GetIndex}(\mathsf{size})$. Note that, GetIndex is computed on a value size which can depend on the application or the property that we are trying to prove. The value size can depend on the size of A and B. For greater clarity then we introduce and additional BB commitment of the value size. Thus, after string A, B, C are committed, P additional sends VSSCom of size.

To enable the prover to compute such ϕ over multiple committed strings we have to slightly modify procedure BBProve shown in Protocol 3. Recall that BBProve takes as input a set of queries $q_1, \ldots, q_{\ell_d}$ and performs the followings steps: 1) computes the commitments for the paths corresponding to the queries received, 2) prepares and commit to one instance of MPC-ϕ for the leaves of each path (belonging to the *single* tree) committed in Step 1; 3) after receiving $p_1, \ldots, p_t$ it opens t views for each node/MPC-in, MPC-ϕ sent before.

Very roughly, in the multiple strings setting, the main change is only in the computation of Step 2. Namely, now BBProve is invoked on multiple queries $((q_1^1, q_1^2), \ldots, (q_{\ell_d}^1, q_{\ell_d}^2))$ and protocol MPC-ϕ is computed over the leaves belonging to multiple trees (instead of a single tree), and possibly on the views of some string committed using VSSCom.

In the following we show how to modify procedure BBProve of Protocol 3 to work with multiple strings.

Protocol 4. *Commit-and-prove Protocol via Extendable Merkle Tree over Multiple Strings.*

Commitment

- *P runs* $\mathsf{BBCom}(A)$, $\mathsf{BBCom}(B)$ *with V. Additionally sends:* $\mathsf{VSSCom}(C) = P_1^{\mathsf{vss}C}, \ldots, P_n^{\mathsf{vss}C}$ *and* $\mathsf{VSSCom}(\mathsf{size}) = P_1^{\mathsf{vsssize}}, \ldots, P_n^{\mathsf{vsssize}}$ *to V.*

Queries

- *V computes sets* $\mathcal{I}_A = \{I_1^A, \ldots, I_{\ell_d}^A\}$; $\mathcal{I}_B = \{I_1^B, \ldots, I_{\ell_d}^B\}$; *For* $\kappa = 1$ *to* ℓ_d *V computes* $(I_j^A, I_j^B) \leftarrow \mathsf{GetIndex}(\kappa)$.
- *($V \to P$) V sends* $I_1, \ldots, I_{\ell_d}$ *to P.*
  For easiness of explanation we now assume that each set I_l^A, I_l^B *contains one query* q_l^A, q_l^B *only. Hence, we consider* $\mathcal{I} = \{q_1^A, q_1^B, \ldots, q_{\ell_d}^A, q_{\ell_d}^B\}$.

Proof: $\mathsf{BBProve}(\mathcal{I}_A, \mathcal{I}_B)$ *for property ϕ.*

1. **P computes a path for each query in $\mathcal{I}_A$.** *(Follow steps described in Protocol 3 without computing views* MPC-ϕ*).*
2. **P computes a path for each query in $\mathcal{I}_B$.** *(Follow steps described in Protocol 3 without computing views* MPC-ϕ*).*

3. *P* **compute proof of property ϕ over leaves in position $q^A_{\mathsf{size}}, q^B_{\mathsf{size}}$.** *Run* MPC-$\phi$ *to prove that $\phi(v_A, v_B, C)$ is true, where $v_A = \mathtt{Recon}([P_1^{\mathsf{VSS}^{q^A_{\mathsf{size}}}}, |\cdots|, P_n^{\mathsf{VSS}^{q^A_{\mathsf{size}}}}]_{\mathsf{path}A})$; $v_B = \mathtt{Recon}([P_1^{\mathsf{VSS}^{q^B_{\mathsf{size}}}}, |\cdots|, P_n^{\mathsf{VSS}^{q^B_{\mathsf{size}}}}]_{\mathsf{path}B})$, and $C = \mathtt{Recon}(P_1^{\mathsf{VSS}C}, \ldots, P_n^{\mathsf{VSS}C})$.*
4. *P* **compute proof of property ϕ over leaves in position q_l^A, q_l^B with $l \neq \mathsf{size}$.** *Compute proof of property (proving trapdoor condition $l \neq \ell_s$). Consider the node in position q_l, namely the VSS: $[P_1^{\mathsf{VSS}^{q_l}} | \cdots | P_n^{\mathsf{VSS}^{q_l}}]$. Run* MPC-$\phi$ *using as witness the condition $l \neq \ell_s$ (namely, the query does not lie on the leaf level of the real tree). Note: the proof of property is computed only using the VSS part of the node.*

Verification. *Follow Protocol 3 with the difference that V has to check the views of two paths $\mathsf{path}A, \mathsf{path}B$ for each query (instead of just one), and additionally has to check views for VSS of the size and VSS of C.*

C Security Proof our Size-Hiding Black-Box Commit-and-Prove Protocol

In this section we formally prove that Protocol 3 satisfies properties of Definition 5.

C.1 Soundness

Proof Overview. The soundness of Protocol 3 relies on the binding of the extendable merkle tree. Indeed, if it holds that once a commitment transcript is fixed, there exists at most one string that any PPT P^* can successfully open, then soundness follows directly from the t-robustness of the MPC protocols being used: $VSS, \mathsf{MPC\text{-}in}, \mathsf{MPC}\text{-}\phi$.

The proof is structured in four hybrids. In hybrid 1 the verifier can see everything in the clear, namely, the node of each path is sent without commitment. In this hybrid the binding of the tree relies only on the collision resistance property of the family $\mathcal{H}$, and because all nodes of the tree are revealed, the verifier can verify ϕ on its own, thus soundness follows directly from the collision resistance property as well. Once we established that when everything is sent in the clear P^* cannot cheat by opening nodes (VSS views) that are not consistent to the root of the real tree, we move to argue that P^* cannot cheat even if the verifier V obtains only t views and the consistency is proved via MPC-in-the-head. Thus, in hybrid 2 we consider a verifier that obtains only t views for each node of the real tree, and checks the consistency of such nodes by looking at the views of MPC-in players (for simplicity, at this point we still assume that the verifier knows the real depth of the tree ℓ_s and obtains all the views for the nodes at level ℓ_s, in order to check that Φ, on such nodes, is satisfied). In this hybrid P^* can cheat in the consistency proof by cheating in the execution of MPC-in. Due to the t-robustness of MPC-in, this happens with negligible probability. In hybrid 3, we consider a verifier V that does not know the real depth of the tree (i.e., ℓ_s), and therefore it checks the consistency of ℓ_d paths. In this hybrid the prover will add fake nodes, and for these it can cheat. However, it follows from the exact same argument as hybrid 3, that the probability that P^* successfully cheat in this hybrid, is negligible.

Hybrid 4 corresponds to the actual protocol where the paths are committed, and thus binding and soundness rely on the binding of the commitment scheme. The invariant is that in each hybrid, any P^* able to open two accepting but conflicting paths can be reduced to a collision forming algorithm working in hybrid 1.

Recall that a node is connected to its parent via two tiers. First, the node is connected to the label of the parent via hash function (here we use the collision resistance property of $\mathcal{H}$). Then, the label of the parent node is connected to the VSS part of the node, via the innode property (see Fig. 2 to visualize this connection). Such property is checked using MPC-in-the-head $\mathsf{MPC\text{-}in}$, which is run on top of the VSS part of the node and the VSS of the depth. Finally, the property ϕ is tested on the VSS part of the leaf in position q via $\mathsf{MPC}\text{-}\phi$.

For simplicity in the proof we consider a verifier asking for one query per level, namely $q_1, \ldots, q_{\ell_d}$ (instead of a set $I_1, \ldots, I_{\ell_d}$).

In the following we outline the four hybrids used in the proof.

Hybrid 1. (Breaking the Collision Resistance Property of the hash function (Sec. C.1.1)). In Hybrid 1 we consider a protocol where P sends everything in the *clear*. In this hybrid the size of the string, represented in $\mathsf{VSS}^{\mathsf{depth}}$, can be reconstructed by V. Hence, we assume that V only computes queries for level $\ell_s = \mathtt{Recon}(\mathsf{VSS}^{\mathsf{depth}})$. Let q be the node queried. In the proof phase, V obtains the path from the root to node q. For each node γ along the path, it checks the innode property by itself: given $[l^\gamma, \mathsf{VSS}^\gamma]$ it checks that $\mathtt{Recon}(\mathsf{VSS}^\gamma) = l^\gamma$. Furthermore, V tests property ϕ by itself, by checking $\phi(\mathtt{Recon}(\mathsf{VSS}^q)) = 1$. Hence, protocols $\mathsf{MPC\text{-}in}$ and $\mathsf{MPC}\text{-}\phi$ are not executed by P.

In this hybrid we show that, for any query q, if a PPT adversarial prover P^* provides two accepting paths $\mathsf{path0}, \mathsf{path1}$ that open to conflicting values (i.e, $\mathsf{path0}$ opens to v_0 and $\mathsf{path1}$ opens to v_1 and $v_0 \neq v_1$) with non-negligible probability, then we can transform P^* in a collision forming circuit, thus breaking the collision resistance property of $\mathcal{H}$.

Hybrid 2. (Based on t-robustness of MPC-in-the-head protocols (Sec. C.1.2)). In the second hybrid we consider an oblivious verifier that for every node γ only looks at t views among n views of VSS^γ (except for the leaf, i.e., the node in position q). However, for simplicity at this point we still assume that for $\mathsf{VSS}^{\mathsf{depth}}$, the verifier obtains all n views, and thus it can reconstruct the size of the tree by running $\ell_s \leftarrow \mathtt{Recon}(P_1^{\mathsf{depth}}, \ldots, P_n^{\mathsf{depth}})$.

In this hybrid V is able to check only t hash values for each label l. Only t views are not sufficient to run the VSS reconstruction phase and to check the innode property. Therefore in this hybrid P additionally sends the views of $\mathsf{MPC\text{-}in}$'s players, and the verifier checks the innode property by looking at both the t views of VSS^γ, $\mathsf{VSS}^{\mathsf{depth}}$ and of $\mathsf{MPC\text{-}in}^\gamma$'s players. For the leaf at position q, V still observes all n views of VSS^q. It reconstructs the value s_q and check that $\phi(s_q)$ is true.

In this hybrid we use the perfect t-correctness and t-robustness of $\mathsf{MPC\text{-}in}$ and VSS. Perfect t-correctness guarantees that, for any possible randomness and any possible inputs used by the *honest* players, the protocol outputs the correct value. Perfect t-robustness guarantees that in order to yield honest players to output a wrong value, the number of corrupted players must be strictly greater than t. Because t is a constant fraction of n, it follows that V will open one of the inconsistent views whp.

Hybrid 3. (Based on t-robustness of MPC-in-the-head protocols (Sec. C.1.3)). In the third hybrid we further restrict the knowledge of the verifier: V does not reconstruct the actual depth ℓ_s of the tree and it is not able to check property ϕ by itself. Consequently, in hybrid 3, V selects queries $q_1, \ldots, q_{\ell_d}$, one for each possible depth, and obtains ℓ_d paths from P. The prover P computes the paths that are outside the depth ℓ_s by adding fakes nodes. The MPC-in-the-head for such nodes

is successfully computed using the trapdoor condition: $\mathtt{Recon}(\mathsf{VSS}^{\mathsf{depth}} \neq l)$, for any depth l such that $l \neq \ell_s$. In this hybrid, binding follows from the t-robustness of $\mathsf{MPC\text{-}in}$ and $\mathsf{MPC}\text{-}\phi$.

Hybrid 4. (Breaking binding of the underlying commitment scheme (Sec. C.1.4)). In the last hybrid the prover commits to all the views and opens only t views requested by the verifier. Due to the binding of the commitment scheme used, this hybrid is indistinguishable from the previous one.

We now provide the formal proof of the following theorem.

Theorem 3. *If $\mathcal{H}$ is a family of collision-resistant hash functions, if protocols $\mathsf{MPC\text{-}in}, \mathsf{MPC}\text{-}\phi$ are* perfect *t-robust for functionalities $\mathcal{F}_{\mathsf{innode}}, \mathcal{F}_\phi$ respectively, if VSS has perfect t-robustness, where $t = \Omega(n)$ and $n = ct$ for some constant $c > 1$, and if SHCom is a computationally binding commitment scheme, then Protocol 3 is sound.*

Proof. The proof is organized in four hybrids.

C.1.1 Hybrid 1. (Breaking Collision Resistance).

In Hybrid 1 the prover sends all the messages in the clear. Fix $h \in \mathcal{H}$ chosen uniformly at random by V. In the $\mathsf{BBCom}(\cdot)$ phase P^* sends the root l^λ and $[P_1^{\mathsf{depth}}, \ldots, P_n^{\mathsf{depth}}]$ to V. From V reconstructs $\ell_s = \mathtt{Recon}(P_1^{\mathsf{depth}}, \ldots, P_n^{\mathsf{depth}})$ (if any view is inconsistent, V aborts). V then computes query $q \xleftarrow{\$} \mathsf{GetIndex}(\ell_s)$ and P^* answers with a ℓ_s-long authentication path in the clear, for value s_q. (As everything is in the clear, in this hybrid the prover and the verifier work only with paths of length ℓ_s.)

In this hybrid an authentication path consists of the sequence of nodes on the path from the leaf q to the root.

Definition 24. *A path is an* **accepting** *in this experiment if, for every node γ:*

1. *Hash consistency. For each label l^γ, for $j = 1, \ldots, n$ it holds that $l^\gamma[j] = h(P_j^{\mathsf{VSS}^{\gamma 0}})$ and $l^\gamma[j+n] = h(P_{j+n}^{\mathsf{VSS}^{\gamma 1}})$.*
2. *Node consistency (innode property). For each label l^γ, it holds that $l^\gamma = \mathtt{Recon}(P_1^{\mathsf{VSS}^\gamma}, |\cdots|, P_n^{\mathsf{VSS}^\gamma})$.*
3. *Leaf Consistency. It holds that s_q corresponds to the value $\mathtt{Recon}(P_1^{\mathsf{VSS}^q}, |\cdots|, P_n^{\mathsf{VSS}^q})$.*

Towards a contradiction, assume that, for query q, P^* is able to provide two accepting paths $\mathsf{path0}$ and $\mathsf{path1}$ which opens to different bits, with some non-negligible probability p. We say that such query q is *conflicting* (following terminology of [BG08]). We show that such P^* can be transformed in a collision forming circuit for h.

From the definition of accepting path, this means that $v_0 = \mathtt{Recon}(\mathsf{VSS}^q_{\mathsf{path0}})$ and $v_1 = \mathtt{Recon}(\mathsf{VSS}^q_{\mathsf{path1}})$ and $v_0 \neq v_1$.

Due to Lemma 1, if the reconstructed values are different, then there exist at least $t+1$ views in which $\mathsf{VSS}^q_{\mathsf{path0}}$ and $\mathsf{VSS}^q_{\mathsf{path1}}$ differ. Consequently, the label at parent position $l^{q'}$ (where q' corresponds to the first $|q|-1$ bits of q), should differ in the same positions. Due to the correctness of the VSS protocol a different value of a label $l^{q'}$ induces a different value of $\mathsf{VSS}^{q'}$, and so on and so forth.

Because, both path0 and path1 start from the *same* root, but end with *different* leaves $\mathsf{VSS}^q_{\mathsf{path0}}$ and $\mathsf{VSS}^q_{\mathsf{path1}}$ sharing values s_0 and s_1 respectively, there must be a label on the two paths, such that, different VSS children are hashed into the same parent label. Therefore we extract a collision. This contradicts the collision resistance property of h.

Lemma 1. *Let v_0 and v_1 be two strings such that $v_0 \neq v_1$. For $b = 0, 1$ let $\mathsf{VSS}^b = (P_1^b, \ldots, P_n^b)$ be the vector of VSS players sharing the value v_b, i.e., such that $\mathtt{Recon}(P_1^b, \ldots, P_n^b) = v_b$. If $v_0 \neq v_1$ then VSS_0 and VSS_1 differ in at least $t + 1$ views.*

Proof. For a VSS with threshold t, any subset of $n - t$ players is able to uniquely reconstruct the secret. Assume that VSS_0 and VSS_1 differ in less then $t + 1$ views and $v_0 \neq v_1$. This implies that they have at least $n - t$ views in common. Due to the perfect correctness of the VSS that we use, any $n - t$ set of views must reconstruct to the same secret, thus $\mathtt{Recon}(\mathsf{VSS}_0) = \mathtt{Recon}(\mathsf{VSS}_1)$, thus contradicting the assumption that $v_0 \neq v_1$. □

Remark 3. In this hybrid we show how to break collision resistance of $\mathcal{H}$, *assuming* that two accepting paths for the same queries q have been provided with some *non-negligible* probability by a PPT P^*. When using our tool to implement WIUARG and ZK protocol, we will need to show how to relate the probability of P^* proving a false theorem to the probability of providing two accepting paths.

Hybrid 1.1. This hybrid is exactly as before except that now the check for hash consistency is performed over t randomly chosen views instead on n views. Namely, for a label l^γ in position γ, the verifier checks only $2t$ hash values (t for the left child, and t for the right child), in positions $p_1, \ldots, p_t$, which are chosen uniformly at random.

It follows from Lemma 1 that the VSS of different values must differ in at least $(n - t)$ views. Hence, because t views are randomly selected by V, with probability at least $1 - \frac{1}{t^t}$, a collision will be detected in this hybrid as well, with overwhelming probability.

C.1.2 Hybrid 2. (Breaking t-robustness of MPC).

In this hybrid we consider an oblivious verifier who refrains from looking at all views opened by P^*, instead it looks only to t (randomly chosen) views. (Except for the leaf in position q for which V will check the property ϕ by looking at all the views.) In this hybrid V is not able to verify the innode property by itself. Therefore, P^* additionally computes and sends views of MPC-in's players, and V checks that such views are correct and consistent.

Let $p_1, \ldots, p_t$ be the randomly chosen positions that V picks. (Note that, at this stage, we are still assuming that the prover reveals the path in the clear, thus P^* is not aware of positions $p_1, \ldots, p_t$ which are actually checked by V). In this hybrid a path is **accepting** if:

1. Hash consistency. For each label l^γ, where γ is a position along the path between the root and leaf q, it holds that $l^\gamma[p_j] = h(P_{p_j}^{\mathsf{vss}^{\gamma 0}})$ and $l^\gamma[p_j + n] = h(P_{p_j}^{\mathsf{vss}^{\gamma 1}})$, with $j = 1, \ldots, t$.
2. Node consistency (innode property). For each node γ in the path between position q and the root, the verifier consider player $P_{p_j}^{\mathsf{in}^\gamma}$ of MPC-in checks that: 1) input consistency: check that player $P_{p_j}^{\mathsf{in}^\gamma}$ plays with input: $l^\gamma[p_j]$, $l^\gamma[p_j + n]$, $P_{p_j}^{\mathsf{vss}^\gamma}$, $P_{p_j}^{\mathsf{depth}^\gamma}$. 2) Views consistency. Check that the opened views are consistent with the honest execution of the protocol and with each other. 3) Output correctness. Check that the output of all the revealed views is 1.

3. Value consistency (proof of predicate). V runs the VSS-reconstruction for leaf VSS^q, and obtains value s_q such that $\phi(s_q)$ is true.

The difference between this hybrid and the previous one, is only in the fact that the consistency checks are performed indirectly, via checking protocol $\mathsf{MPC\text{-}in}$.

Assume that, in this hybrid P^* opens $\mathsf{path0}$ and $\mathsf{path1}$ for bits $s_0 \neq s_1$. As discussed before, for $s_0 \neq s_1$ it must hold that $[\mathsf{VSS}^q]_{\mathsf{path0}}$ and $[\mathsf{VSS}^q]_{\mathsf{path1}}$ must differ in at least $n-t$ views, which, due to the innode property, propagates up to the tree leading to a collision. However, now P^* can cheat in the proof of consistency by cheating in the VSS and protocol $\mathsf{MPC\text{-}in}$, and possibly replace the values of a label l^γ along the path without having to change the corresponding VSS^γ (or viceversa). We argue that, due to the t-robustness of the MPC protocols being used, if P^* cheats in any of the MPC-protocols, it will be caught with overwhelming probability over the choice of the players $p_1, \ldots, p_t$.

Fix a node γ, the values sent by P^* for such node are:

- label $l^\gamma = [h_1^{\gamma 0}, \ldots, h_n^{\gamma 0}, h_1^{\gamma 1}, \ldots, h_n^{\gamma 1}]$.
- VSS views sharing the label. $\mathsf{VSS}^\gamma = [P_1^{\mathsf{VSS}^\gamma}, \ldots, P_n^{\mathsf{VSS}^\gamma}]$.
- VSS views sharing the depth of the tree. $\mathsf{VSS}^{\mathsf{depth}} = [P_1^{\mathsf{depth}}, \ldots, P_n^{\mathsf{depth}}]$.
- MPC views. $[P_1^{\mathsf{in}^\gamma}, \ldots, P_n^{\mathsf{in}^\gamma}]$.

Observe that, due to the collision-resistance property of $\mathcal{H}$ we have established that fixed a root, the views of all nodes of the real tree are fixed.

Recall that functionality $\mathcal{F}_{\mathsf{innode}}$ outputs 1 iff one of this conditions is true.

1. $\mathtt{Recon}(P_1^{\mathsf{VSS}^\gamma}, \ldots, P_n^{\mathsf{VSS}^\gamma}) = [h_1^{\gamma 0}, \ldots, h_n^{\gamma 0}, h_1^{\gamma 1}, \ldots, h_n^{\gamma 1}]$.
2. $\mathtt{Recon}(P_1^{\mathsf{depth}}, \ldots, P_n^{\mathsf{depth}}) \geq |\gamma|$.

V will look at t positions and must perform the following checks: (1) The **input** of players $P_i^{\mathsf{in}^\gamma}$ should be $P_i^{\mathsf{VSS}^\gamma}$, $\mathrm{P}_i^{\mathsf{depth}}, h_i^{\gamma 0}, h_i^{\gamma 1}$; (2) The **output** of players $P_i^{\mathsf{in}^\gamma}$ should be 1.

Assume that node γ is not consistent, namely it holds that: $\mathtt{Recon}(P_1^{\mathsf{VSS}^\gamma}, \ldots, P_n^{\mathsf{VSS}^\gamma}) \neq [h_1^{\gamma 0}, \ldots, h_n^{\gamma 1}]$ AND $\mathtt{Recon}(P_1^{\mathsf{depth}}, \ldots, P_n^{\mathsf{depth}}) \leq |\gamma|$, and hence an honest execution of $\mathsf{MPC\text{-}in}$ of input (VSS^γ, $\mathsf{VSS}^{\mathsf{depth}}$, $h_1^{\gamma 0}, \ldots, h_n^{\gamma 1}$) should output 0 to all players $P_i^{\mathsf{in}^\gamma}$. Toward a contradiction, assume that V instead accepts the views $P_{p_j}^{\mathsf{in}^\gamma}$ for $j \in [t]$ for protocol $\mathsf{MPC\text{-}in}$.

It follows that one the following two cases has happened.

Case 1. P^* runs a possibly honest execution of protocol $\mathsf{MPC\text{-}in}$ but players of $\mathsf{MPC\text{-}in}^\gamma$ have as secret input VSS views that are different from the views of VSS^γ, and on such views the output of $\mathcal{F}_{\mathsf{innode}}$ is 1.

Namely, P^* runs $\mathsf{MPC\text{-}in}$ on input the VSS views $P_1^{\mathsf{VSS}^{*\gamma}}, \ldots, P_n^{\mathsf{VSS}^{*\gamma}}$ and $h_1^{*\gamma 0}, \ldots, h_n^{*\gamma 1}$ which do not all correspond to the VSS views $P_1^{\mathsf{VSS}^\gamma}, \ldots, P_n^{\mathsf{VSS}^\gamma}$ of the node, but such that $\mathcal{F}_{\mathsf{innode}}$ outputs 1 to all players.

We argue that, in order for players of $\mathsf{MPC\text{-}in}$'s to output 1 instead, it should be that the number of views $P_i^{\mathsf{VSS}^{*\gamma}}$ used as input in $\mathsf{MPC\text{-}in}$ that differ from the once used to compute the tree $P_i^{\mathsf{VSS}^\gamma}$, are $(n-t)$. Since V randomly choose t views, with high probability V will choose a player P_j^{in} which has in input a view $P_j^{\mathsf{VSS}^{*\gamma}}$ that does not correspond to $P_j^{\mathsf{VSS}^\gamma}$.

Case 2. P^* cheats in the execution of the MPC-in-the-head. In this case, we assume that P^* cheats in the execution of the MPC-in-the-head protocols $\mathsf{VSS}, \mathsf{MPC\text{-}in}$ and lead the players to an incorrect output.

If P^* cheats in the execution of $\mathsf{MPC\text{-}in}$, then there must be views which are inconsistent (see Def. 2) with an honest execution of $\mathsf{MPC\text{-}in}$.

Informally, there are two cases: (1) P^* corrupts $\leq t$ players. Let the set of corrupted players be T. Now, fixed the input of the uncorrupted players $\{P_i^{\mathsf{VSS}^{*\gamma}}\}_{i\in\bar{T}}$, for any tuple of inputs of the corrupted players $\{z_j^*\}_{j\in T}$, it holds that $\mathtt{Recon}(\{P_i^{\mathsf{VSS}^{*\gamma}}\}_{i\in\bar{T}}, \{z_j^*\}_{j\in T}) \neq [h_1^{\gamma 0}, \ldots, h_n^{\gamma 1}]$, and the output of $\mathcal{F}_{\mathsf{innode}}$ is therefore 0. (Note that, the same argument holds for views P_i^{depth}, but we omit it for simplicity). Due to the t-robustness of $\mathsf{MPC\text{-}in}$, by only corrupting t-players, P^* cannot influence the output of the honest player, who will still output 0. However, the verifier misses an honest player with probability $\sim \left(\frac{t}{n}\right)^t$, which is negligible. (2) P^* corrupts $> t$ players. In this case, many corruptions will generate many inconsistent views, and thus the verifier will pick at least one of this views whp (this follows from the fact that $n = ct$).

The formal argument follows precisely the one shown in [IKOS07].

Corrupted players do not follow the protocol, therefore they cause inconsistent views. Two views V_i and V_j are inconsistent if some incoming message from P_j registered in view V_i is different from the outgoing message for P_i implicit in the view V_j.

Given views $P_1^{\mathsf{in}^\gamma}, \ldots, P_n^{\mathsf{in}^\gamma}$, consider the *inconsistency graph* where there is one vertex for each view, and there exists a edge between node i and node j if views $P_i^{\mathsf{in}^\gamma}$ and $P_j^{\mathsf{in}^\gamma}$ are inconsistent.

There are two cases.

Case $\leq t$. There exists in G a vertex cover [6] B of size t. This means that there are t nodes that cover all the inconsistencies. Due to the t-robustness of $\mathsf{MPC\text{-}in}$ and VSS, it holds that with only t inconsistencies the uncorrupted players will still output the correct output of $\mathcal{F}_{\mathsf{innode}}$, which is 0. Now the probability that V accepts is equal to the probability that V misses any uncorrupted players, which corresponds to the probability of picking only views of the set B, which is at most $(t/n)^t$.

Case $> t$. There exists a in G a vertex cover B of size $> t$. This means that G has a matching[7] In this case, instead of looking at the vertices of B, we look at the *edges* of M. Now, if the verifier picks at least one edge of M, it will reject (the reason is that an edge (i, j) in G exists iff V_i and V_j are inconsistent).

The probability that the t vertices that the verifier picks miss all the edges of G is smaller than the probability that they miss all vertices in the matching, which is $2^{\Omega(t)}$. (To see why, supposes that the first $t/2$ vertices chosen by V are $i_{j_1}, \ldots, i_{j_{t/2}}$ and they are all not in M, then the $t/2$ matching neighbours have probability $\Omega(t/n) = \Omega(t/ct) = \Omega(1)$ of being hit by each subsequent vertex i_j).

C.1.3 Hybrid 3. (Adding Fake Nodes)

In this hybrid V does not know the actual depth ℓ_s of the Merkle tree. Therefore, here V sends a sequence of queries $q_1, \ldots, q_{\ell_d}$, one for each possible depth. The difference with the previous

[6] For a graph G, a vertex cover is a set of vertices that cover all edges. Namely, a vertex cover is a set B of vertices such that, each edge of the graph is incident to at least one vertex of B.

[7] For a graph G, a matching is a set of edges that have no vertices in common.

hybrid is in the number of nodes/MPC-in-the-head computed by the prover. Additionally, V checks property ϕ by looking at the views of players $\mathsf{MPC}\text{-}\phi$, as the leaf level of the real tree is unknown in this hybrid. More in detail, the differences with the previous hybrid are:

1. Here the prover sends one path for each query, namely it opens ℓ_d paths. We only require that the path for query q_{ℓ_s} is consistent (where ℓ_s is committed in $\mathsf{VSS}^{\mathsf{depth}}$).

2. The prover must open paths which are longer than the real tree. Crucially, for all nodes lying on level $\ell > \ell_s$, no property must be satisfied (except the hash consistency with the parents).

In this hybrid soundness depends on the robustness of the MPC protocols $\mathsf{MPC\text{-}in}, \mathsf{MPC}\text{-}\phi, \mathsf{VSS}$, and therefore it follows the same arguments of hybrid 2.

Assume that P^*, for the same root l^λ, $[P_1^{\mathsf{depth}}, \ldots, P_n^{\mathsf{depth}}]$ is able to open to two sets $\{\mathsf{path0}_i\}_{i\in[\ell_d]}$ and $\{\mathsf{path1}_i\}_{i\in[\ell_d]}$, which are accepting. In this hybrid a set of paths is accepting if all paths of the set are accepting (according to the same checks performed above). Now note that, $\mathsf{path0}$ and $\mathsf{path1}$ could have many paths in which they differs (indeed, P^* has the freedom of creating fake nodes arbitrarily). But, because the root l^λ is fixed, and the size of the tree is committed in $\mathsf{VSS}^{\mathsf{depth}}$, then it follows from the same argument of hybrid 2, that if $\mathsf{path0}$ and $\mathsf{path1}$ are both accepting, then the path for query q_{ℓ_s} is the same in both $\mathsf{path0}$ and $\mathsf{path1}$.

Therefore, hybrid 3 is indistinguishable from hybrid 2.

C.1.4 Hybrid 4. (Breaking Binding)

In this hybrid all the paths and auxiliary MPC protocols are committed. Thus V explicitly asks for positions $p_1, \ldots, p_t$ and P decommits the corresponding views.

Precisely, P^* sends $\tau = \mathsf{C}_\lambda, \mathsf{C}_{\mathsf{level}}$ in the first step. Then, after seeing queries $q_1, \ldots, q_{\ell_d}$ it sends commitment to each path and $\mathsf{MPC\text{-}in}, \mathsf{MPC}\text{-}\phi$ protocols. Finally, after receiving positions $p_1, \ldots, p_t$ from the verifier, it opens the corresponding views for each commitment.

The only difference with the previous hybrid is in the fact that now P^* knows the positions checked by the verifier and it could open the commitments to the corresponding views adaptively (in which case the argument using the t-robustness of the MPC protocols does not hold). It follows from the computationally binding property of the underlying commitment scheme that hybrid 4 is indistinguishable from hybrid 3.

This hybrid corresponds to the real protocol execution. This completes the proof.

□

C.2 Witness Indistinguishability

Witness indistinguishability basically follows from the perfect t-privacy of the MPC-in-the-head protocols in use, and from the statistically hiding property of the commitment scheme. However, to prove the claim, we additionally need the fact that statistically-hiding commitments imply security in presence of Selective Opening Attacks under the indistinguishability definition of [Hof11].

We first explain why statistically hiding and perfect t-privacy alone are not sufficient to argue about WI, but we need a commitment scheme scheme that is IND-SO-COM secure (Def. 8). Intuitively, we would like to argue the following. A malicious verifier sees only t views of the MPC protocols. Due to the perfect t-privacy of the MPC protocols, given t views, any vector of inputs (which can explain the output of the t views revealed) for the remaining $n - t$ players is equally likely.

However, in our case, our verifier see t views and $(n-t)$ commitments of views. Now we would like to argue that, since commitments are statistically hiding, the verifier should not get any advantage from having such commitments it her view. Namely, we need to argue that the hiding of the unopened commitment is still preserved given that some commitments have been opened. For such argument the definition of hiding is not sufficient, when the messages committed are related (in our case the messages committed are views of an MCP protocol, and therefore are related).

This is precisely the setting of selective opening attack. Therefore, we employ the definition of hiding in presence of Selective opening Attacks (IND-SO-COM) introduced in [Hof11] (see Def. 8). Such indistinguishability definition is sufficient for us since we are only interested in proving that our construction is witness indistinguishable.

Summing up, to argue witness indistinguishability, we show that, for any malicious verifier V^* that distinguishes the witness used by the prover with some non-negligible advantage ϵ, there exists an adversary $\mathsf{A}^{\mathsf{soa}}$ that distinguishes the real and ideal experiment with related probability.

Theorem 4. *If* SHCom *is a statistically-hiding commitment scheme, and* $\mathsf{VSS}, \mathsf{depth}$ *are perfect t-private VSS scheme,* $\mathsf{MPC\text{-}in}$ *t-perfectly realizes* $\mathcal{F}_{\mathsf{innode}}$*, and* MPC-ϕ *t-perfectly realizes* $\mathcal{F}_{\phi}$*, then Protocol 3 is statistically witness indistinguishable.*

Proof. Assume not, then for a property ϕ, there exist a V^*, witnesses w^0, w^1, auxiliary input z, a distinguisher $\mathcal{D}$ such that:

$$\mathsf{Adv}^{\mathsf{WI}} = \text{Prob}[\mathcal{D}(\phi, z, \langle P(w^0), V^*(z)\rangle = 1] - \text{Prob}[\mathcal{D}(\phi, z, \langle P(w^1), V^*(z)\rangle = 1] \geq \epsilon$$

We first use the following Lemma from [Hof11], to assume that SHCom is IND-SO-COM secure.

Lemma 2 (Statistically hiding schemes are IND-SO-COM secure. [Hof11])**.** *Fix arbitrary n and* $\mathcal{I}$ *as in Definition 8, and let* (C, R) *be a statistically hiding commitment scheme. Then* (C, R) *is indistinguishable under selective openings in the sense of Definition 8.*

Then, following the proof technique of [Hof11], we show how to reduce a WI adversary V^* to an adversary of the IND-SO-COM experiment (Def. 8). Adversary $\mathsf{A}^{\mathsf{soa}}$ is running in the IND-SO-COM experiment, and has to distinguish whether it corresponds to the real or the ideal game. For convenience we recall the IND-SO-COM experiment. First, a sampling algorithm $\mathcal{M}$ is invoked, on input an arbitrary auxiliary input. $\mathcal{M}$ samples the vector of messages $\mathbf{M}$ according to some distribution, and outputs $\mathbf{M}$ to the experiment. The experiment commits to each message of such vector, using the commitment scheme that is claimed to be IND-SO-COM secure. The commitments are sent to $\mathsf{A}^{\mathsf{soa}}$. Upon receiving the commitments, $\mathsf{A}^{\mathsf{soa}}$ selects a set of of indexes I, and sends I to the experiment. The experiment answers by opening the commitments in positions I. The remaining messages instead are sent directly without decommitment information. In the real game, the experiment sends the messages $\mathbf{M}$ obtained from the $\mathcal{M}$ in the first step. In the ideal game, the experiments *resamples* the vector conditioned on the messages opened before. Namely, it picks $\mathbf{M}' \leftarrow \mathcal{M}|M_I$, and sends $\mathbf{M}'$ to $\mathsf{A}^{\mathsf{soa}}$. The adversary $\mathsf{A}^{\mathsf{soa}}$ wins if it is able to tell a part the real from the ideal experiment with non-negligible probability.

To show a reduction, we have to define the sampling algorithm $\mathcal{M}$ and the *resampling* strategy $\mathcal{M}|M_I$. We first provide an informal discussion of both algorithms, and of the strategy of $\mathsf{A}^{\mathsf{soa}}$. Then we provide the formal specification, and analysis.

Recall that by assumption we have a verifier V^*, witnesses w^0, w^1 of possibly different size, such that for a property ϕ, an execution between P and V^* where prover uses witness w^0 is distinguishable from an execution in which the prover uses witness w^1.

For this proof we set $d = \max(|w^0|, |w^1|) = poly(n)$. Thus, d is a fixed polynomial instead of being a super-polynomial $n^{\log n}$. Note that this is without loss of generality, and allows a cleaner proof. At the end of the proof we explain of to extend it to the case in which d is super-polynomial.

Sampling Algorithm $\mathcal{M}$. Let d be the size of the longest string between w_0 and w_1. The sampling algorithm takes as input $z' = (w^0, w^1, d, h)$ and $N = poly(d)$, and essentially follows the procedure of the honest prover P on input (w^b), for a randomly chosen b. It compute the real tree, running $\mathsf{BuildRealTree}(w_b, h)$. Then it extends the real tree so that it reaches the length ℓ_d (unless $|w_b| = d$). This is done by running the same procedure of prover P except that, while the prover does it only for few paths (requested by V), here the extension is computed for each node lying on level ℓ_d. Because d is polynomial, the extension takes also polynomial time. For each node, the sampling algorithm prepares views of $\mathsf{MPC\text{-}in}$ and MPC-ϕ.

Note that, the extension of the tree is needed to make the message vector $\mathbf{M}$ of the same length, regardless of the witness used. Indeed, if this was not that case, $\mathsf{A}^{\mathsf{soa}}$ can immediately distinguish which bit is used by the $\mathcal{M}$ algorithm, just looking at the numbers of commitment that she receives. Finally, $\mathcal{M}$ outputs $\mathbf{M}$, which consists of all nodes and labels of the real tree, fake nodes (in case the witness chosen was shorter). Additionally it contains executions of $\mathsf{MPC\text{-}in}$ and MPC-ϕ for each node. Finally it runs VSS for the size of the real tree. The last message of $\mathbf{M}$ is the bit b chosen above. The vector $\mathbf{M}$ is sent to the IND-SO-COM experiment.

Re-sampling Algorithm $\mathcal{M}|M_I$. As it is already clear, the commitments obtained by $\mathsf{A}^{\mathsf{soa}}$ will be parsed as commitments of nodes (and the corresponding instance of $\mathsf{MPC\text{-}in}$) of the tree representing w^b. In the reduction, $\mathsf{A}^{\mathsf{soa}}$ (following the malicious V^*) will ask for the opening of t views, for each node of some of the paths. The openings that $\mathsf{A}^{\mathsf{soa}}$ will receive are in the set M_I. Due to the perfect t-privacy of such MPC-protocols, it holds that given to the adversary the views of t players (and therefore their inputs/outputs), any vector of inputs to the other players that *is consistent* with the t outputs observed by the adversary, is equally likely. Namely, fixed t views, if there exist two input vectors (and therefore views) for the remaining $n - t$ players that are consistent with the output of the t opened views, then both vectors are equally likely. Given that such views exist, in exponential time [8], one can compute them.

This is precisely what the re-sampling algorithm does. $\mathcal{M}|M_I$ works as follows. It picks a random bit σ, and then reconstructs the tree, conditioned on the views contained in M_I and w_σ. Just as an example, consider a sequence of t views $\in M_I$ that represent the VSS part of a node of level $\ell_{|w_\sigma|}$ (the leaf of the real tree for w_σ). Such t views must be justified as views of players that secret share a bit of w_σ. Let δ be such bit. The algorithm will take the t views and computes the remaining $(n-t)$ views that are consistent with one already opened, and such that the reconstruction algorithm applied on any of the n views, outputs the string δ.

Summing up, the re-sampling algorithm just computes the values $\mathbf{M}^*|M_I$ so that they are consistent with an honest execution of the prover on input the witness w_σ.

[8] Note that we are proving statistically WI, so the adversary, and therefore the experiment can be unbounded.

The adversary $\mathsf{A}^{\mathsf{soa}}$**.** The adversary $\mathsf{A}^{\mathsf{soa}}$ essentially simulates the honest prover to V^*, using the commitments (and later the openings) received from the experiment. Specifically, $\mathsf{A}^{\mathsf{soa}}$ first receives the commitments $\mathbf{C}$ for the entire (possibly extended) real tree, and it forwards only the commitment of the root and of $\mathsf{VSS}^{\mathsf{depth}}$ to V^*.

When V^* sends queries $q_1, \ldots q_{\ell_d}$, $\mathsf{A}^{\mathsf{soa}}$ proceeds as follows. From $\mathbf{C}$ it selects the committed paths corresponding to the queries, along with the commitment of the executions of $\mathsf{MPC\text{-}in}$, MPC-ϕ.

Then, when V^* picks the t positions $p_1, \ldots, p_t$ to be opened for each node and MPC, $\mathsf{A}^{\mathsf{soa}}$ computes the set of indexes I accordingly. Namely, such that it obtains the openings for t views for each node (and $\mathsf{MPC\text{-}in}, \mathsf{MPC}$-$\phi$) on the paths previously forwarded to V^*. $\mathsf{A}^{\mathsf{soa}}$ obtains from the experiment the openings for the selected views.

Additionally, $\mathsf{A}^{\mathsf{soa}}$ receives the vector $\mathbf{M}$ containing the views of all the nodes in the clear, but without decommitment information.

By looking at $\mathbf{M}$ $\mathsf{A}^{\mathsf{soa}}$ obtains the witness that is claimed to be used to compute the real path, let us denote it by w_{b^*}.

When V^* terminates, $\mathsf{A}^{\mathsf{soa}}$ runs the distinguisher $\mathcal{D}$. Let b the output of $\mathcal{D}$. $\mathsf{A}^{\mathsf{soa}}$ then outputs $b^* \oplus b$.

When $\mathsf{A}^{\mathsf{soa}}$ is running in the real world, then $\mathsf{A}^{\mathsf{soa}}$ is simulating the prover identically, thus the output of $\mathsf{A}^{\mathsf{soa}}$ relates to the advantage of $\mathsf{Adv}^{\mathsf{WI}} + 1/2$. (See formal arguments below). When $\mathsf{A}^{\mathsf{soa}}$ is running in the ideal world, the bit b^* is randomly chosen, thus the output of $\mathsf{A}^{\mathsf{soa}}$ is also a random bit.

Thus we conclude that the difference between the output of $\mathsf{A}^{\mathsf{soa}}$ in the real and ideal game, relates to the distinguishing capacity of V^*.

In the following we give the details of sampling algorithm $\mathcal{M}$, the adversary $\mathsf{A}^{\mathsf{soa}}$, the resampling algorithm $\mathcal{M}||M_I$. We finally analyze the advantage of $\mathsf{A}^{\mathsf{soa}}$.

Sampling Algorithm. Algorithm $\mathcal{M}(z', N)$.
On input $z' = (w_0, w_1, d, h)$ and $N = poly(d)$.

1. Toss a coin $b \in \{0, 1\}$.
2. Run $\mathsf{BuildRealTree}(w_b, h)$ and obtain the real tree. Run $\mathtt{Share}(\ell_{w_b})$.
3. Extend the tree with fake nodes till level ℓ_d.
4. Compute $\mathsf{MPC\text{-}in}$ (using the trapdoor condition for the fake nodes) and MPC-ϕ for each node of the tree.
5. define $(M_i)_{i \in N}$ as a concatenation of messages, where a message is either a view of an MPC player (either of VSS or $\mathsf{MPC\text{-}in}$, MPC-ϕ), or an hash of a view (if it is part of the label).
6. define $M_{N+1} = b$.
7. output $\mathbf{M} \leftarrow \{M_i\}_{i \in N+1}$

SOA Adversary. Next we show the reduction. Adversary $\mathsf{A}^{\mathsf{soa}}$ is running the IND-SO-COM experiment. The reduction proceeds as follows. We assume that IND-SO-COM experiment and $\mathsf{A}^{\mathsf{soa}}$ non uniformly obtain h as auxiliary input.

1. Run V^* on input $z' = w^0, w^1, d$. V^* plays h in the first round.
2. Obtain N+1 commitments $\mathbf{C}$= $\mathsf{C}_1, \ldots, \mathsf{C}_{N+1}$ from the IND-SO-COM experiment. $\mathsf{C}_1, \ldots, \mathsf{C}_{N+1}$ are interpreted as commitments for the tree and for executions of $\mathsf{VSS}^{\mathsf{depth}}$, $\mathsf{MPC\text{-}in}$, MPC-ϕ.
3. Forward to V^* the commitment of the root, and the commitment of $\mathsf{VSS}^{\mathsf{depth}}$'s players.
4. Receive $q_1, \ldots q_{\ell_d}$ from the V^*.
5. For each $i = 1, \ldots, \ell_d$, compute path_{q_i} taking appropriates commitments from $\mathbf{C}$.
6. Upon receiving the set of t indexes from V^*, translate this set into the appropriate set I that allows to obtain t views for each VSS and $2t$ parts of the label, and t views for each $\mathsf{MPC\text{-}in}, \mathsf{MPC}$-$\phi$ and for $\mathsf{VSS}^{\mathsf{depth}}$. Send I to the experiment IND-SO-COM.
7. Obtain the openings M_I from the experiment.
8. Obtain the full vector of plaintext $\mathbf{M} = \{M_j\}_{j \notin I}$ from the experiment (*without decommitment*).
9. when the interaction with V^* finishes, let T be the transcript of the interaction between V^* and the simulated prover. Run $b_{V^*} \leftarrow \mathcal{D}(z, T)$.
10. Let $b' = \mathbf{M}'_{N+1}$.
11. output $b' + b_{V^*}$.

Re-sampling algorithm. Algorithm $\mathcal{M}|M_I(z', N)$.

- pick a random bit b, compute w_b and let ℓ_{w_b} the depth of the tree associated to w_b.
- interpret the opened messages M_I as the opening of ℓ_d (partial) paths of the tree computed by $\mathcal{M}$. The goal of the sampling algorithm is to compute the remaining views consistently with a tree for w_b.

  Recall that, the sampling algorithm has computed tree of depth ℓ_d where d is the sizeof the longest string between w_0 and w_1. The vector of opened messages M_I contains the opening of only some of the paths. Specifically, for each node along a path, t views of VSS part and $2t$ string of the label part have been reveled.

  The main goal of $\mathcal{M}|M_I$ here is to fill the gap, and to reconstruct the remaining $(n-t)$ values for each node, *consistently* with a tree for w^b.

  $\mathcal{M}|M_I$ starts from level ℓ_D.

  For $\ell = \ell_d$ to 1. For any $\gamma \in \{0,1\}^\ell$.

  - (Leaf) if $\ell = \ell_{w_b}$ [Compute the nodes at level ℓ_d as real leaves].
    1. Let c the bit of the w_b at position γ.

2. (Label) From M_I take the $2t$ values belonging to the "label part" of node l^γ. Fill the remaining $2(n-t)$ part of l^γ by picking random values.
3. (VSS) From M_I take the t views belonging to the "VSS part" of node VSS^γ. Compute $(n-t)$ views, so that the complete VSS^γ corresponds to the VSS of the value c computed above.
4. (in) From M_I take the t views belonging to the $\mathsf{MPC\text{-}in}$ execution for node γ. Compute the remaining $(n-t)$ views, consistently with the opened views *and* the views computed above.
5. (MPC-ϕ) From M_I take the t views belonging to the MPC-ϕ execution for node γ. Compute the remaining $(n-t)$ views, consistently with the opened views *and* the views computed above, to prove that $\phi(w[\gamma])$ is true.

– (Node) if $\ell < \ell_{w_b}$ [Computes parent nodes according to nodes computed so far]

1. (Parent Node Label) From M_I take the $2t$ values belonging to the "label part" of node l^γ. Compute the remaining $2(n-t)$ part of l^γ by hashing the views of the VSS part of the children.
2. (VSS) From M_I take the t views belonging to the "VSS part" of node VSS^γ. Compute $(n-t)$ views, so that the complete VSS^γ corresponds to the VSS of the string l^γ computed above.
3. ($\mathsf{MPC\text{-}in}$). From M_I take the t views belonging to the $\mathsf{MPC\text{-}in}$ execution for node γ. Compute the remaining $(n-t)$ views, consistently with the opened views *and* the views computed above.
4. (MPC-ϕ) From M_I take the t views belonging to the MPC-ϕ execution for node γ. Compute the remaining $(n-t)$ views, consistently with the opened views *and* the views computed above, to prove that $\ell \neq \ell_{w_b}$ (trapdoor condition).

– (Fake Node) if $\ell > \ell_{w_b}$ [Computes fake nodes]. For $\gamma' \in \{0,1\}^{\ell-1}$.s

1. (VSS children) From M_I take the t views belonging to the "VSS part" of node $\mathsf{VSS}^{\gamma'0}$. Compute $(n-t)$ views, so that the complete $\mathsf{VSS}^{\gamma'0}$ corresponds to the correct VSS of some random string. Do the same for node $\mathsf{VSS}^{\gamma'1}$
2. (Label) From M_I take the $2t$ values belonging to the "label part" of node l^γ. Fill the remaining $2(n-t)$ part of l^γ by computing the hash of the views of nodes $\gamma'0, \gamma'1$ computed above.
3. ($\mathsf{MPC\text{-}in}$). From M_I take the t views belonging to the "VSS part" of node VSS^γ. With the above views, compute the remaining $n-t$ views of $\mathsf{MPC\text{-}in}$ for node γ, by proving the trapdoor condition $\ell > \ell_{w_b}$.
4. (MPC-ϕ) From M_I take the t views belonging to the MPC-ϕ execution for node γ. Compute the remaining $(n-t)$ views, consistently with the opened views *and* the views computed above, to prove that $\ell \neq \ell_{w_b}$ (trapdoor condition).

– Compute views of $\mathsf{VSS}^{\mathsf{depth}}$. From M_I take the t views belonging to VSS protocol. Compute $(n-t)$ views so that the final n views are an honest execution of the VSS of ℓ_{wi_b}.

- Output: all the messages computed above, and output $\mathbf{M}'_{N+1} = \sigma$

Analysis of $\mathsf{A}^{\mathsf{soa}}$ advantage. By hypothesis assumptions, there exists a circuit V^* and a distinguisher $\mathcal{D}$ such that,

$$\mathsf{Adv}^{\mathsf{WI}} = \text{Prob}[\mathcal{D}(z, \langle P(w_0), V^*(z)\rangle = 1] - \text{Prob}[\mathcal{D}(z, \langle P(w_1), V^*(z)\rangle = 1] \geq \epsilon$$

We analyze how the adversary $\mathsf{A}^{\mathsf{soa}}$ can exploit the distinguishing capacity of $\mathcal{D}$.

Advantage in the real experiment When $\mathsf{A}^{\mathsf{soa}}$ is playing in the real experiment, then the following happens. If $\mathcal{M}$ chose $b = 0$, then the real tree is computed for w_0 and the $\mathsf{A}^{\mathsf{soa}}$ correctly simulates an interaction between $P(w_0)$ and V^*. Since $\mathbf{M}'_{N+1} = 0$, consequently $\mathsf{A}^{\mathsf{soa}}$ outputs $\mathcal{D}(z, \langle P(w_0), V^*(z)\rangle$. Conversely, if $b = 1$, then $\mathsf{A}^{\mathsf{soa}}$ is simulating an interaction between $P(w_1)$ and V^*. Hence, since $\mathbf{M}'_{N+1}$, $\mathsf{A}^{\mathsf{soa}}$ outputs $1 - \mathcal{D}(z, \langle P(w_1), V^*(z)\rangle$.

Thus, probability that $\mathsf{A}^{\mathsf{soa}}$ outputs 1 in the real experiment is

$$\begin{aligned} \big|\text{Prob}[\mathbf{Exp}^{\mathsf{ind\text{-}so\text{-}real}}_{\mathsf{Com},\mathcal{M},A}(n) = 1] &= \frac{1}{2}(\text{Prob}[\mathcal{D}(z, \langle P(w_0) = 1 + \\ &= 1 - \text{Prob}[\mathcal{D}(z, \langle P(w_1) = 1] \\ &\leq \frac{1}{2} \cdot \mathsf{Adv}^{\mathsf{WI}} + \frac{1}{2}. \end{aligned} \tag{1}$$

Advantage in the ideal experiment In the ideal experiment, the resampling algorithm computes the bit $\mathbf{M}'_{N+1}$ by sampling uniformly at random. This is possible due to the t-privacy of the MPC protocols run by the prover. Indeed, since t views can be used to explain both witness w_0, w_1. Namely, messages M_I seen by $\mathsf{A}^{\mathsf{soa}}$ are equally likely to appear, regardless of the witness used. Since $\mathbf{M}'_{N+1}$ is a freshly tossed coin, we get:

$$\text{Prob}[\mathbf{Exp}^{\mathsf{ind\text{-}so\text{-}ideal}}_{\mathsf{Com},\mathcal{M},A}(n) = 1] = \frac{1}{2}$$

It follows that,

$$\mathbf{Adv}^{\mathsf{ind\text{-}so}}_{\mathsf{Com},\mathcal{M},A} = \big|\text{Prob}[\mathbf{Exp}^{\mathsf{ind\text{-}so\text{-}real}}_{\mathsf{ExCom},\mathcal{M},A}(n) = 1] - \text{Prob}[\mathbf{Exp}^{\mathsf{ind\text{-}so\text{-}ideal}}_{\mathsf{ExCom},\mathcal{M},A}(n) = 1]\big| = \frac{1}{2} \cdot \mathsf{Adv}^{\mathsf{WI}}$$

which contradicts the assumption that ExCom is IND-SO-COM secure.

Case d is super poly. If $d = n^{\log n}$ then the sampling algorithm cannot compute a full tree of length ℓ_d as it would have exponentially many leaves. So, the sampling algorithm will compute a tree of length D, with $D = \max(|w^0|, |w^1|)$. Thus, $\mathsf{A}^{\mathsf{soa}}$ receives commitments only for paths of length ℓ_D.

To answer to the queries of length ℓ_d that will be asked by V^*, $\mathsf{A}^{\mathsf{soa}}$ simply extends the tree received from the experiment with commitment of zeros. Later $\mathsf{A}^{\mathsf{soa}}$ opens such commitments adaptively on the revealed messages M_I received from the experiment, by breaking the computational hiding of *its own* commitments. □

D Black-Box Witness Indistinguishable Universal Argument

A WI UARG protocol can be seen as a commit-and-prove protocol in which P commits to a PCP oracle π, V computes the positions of the oracle that must be checked by running algorithm $\mathsf{Q}_{\mathsf{pcp}}$, and P proves that, had it opened such positions, the decision algorithm associated to the PCP Verifier, denoted by $\mathsf{D}_{\mathsf{pcp}}$, would have accepted.

Hence, Black-box WIUARG are straightforwardly implemented by properly instantiating the construction outlined in Sec. 3.2 (formally described in Prot. 3) as follows. Algorithm $\mathsf{GetIndex}$ is instantiated with algorithm $\mathsf{Q}_{\mathsf{pcp}}$ used by the PCP verifier to select the queries. The property ϕ here is defined over multiple leaves, and it satisfied iff algorithm $\mathsf{D}_{\mathsf{pcp}}$ executed over the values of the leaves output 1. More formally, such property is denote by $\phi\text{-}\mathsf{D}_{\mathsf{pcp}}(y,\cdot)$, and is defined over instances $y \in \mathcal{L}_{\mathcal{U}}$ and a set of indexes I, and it is true iff $\mathsf{D}_{\mathsf{pcp}}(y,\{\pi_i\}_{i\in I}) = 1$. Finally, here we use *extractable* SH commitment scheme (instead of just SH), in order to obtain the weak-proof-of-knowledge property. To highlight this fact we use notation $\mathsf{BBExtCom}$ and $\mathsf{BBExtProve}$.

Protocol 5. *Black-Box Witness Indistinguishable Universal Argument.*

Common Input: $y = (M, x, \mathtt{t}) \in \mathcal{L}_{\mathcal{U}}$, and $d = poly(\mathtt{t})$. Auxiliary Input to P: w such that $(y, w) \in \mathcal{R}_{\mathcal{U}}$.

$P \to V$: **Commitment of the PCP.**

1. Run M on input (w, x) and obtain $\mathtt{t}' = T_M(x, w)$. Let $w' = (w, \mathtt{t}')$ such that $(y, w') \in \mathcal{R}'_{\mathcal{U}}$.
2. Invoke $\mathsf{P}_{\mathsf{pcp}}$ on input (y, w') and obtain the proof π.
3. Run $\mathsf{BBExtCom}(h, \pi)$ to obtain $\mathsf{ExC}^{\pi}{}_0$, $\mathsf{ExC}^{\mathsf{lev}\pi}{}_1$, ..., $\mathsf{ExC}^{\mathsf{lev}\pi}{}_n$ and send it to V.

$V \to P$: **Queries.**

Uniformly select random tapes $r_1, \ldots, r_{\ell_d}$ and send them to P. P and V run $I_l \leftarrow \{\mathsf{Q}_{\mathsf{pcp}}(y, r_l, i)\}_{i\in[k]}$ for $l \in [\ell_d]$. Obtain queries $\mathcal{I} = \{I_1, \ldots, I_{\ell_d}\}$.

$P \leftrightarrow V$: **Proof.** P and V run $\mathsf{BBExtProve}(I_1, \ldots, I_{\ell_d})$ for property $\phi\text{-}\mathsf{D}_{\mathsf{pcp}}$. V accepts iff the verifier of $\mathsf{BBExtProve}$ accepts.

D.1 Security of Our Black-box WI Universal Argument

In this section we formally prove the following theorem:

Theorem 5. *If Protocol 3 is a Size-Hiding BB commit-and-prove protocol with parameter $d = poly(\mathtt{t})$, wrt property $\phi\text{-}\mathsf{D}_{\mathsf{pcp}}$, then Protocol 5 is a Black-Box Witness Indistinguishable Universal Argument system.*

Soundness. Soundness follows almost directly from the soundness of the commit-and-prove protocol, except that – as stated in Remark 3 – here we have to related the probability of proving a false theorem with the probability of finding a collision in Hybrid 1. Additionally we have to show a oracle-recovery-procedure for the weak proof of knowledge property.

In the following we show how to relate the non-negligible probability of success of P^* in proving a false theorem to the probability of *finding* a collision in polynomial time, by using the analysis provided [BG08].

Assume, toward a contradiction, that there exists a P^* and a string $y = (M, x, \mathtt{t}) \notin \mathcal{L}_{\mathcal{U}}$, and a polynomial $p = p(|y|)$ such that P^* convinces V with probability $> 1/p$. Because $y \notin \mathcal{L}_{\mathcal{U}}$, from the soundness of the PCP in use (which is 2^{-n}), it holds that with overwhelming probability over the coin tossed by V when running $\mathsf{Q}_{\mathsf{pcp}}$, there exists no proof π which will be accepted by the decision algorithm $\mathsf{D}_{\mathsf{pcp}}$.

Now, recall Hybrid 1 in Sec. C.1. In such hybrid we show that, fixed a commitment transcript τ, *if* a query q is conflicting, i.e., P^* was able to open two accepting paths, path0 and path1 for conflicting values v_0, v_1, then we can transform such P^* in a collision forming circuit (thus breaking the collision resistance of h). In this proof we show how to relate the probability of success of P^* proving a false theorem to the probability of finding a conflicting query q. We prove this by following the analysis [BG08] for the computational soundness of their universal argument.

D.1.1 Extending Hybrid 1 presented in C.1.1

As we are hybrid 1, here we consider a version of Protocol 5 in which the prover sends everything in the clear. Thus, in this hybrid the size ℓ_s is know to V and in the query phase only $r = r_{\ell_s}$ will be sent.

Fix any polynomial p. Fix an arbitrary family, $\{P_n^*\}_{n \in N}$, of (deterministic) polynomial-size circuits, a generic n and $y \in \{0,1\}^n$, such that $Pr[(P_n^*, V)(y) = 1] > \epsilon \stackrel{def}{=} 1/p(n)$.

First, we establish some key notions regarding the interaction between P_n^* and V that are used in the proof.

- The i-th q_i is the value $q_i = \mathsf{Q}_{\mathsf{pcp}}(y, r, i)$.
- The i-th answer is the bit reconstructed from the node in position q_i. Namely, $a_{q_i} = \mathtt{Recon}(P_1^{\mathsf{VSS}^{q_i}}, \ldots, P_n^{\mathsf{VSS}^{q_i}})$.
- The authentication of this answer is the corresponding path path.
- The i-th answer is proper if the corresponding path is accepting (as per Definition 24).

Next, consider the probability distribution induced by verifier's choice of h and r. Fix any $h \in \mathcal{H}$, consider the conditional probability $p_{h,y}$ that V accepts y when choosing h. Given the hypothesis that P_n^* convinces V with probability ϵ, it follows that for at least $\epsilon/2$ of the possible h are such that $p_{h,y} \geq \epsilon/2$. We fix any such h for the rest of the proof.

Now we consider the residual probability induced by the choice of r.

- for a query q, a query index $i \in [n]$, a possible answer $\sigma \in \{0,1\}$, and a parameter $\delta \in [0,1]$, we say that σ is δ-strong for (i, q) if, conditioning on the i-th query being q, the probability that P_n^* *properly answers* the i-th query with σ is at least δ. Namely,

$$Pr[\mathtt{Recon}(P_1^{\mathsf{VSS}^{q_i}}, \ldots, P_n^{\mathsf{VSS}^{q_i}})) = \sigma | q_i = \mathsf{Q}_{\mathsf{pcp}}(y, r, i)] \geq \delta$$

- A query q has δ-conflicting answers if there exists i and j (possibly $i = j$) such that 0 is δ-strong for (i, q) and 1 is δ-strong for (j, q).

We first show that, for the prover to be convincing with non-negligible probability, it should be the case that many of the queries are δ-strong (Claim 1).

Claim 1. *The probability that the verifier accepts while receiving only δ-strong answers is at least $p_{y,h} - k\delta$*

The proof of this claim follows similar arguments provided in [BG08] (see pag. 1674).

Then we show that, among the δ-strong queries, it is rare to find δ-conflicting queries. Indeed, if this was not the case, a noticeable fraction of δ-strong queries are also δ-conflicting, then one can use such queries to find a collision in h (Claim 2)

Hence, fix $\delta = p_{y,h}/2m$, and assume that all prover's answers are δ-strong.

Claim 2. *There exist a PPT oracle machine that, for any δ, given $h \in \mathcal{H}$ and oracle access to P_n^*, finds collisions with respect to h with success probability that is polynomially related to δ/k and to the probability that the verifier make a query that has δ-*conflicting *answers. By denoting as μ_h the probability that the verifier (after choosing h), makes a query that has δ-conflicting answers. Then, probability of finding a collision is at least $\mu_h\delta^2/k^3$.*

Proof. For a uniformly selected $r \in \{0,1\}^{poly(n)}$ and $i \in [k]$, it follows that probability that $q_i = \mathsf{Q}_{\mathsf{pcp}}(y, r, i)$ is δ-conflicting is at least (μ_h/k).

Thus one can, uniformly select $i0, i1 \in [k]$, invoke the reverse-sampling algorithm S_{pcp} on input $(y, i0, q_i)$ and $(y, i1, q_i)$ and obtain uniformly distributed $r0$, $r1$, satisfying $q_i = \mathsf{Q}_{\mathsf{pcp}}(y, r0, i0)$ and $q_i = \mathsf{Q}_{\mathsf{pcp}}(y, r1, i1)$. Running P_n^* as subroutine twice feeding $h, r0$ the first time and $h, r1$ the second time, one obtain $\mathsf{path0}$ and $\mathsf{path1}$. Assuming that q_i is δ-conflicting with probability at least $(\delta/m)^2$, both $\mathsf{path0}$ and $\mathsf{path1}$ are accepting for q_i but opens to conflicting values v_0 and v_1. Thus with probability at least $(\mu_h/k) \cdot (\delta/m)^2$ obtain two different *proper* answers to the same query q_i. At this point one can apply the same arguments of Hybrid 1 C.1.1, to show a collision. □

Thus, on a typical h and for $\delta > 1/poly(n)$, the quantity μ_h must be negligible, otherwise one can derive a contradiction to the collision resistance of $\{h\}$.

Consequently, for $\delta = p_{y,h}/2m > poly(n)$ we will focus on the case that the prover's answer are *not* δ-conflicting.

Establishing Weak Proof of knowledge in Hybrid 1 Claim 1 and Claim 2 establish that, that prover's answers are δ-strong but not $\delta/2$-conflicting. Thus, we can use the such prover to construct an oracle for the PCP system which makes V_{pcp} accept with probability at least $p_{yh}/2$. Next, following [BG08] we show that, if the queries are not conflicting, then we can build an oracle-recovery procedure that recovers the bit of the PCP oracle. Such oracle is then used by the extractor E_{pcp} associated to the PCP in use.

Note that, it is not the case that *all* the prover's answers are δ strong and not $\delta/2$-conflicting. Some queries may have no strong answers or be conflicting. Such cases happen with negligible probability, and when happen the oracle-recovery procedure may fail to recover the entry of the PCP oracle. This is not a problem since we are trying to convince a PCP verifier, and with sufficiently high probability, such verifier will not make those queries.

Oracle-Recovery Procedure: On input (y, h) and a query q, and with oracle access to P_n^*, recover the q-th bit of the PCP as follows. Let $T \stackrel{def}{=} poly(n/\delta)$ and $\delta = \epsilon/4k$.

1. For $i = 1, \ldots, k$ and $j = 1, \ldots, T$, invoke S_{pcp} on input (y, i, q) ad obtain $r_{i,j}$.

2. For $i = 1, \ldots, k$ and $j = 1, \ldots, T$ invoke P_n^* feeding it with h and $r_{i,j}$. If the i-th answer a_i is proper, reconstruct the bit supported by (i, j) by running $\mathtt{Recon}(a_i)$. Record (i, j) as supporting the reconstructed bit.

3. If for some $i \in [k]$, there are $(2\delta/3) \cdot T$ records for the form $(i, \cdot)$ supporting the value $\sigma \in \{0, 1\}$, then identify σ as a candidate.

4. If a single value of σ is defined as candidate, then set the q-th bit of the PCP oracle accordingly. (Otherwise, do nothing).

The above oracle-recovery procedure, is almost identical to the procedure shown in [BG08], except for Step 2. The difference being that in the UA shown in [BG08] the verifier obtains the q-th bit of the PCP directly from the prover, while our verifier obtains only the VSS of such value. Thus, our verifier (and therefore the recovery procedure) obtains the actual bit by running the reconstruction procedure. Due to perfect correctness of the VSS, this step has the same effect of receiving the bit directly from the prover. Therefore, the same analysis of [BG08] holds for our oracle-recovery procedure as well.

Hence, using Claim 3.5.3 of [BG08] (see pag. 1676) we conclude that, if P^* makes the verifier V accept with non-negligible probability, then with similar probability we can reconstruct an oracle that convinces an external PCP verifier with polynomially related probability.

Having such oracle-recovery procedure, [BG08] shows how this procedure can be run as subroutine by the extractor E_{pcp} to extract bits of the witness used by P^*. In this way the weak-proof of knowledge is established. Since the extractor will be identical in our case, with refer the reader to [BG08] (pag 1676) for details.

The remaining of the proof follow identically the proof provided in Section C.1.

D.1.2 Oracle-recovery-procedure in Hybrid 4

As final step we notice that the oracle-recovery-procedure shown previously work in Hybrid 1, were everything is sent in the clear. In hybrid 4, which corresponds to the final protocol, the prover sends the (extractable) commitments of the paths. Therefore we modify the previous procedure as follows. The idea is to modify any P^* for hybrid 4 in a P^* for hybrid 1.

Oracle-Recovery Procedure: On input (y, h), the extractor preliminary extracts the size of the PCP oracle as follows (this is crucial to detect the size of query q). Obtain C_0, C_1, ..., C_n running the commitment phase with P^* on input h. Run the extractor associated to ExCom and obtains value l'^0 and views $[P_1^{\mathsf{depth}}, \ldots, P_n^{\mathsf{depth}}]$. If the extractor fails, abort. Else, run $\mathtt{Recon}(P_1^{\mathsf{depth}}, \ldots, P_n^{\mathsf{depth}})$ to obtain ℓ_s. Output ℓ_s.

Then, on input $q \in \{0, 1\}^{\ell_s}$, and with oracle access to P_n^*, recover the q-th bit of the PCP as follows. Let $T \stackrel{def}{=} poly(n/\delta)$ and $\delta = \epsilon/4k$. Pick randomness $r_1, \ldots, r_{\ell_s - 1}, r_{\ell_s + 1}, \ldots r_{\ell_d}$ to play in the second verifier's step.

1. For $i = 1, \ldots, k$ and $j = 1, \ldots, T$, invoke S_{pcp} on input (y, i, q) ad obtain $r_{i,j}$.

2. For $i = 1, \ldots, k$ and $j = 1, \ldots, T$ invoke residual prover P_n^* with input $r_1, \ldots, r_{\ell_s - 1}, r_{i,j}$, $r_{\ell_s + 1}, \ldots, r_{\ell_d}$. If the i-th answer a_i is proper, reconstruct the bit supported by first extracting views $(P_1^{\mathsf{VSS}^q} \mid \cdots \mid P_n^{\mathsf{VSS}^q})$ from the respective extractable commitments, and then running $(i, j) \leftarrow \mathtt{Recon}(P_1^{\mathsf{VSS}^q} \mid \cdots \mid P_n^{\mathsf{VSS}^q})$. Record (i, j) as supporting the reconstructed bit.

3. If for some $i \in [k]$, there are $(2\delta/3)\cdot T$ records for the form $(i, \cdot)$ supporting the value $\sigma \in \{0,1\}$, then identify σ as a candidate.

4. If a single value of σ is defined as candidate, then set the q-th bit of the PCP oracle accordingly. (Otherwise, do nothing).

Witness Indistinguishability. WI follows straight-forwardly from the WI property of the underlying commit-and-prove Prot. 3.

E Security of Our Public-Coin ZK Argument

In this section we formally prove that Protocol 2 is a Zero Knowledge Argument System.

E.1 Soundness

Lemma 3. *If Protocol 3 is a Size-Hiding BB commit-and-prove protocol wrt ϕ-ZK and with parameter $d = n^{\log n}$, then Protocol 2 is computationally sound.*

Assume that $x \notin L$. For a polynomial $p = poly(n)$, assume that there exists a PPT P^* which convinces the verifier with non negligible $1/p$.

Because $x \notin L$, we have 3 possible cases. Case 1. P^* commits to an encoding of a program Π which actually predicts the value r sent by the verifier in the trapdoor generation phase. Because r is randomly chosen in $\{0,1\}^n$, this event happens with probability 2^{-n}, and thus is negligible. Thus this case can be ignored.

Case 2. $z = \mathsf{BBProve}(\Psi)$ is a commitment of a string Ψ which is not the encoding of a program predicting r. Thus $x \notin L$ and $((z, r, \mathtt{t}), (\Psi)) \notin \mathcal{L}_{\mathcal{P}}$. In this case, due to the δ-soundness of the PCPP in use, and the use of ECC with distance δ, there exists no accepting PCPP oracle for theorem $((z, r, \mathtt{t}), (\Psi))$. Thus, one can show how to relate the probability that P^* convinces V, to the probability of finding conflicting queries for the paths of the extractable merkle tree of the PCPP oracle. This case follows similar arguments shown for soundness of our UAWI and is therefore omitted.

Case 3. $z = \mathsf{BBProve}(Y)$ is a commitment of a string Y which is not the encoding of a program predicting r. Thus $x \notin L$ and $((z, r, \mathtt{t}), (Y)) \notin \mathcal{L}_{\mathcal{P}}$. However, after receiving r, P^* computes a valid PCP of Proximity oracle for theorem $((z, r, \mathtt{t}), (Y')) \notin \mathcal{L}_{\mathcal{P}}$, where $Y' = \mathsf{ECC}(\Pi)$ and Π is a program predicting r. However, $Y' \neq Y$. In this case, due to the δ-soundness of the underlying PCP of Proximity, and the use of ECC with distance δ, one can show how to relate the probability that P^* convinces V, to the probability of finding conflicting queries for the paths of the extractable merkle tree of Y.

In the following we focus on Case 3. As done before, to re-use the soundness proof shown for the underlying commit-and-prove protocol, we need to show how to relate the probability of success of P^* in Case 3, to the probability of detecting a collision in Hybrid 1 C.1.1.

Extending Hybrid 1 (Sec. C.1.1). One can use P^* to create a collision-forming circuit $\mathcal{C}$ as follows. $\mathcal{C}$ runs P^* as subroutine, on input $x, \mathtt{t}$ (where $x \notin L$ by assumption), and a randomly chosen $h \in \mathcal{H}$. P^* answers with $z = (\mathsf{root}_Y, \mathsf{VSS\text{-}LevY})$ where root_Y is the root of the extendable merkle tree computed on Y, and VSS-LevY is the vector of views of the depth players sharing the

size ℓ_Y of Y. Then, $\mathcal{C}$ sends a randomly chosen r. At this point the public theorem $a = (z, r, \mathtt{t})$ is computed. P^* then commits to the PCPP oracle, and $\mathcal{C}$ obtains the pair $(\mathsf{root}_\pi, \mathsf{VSS\text{-}LevP})$ where root_π is the root of the extendable merkle tree computed for the PCPP oracle π and $\mathsf{VSS\text{-}LevP}$ is the VSS of the size of π. Additionally, $\mathcal{C}$ receives $P_1^{\mathsf{VSS\text{-}Witn}}, \cdots, P_n^{\mathsf{VSS\text{-}Witn}}$ from P^*.

Next, $\mathcal{C}$ sends a random ϑ, serving to compute the PCPP queries $(q_1, p_1), \ldots, (q_k, p_k)$ from $\mathsf{Q}_{\mathsf{pcpx}}(a, \vartheta, k)$. With some non-negligible probability p, over the random choice of (h, r, ϑ) P^* has sent an accepting proof which consists in the corresponding paths for the (indeed, by assumption, P^* convince a verifier with non-negligible probability). Now, $\mathcal{C}$ picks $i \in \{1, \ldots, n\}$, hoping that query (q_i, p_i) is conflicting. It considers the complete paths pathY_{q_i}, and $\mathsf{path}\pi_{p_i}$ open by P^* to answer to such a query. It reconstructs bit Y_{q_i} by running $\mathtt{Recon}([P_1^{\mathsf{VSS}^{q_i}}, |\cdots|, P_n^{\mathsf{VSS}^{q_i}}]_{\mathsf{pathY}})$.

Next, to obtain a collision, $\mathcal{C}$ rewinds P^*, up to the point in which P^* has committed to root_Y and $\mathsf{VSS\text{-}LevY}$, and plays with r' (with probability $1 - 1/2^n$ $r \neq r'$). Set $a' = (z, r', \mathtt{t})$. Note that z has not changed. Then it obtains a commitment to a new PCP oracle, and using the efficient sampling algorithm associated to the PCP of Proximity, it compute ϑ' so that $(q_i, p_i) = \mathsf{Q}_{\mathsf{pcpx}}(a, \vartheta', i)$. It then obtains again the completes paths pathY'_{q_i}, $\mathsf{path}\pi'_{p_i}$ for such a query. If the answer is accepting, then $\mathcal{C}$ reconstructs the bit of Y'_{q_i} as before.

Now since $r \neq r'$, it must be the case that $\Pi \neq \Pi'$. Consequently, codewords $Y = \mathsf{ECC}(\Pi)$ should differ from $Y' = \mathsf{ECC}(\Pi')$ in a constant number of position (given that $|Y| = |Y'|$). Since i was randomly chosen, then with non-negligible probability i is one of the positions in which Y and Y' differ.

The remaining part of the proof follows the same arguments shown in Sec. C.1.

E.2 Zero Knowledge

We show a PPT straight-line ZK simulator Sim for Protocol 2. Sim works as follows. In the trapdoor generation phase it commits to the code of V^*. Namely it computes $z = \mathsf{BBCom}(\mathsf{ECC}(V^*))$. This is in contrast with the prover who commits to 0^n. When receiving r from V^*, set the public theorem to be $(z, r, \mathtt{t})$.

In the proof phase Sim computes the PCPP proof π for the public theorem $(z, r, \mathtt{t})$, and with witness V^*. Formally proving $(a = (z, r, \mathtt{t}), (\mathsf{ECC}(V^*)) \in \mathcal{L}_{\mathcal{P}}$. This is in contrast with the prover who commits to 0^n. Finally, it runs $\mathsf{VSS\text{-}WI}$ on string $0^{|w|}$, instead of the real witness. Again, this is in contrast with the prover who commits to the witness w. When executing $\mathsf{BBProve}$, Sim uses the witness $\mathsf{ECC}(\pi), \pi$ to prove that ϕ-ZK is true, instead of using the fact that $(x, w) \in \mathcal{R}_L$. The difference between Sim and P lies in the strings that are committed using the commit-and-prove protocol. Indeed, Sim commits to the triple: $(\mathsf{ECC}(V^*), \pi, 0^n)$ while P commits to: $(0^n, 0^n, w)$.

Note that here we are using the extension of the commit-and-prove protocol over multiple strings, namely Protocol 4. In order to prove indistinguishability of the transcript generated by Sim, we essentially have to prove that also Protocol 4 is WI.

In the following we prove that Witness Indistinguishability of Protocol 2 follows from the IND-SO-COM security of the underlying commitment scheme and the perfect t-privacy of the MPC protocols in use.

E.2.1 Witness Indistinguishability under Multiple Oracles.

Let $w^0 = (0^n, 0^n, w)$ be the witness used by the honest prover P, and $w^1 = (\mathsf{ECC}(V^*), \pi, 0^n)$, the witness used by Sim in the simulation. Assume that there exists an adversary V^*, a distinguisher

D such that:

$$\mathsf{Adv}^{\mathsf{WI}} = \text{Prob}[\mathcal{D}(\phi\text{-}\mathsf{ZK}, z, \langle P(w^0), V^*(z)\rangle = 1] - \text{Prob}[\mathcal{D}(\phi\text{-}\mathsf{ZK}, z, \langle \mathsf{Sim}(w^1), V^*(z)\rangle = 1] \geq \epsilon$$

We turn V^* into an adversary $\mathsf{A}^{\mathsf{soa}}$ of the underlying commitment scheme that is IND-SO-COM secure.

The proof is structured in the real world, the simulated worlds, and 3 intermediate hybrids. In the real world, which is Hybrid 0, Sim plays the protocol with triple: $(0^n, 0^n, w)$. In the simulated world, which is Hybrid 4, Sim plays the protocol with triple: $(\mathsf{ECC}(V^*), \pi, 0)$. The crucial property that it allows us to move from one hybrid to another exploiting V^* distinguishing advantage, is the fact that ϕ-ZK is true in all hybrids. (Specifically, ϕ-$\mathsf{ZK}(\mathsf{ECC}(V^*), 0^n, w)$ is true; ϕ-$\mathsf{ZK}(\mathsf{ECC}(V^*), \pi, w)$ is true; ϕ-$\mathsf{ZK}(\mathsf{ECC}(V^*), \pi, 0)$ is true).

Hybrid 1. In this hybrid $\mathsf{Sim1}$ runs $\mathsf{BBCom}(\mathsf{ECC}(V^*))$ in the trapdoor generation phase. Then in the proof phase it commits to 0^n running $\mathsf{BBCom}(0^n)$ and to the witness for $(x, w) \in \mathcal{R}_L$, running $\mathsf{VSSCom}(w)$.

Assume that V^*, a theorem x and a distinguisher $\mathcal{D}$, so that $\mathcal{D}$ is able to distinguish Hybrid 1 from Hybrid 0 with non negligible probability. We construct $\mathsf{A}^{\mathsf{soa}}$ trying to break the IND-SO-COM security of the underlying commitment scheme.

Overview of the reduction. $\mathsf{A}^{\mathsf{soa}}$, on input V^*, (x, w) computes witnesses $w^0 = (0^n, 0^n, w)$ and $w^1 = (\mathsf{ECC}(V^*), 0^n, w)$. Then it emulates the prover to V^* as follows. In the trapdoor generation phase $\mathsf{A}^{\mathsf{soa}}$ does not compute the commitments for $\mathsf{BBCom}()$ by itself, but it forward messages $\mathsf{ECC}(V^*)$ and 0^n to the sampling algorithm $\mathcal{M}$ and receives the commitments from the IND-SO-COM experiment. Such commitments are parsed as an extendable merkle tree, with additionally one execution of MPC-in for each node of the tree. Note that instances of MPC-ϕ are computed directly by $\mathsf{A}^{\mathsf{soa}}$. As we explain later, because property ϕ-ZK is computed over multiple string, $\mathsf{A}^{\mathsf{soa}}$ in principle cannot compute such views without knowing the views for the tree computed by the IND-SO-COM experiment. Here the statistically hiding of the underlying commitment scheme (and thus only computational binding) comes in handy. Indeed, $\mathsf{A}^{\mathsf{soa}}$ will commit to zero strings instead of views of MPC-ϕ. Then, when receiving all the openings from IND-SO-COM, $\mathsf{A}^{\mathsf{soa}}$ will break the binding of its own commitments so that the views of MPC-ϕ's players are consistent with the views opened by IND-SO-COM experiment.

As shown already in Sec. C.2, to describe the attack of $\mathsf{A}^{\mathsf{soa}}$ to IND-SO-COM security we need to describe the sampling algorithm $\mathcal{M}$, the re-sampling algorithm $\mathcal{M}|M_I$, and the strategy of $\mathsf{A}^{\mathsf{soa}}$.

Sampling Algorithm $\mathcal{M}$. Let κ be the size of the longest string between $w_0 = \mathsf{ECC}(V^*)$ and $w_1 = 0^n$.

The sampling algorithm takes as input $z' = (w^0, w^1, \kappa, h)$ and $N = poly(\kappa)$, and it computes the tree for w_b, for a randomly chosen b, running $\mathsf{BuildRealTree}(w_b, h)$. Then it extends the real tree so that it reaches the length $|\kappa|$ (unless $|w_b| = \kappa$). For each node, the sampling algorithm prepares views of MPC-in.

Finally, $\mathcal{M}$ outputs $\mathbf{M}$, which consists of all nodes and labels of the real tree, fake nodes (in case the witness chosen was shorter that κ). Additionally it contains executions of MPC-in for each node. Finally it runs depth for the size of the real tree. The last message of $\mathbf{M}$ is the bit b chosen above. The vector $\mathbf{M}$ is sent to the IND-SO-COM experiment.

Re-sampling Algorithm $\mathcal{M}|M_I$**.** As it is clear, the commitments obtained by $\mathsf{A}^{\mathsf{soa}}$ will be parsed as commitments nodes (and instances of $\mathsf{MPC\text{-}in}$) of the tree representing w_b. In the reduction, $\mathsf{A}^{\mathsf{soa}}$ (following the malicious V^*) will ask for the opening of t views for some nodes and some $\mathsf{MPC\text{-}in}$ computed by the sampling algorithm. The views that $\mathsf{A}^{\mathsf{soa}}$ will receive are denoted by the set M_I. Then $\mathsf{A}^{\mathsf{soa}}$ expects to get from the experiment the remaining $n-t$ views (without the decommitment information).

The re-sampling algorithm computes the remaining $n-t$ views as follows. $\mathcal{M}|M_I$ picks a random bit σ, and on input the views already opened in M_I tries to compute the remaining views (i.e., $\mathbf{M}^*|M_I$) so that they are consistent with string w_σ. This is possible due to the perfect t-privacy of the MPC used to compute the extendable merkle tree and $\mathsf{MPC\text{-}in}$.

The adversary $\mathsf{A}^{\mathsf{soa}}$**.** The adversary $\mathsf{A}^{\mathsf{soa}}$ essentially tries to simulate Hybrid 0/ Hybrid 1 to V^* using the commitments respectively of 0^n or $\mathsf{ECC}(V^*)$ received from the IND-SO-COM experiment (and later the openings too).

$\mathsf{A}^{\mathsf{soa}}$, on input h received from V^* sets $z' = (w_0 = 0^n, w_1 = \mathsf{ECC}(V^*), \kappa, h)$ and activated the sampling algorithm $\mathcal{M}$. The sampling algorithm computes the vector of plaintext $\mathbf{M}$ and IND-SO-COM produces the corresponding vector of commitments $\mathbf{C}$ and sent them to $\mathsf{A}^{\mathsf{soa}}$. $\mathsf{A}^{\mathsf{soa}}$ forwards to V^* only the commitment of the root and of depth and obtain the random string r. Then $\mathsf{A}^{\mathsf{soa}}$ computes $\mathsf{BBCom}(h, 0^n)$ and views $\mathsf{VSS\text{-}WI} = P_1^{\mathsf{VSS\text{-}Wit}}, \cdots, P_n^{\mathsf{VSS\text{-}Wit}}$ by running $\mathsf{VSS\text{-}Witn}(\mathtt{Share}, w)$. In the Proof phase, V^* sends queries $\mathcal{I}^M = \{I_1^M, \ldots, I_{\ell_d}^M\}$, and $\mathcal{I}^\pi = \{I_1^\pi, \ldots, I_{\ell_d}^\pi\}$. From $\mathbf{C}$, $\mathsf{A}^{\mathsf{soa}}$ selects the committed paths corresponding to the queries in $\mathcal{I}^M$ along with the commitment of the executions of $\mathsf{MPC\text{-}in}$. Note that the real tree obtained from IND-SO-COM experiments has length $|\kappa|$. Thus, for the queries in sets $\{I_{\ell_\kappa}^M, \ldots, I_{\ell_d}^M\}$, $\mathsf{A}^{\mathsf{soa}}$ extends the paths so to reach the length dictated by the queries, by sending commitments of 0. Later $\mathsf{A}^{\mathsf{soa}}$ will open the correct views accordingly by breaking the binding of its own commitments

For the queries in $\mathcal{I}^\pi$ it computes the paths by itself. Finally, for each query $\mathsf{A}^{\mathsf{soa}}$ computes instances of $\mathsf{MPC}\text{-}\phi$ by committing to all zeros.

Then, when V^* picks the t positions $p_1, \ldots, p_t$ to be opened for each node and MPC, $\mathsf{A}^{\mathsf{soa}}$ computes the set of indexes I to play in the IND-SO-COM experiment accordingly. Namely, such that it obtains the openings for t views for each node (and $\mathsf{MPC\text{-}in}$) on the paths previously forwarded to V^*. $\mathsf{A}^{\mathsf{soa}}$ obtains from the experiment the openings for the selected views. Additionally, $\mathsf{A}^{\mathsf{soa}}$ receives the vector $\mathbf{M}$ containing the views of all the nodes in the clear, but without decommitment information. Using views in $\mathbf{M}$, $\mathsf{A}^{\mathsf{soa}}$ finally computes the correct views for $\mathsf{MPC}\text{-}\phi$ and opens the previous commitments according to such views by breaking the computational binding of the scheme.

By looking at $\mathbf{M}$ $\mathsf{A}^{\mathsf{soa}}$ obtains the witness that is claimed to be used to compute the real path, let us denote it by w_{b^*}.

When V^* terminates, $\mathsf{A}^{\mathsf{soa}}$ runs the distinguisher $\mathcal{D}$. Let b the output of $\mathcal{D}$. $\mathsf{A}^{\mathsf{soa}}$ then outputs $b^* \oplus b$.

When $\mathsf{A}^{\mathsf{soa}}$ is running in the real world, then $\mathsf{A}^{\mathsf{soa}}$ is simulating Hybrid 0 or Hybrid 1 (according to the bit chosen by $\mathcal{M}$) identically, thus the output of $\mathsf{A}^{\mathsf{soa}}$ relates to the advantage of $\mathsf{Adv}^{\mathsf{WI}} + 1/2$. When $\mathsf{A}^{\mathsf{soa}}$ is running in the ideal world, the bit b^* is randomly chosen, thus the output of $\mathsf{A}^{\mathsf{soa}}$ is also a random bit.

Hybrid 2. In this hybrid Sim2 runs $\mathsf{BBCom}(\mathsf{ECC}(V^*))$ in the trapdoor generation phase. Then in the proof phase it computes a valid PCPP proof π and commits to it running $\mathsf{BBCom}(\pi)$ and to the witness for $(x, w) \in \mathcal{R}_L$, running $\mathsf{VSSCom}(w)$. The difference between Hybrid 1 and Hybrid 2 is in the value committed in the second execution of BBCom. In Hybrid 1 Sim1 runs the protocol with strings $w^0 = (\mathsf{ECC}(V^*), 0^n, w)$ while in Hybrid 2, Sim2 uses strings $w^1 = (\mathsf{ECC}(V^*), \pi, w)$, where π is a valid PCPP proof.

Assume that V^*, a theorem x and a distinguisher $\mathcal{D}$, so that $\mathcal{D}$ is able to distinguish Hybrid 1 from Hybrid 2 with non negligible probability. We construct $\mathsf{A}^{\mathsf{soa}}$ trying to break the IND-SO-COM security of the underlying commitment scheme.

Sampling Algorithm $\mathcal{M}$**.** Let d be the size of the longest string between $w_0 = 0^n$ and $w_1 = \pi$. The sampling algorithm takes as input $z' = (w_0, w_1, d, h)$ and $N = poly(d)$, and it computes the tree for w_b, for a randomly chosen b, running $\mathsf{BuildRealTree}(w_b, h)$. Then it extends the real tree so that it reaches the length ℓ_d (unless $|w_b| = d$). For each node, the sampling algorithm prepares views of MPC-in.

Finally, $\mathcal{M}$ outputs $\mathbf{M}$, which consists of all nodes and labels of the real tree, fake nodes (in case the witness chosen was shorter that d). Additionally it contains executions of MPC-in for each node. Finally it runs depth for the size of the real tree. The last message of $\mathbf{M}$ is the bit b chosen above. The vector $\mathbf{M}$ is sent to the IND-SO-COM experiment.

Re-sampling Algorithm $\mathcal{M}|M_I$**.** The re-sampling algorithm computes the remaining $n-t$ views as follows. $\mathcal{M}|M_I$ picks a random bit σ, and on input the views already opened in M_I tries to compute the remaining views (i.e., $\mathbf{M}^*|M_I$) so that they are consistent with string w_σ. This is possible due to the perfect t-privacy of the MPC used to compute the extendable merkle tree and MPC-in.

The adversary $\mathsf{A}^{\mathsf{soa}}$**.** The adversary $\mathsf{A}^{\mathsf{soa}}$ simulates Hybrid 1/ Hybrid 2 to V^* using the commitments respectively of 0^n or π received from the IND-SO-COM experiment (and later the openings too).

In order to do that, $\mathsf{A}^{\mathsf{soa}}$, on input h first compute $\mathsf{BBCom}(h, \mathsf{ECC}(V^*))$, then it computes $\mathsf{VSS\text{-}WI} = P_1^{\mathsf{VSS\text{-}Wit}}, \cdots, P_n^{\mathsf{VSS\text{-}Wit}}$, by running $\mathsf{VSSCom}(w)$. $\mathsf{A}^{\mathsf{soa}}$ sends $z' = (w_0 = 0^n, w_1 = \pi, d, h)$ to the sampling algorithm of IND-SO-COM experiment.

Then, $\mathsf{A}^{\mathsf{soa}}$ receives the commitments $\mathbf{C}$ from the experiment, which correspond to the real tree computed for w_b. $\mathsf{A}^{\mathsf{soa}}$ forwards to V^* only the commitment of the root and of depth and obtain the random string r. In the Proof phase, V^* sends queries $\mathcal{I}^M = \{I_1^M, \ldots, I_{\ell_d}^M\}$, and $\mathcal{I}^\pi = \{I_1^\pi, \ldots, I_{\ell_d}^\pi\}$. From $\mathbf{C}$, $\mathsf{A}^{\mathsf{soa}}$ selects the committed paths corresponding to the queries in $\mathcal{I}^\pi$, along with the commitment of the executions of MPC-in. Note that the real tree obtained from IND-SO-COM experiments has length $|\kappa|$. Thus, for the queries in sets $\{I_{\ell_\kappa}^\pi, \ldots, I_{\ell_d}^\pi\}$, $\mathsf{A}^{\mathsf{soa}}$ extends the paths so to reach the length dictated by the queries, by sending commitments of 0. Later $\mathsf{A}^{\mathsf{soa}}$ will open the correct views accordingly by breaking the binding of its own commitments. The paths for the indexes in $\mathcal{I}^M$ are computed by $\mathsf{A}^{\mathsf{soa}}$ directly. $\mathsf{A}^{\mathsf{soa}}$ also computes instances of MPC-ϕ by committing to zero strings.

Then, when V^* picks the t positions $p_1, \ldots, p_t$, $\mathsf{A}^{\mathsf{soa}}$ computes the set of indexes I to play in the IND-SO-COM experiment accordingly. Namely, such that it obtains the openings for t views for

each node (and MPC-in) on the paths previously forwarded to V^*. $\mathsf{A}^{\mathsf{soa}}$ obtains from the experiment the openings for the selected views.

Additionally, $\mathsf{A}^{\mathsf{soa}}$ receives the vector $\mathbf{M}$ containing the views of all the nodes in the clear, but without decommitment information. Using the opened views $\mathsf{A}^{\mathsf{soa}}$ is able to compute the correct views for MPC-ϕ, and is able to open the previous commitment according to such new views by breaking the computational binding of the commitment scheme.

By looking at $\mathbf{M}$ $\mathsf{A}^{\mathsf{soa}}$ obtains the witness that is claimed to be used to compute the real path, let us denote it by w_{b^*}. When V^* terminates, $\mathsf{A}^{\mathsf{soa}}$ runs the distinguisher $\mathcal{D}$. Let b the output of $\mathcal{D}$. $\mathsf{A}^{\mathsf{soa}}$ then outputs $b^* \oplus b$.

When $\mathsf{A}^{\mathsf{soa}}$ is running in the real world, then $\mathsf{A}^{\mathsf{soa}}$ is simulating Hybrid 1or Hybrid 2 (according to the bit chosen by $\mathcal{M}$) identically, thus the output of $\mathsf{A}^{\mathsf{soa}}$ relates to the advantage of $\mathsf{Adv}^{\mathsf{WI}} + 1/2$. When $\mathsf{A}^{\mathsf{soa}}$ is running in the ideal world, the bit b^* is randomly chosen, thus the output of $\mathsf{A}^{\mathsf{soa}}$ is also a random bit.

Hybrid 3. In this hybrid Sim3 runs $\mathsf{BBCom}(\mathsf{ECC}(V^*))$ in the trapdoor generation phase. Then in the proof phase it computes a valid PCPP proof π and commits to it running $\mathsf{BBCom}(\pi)$. Instead of committing to the witness, Sim3 commits to 0^n running $\mathsf{VSSCom}(0^n)$.

The difference between Hybrid 2 and Hybrid 3 is in the witness committed in VSS-WI. In Hybrid 2 Sim2 commits to a valid witness w while in Hybrid 3, Sim3 commits to the string 0^n. (Namely Sim2 plays with the triple $w^0 = (\mathsf{ECC}(V^*), \pi, w)$, while Sim3 plays with triple $w^1 = (\mathsf{ECC}(V^*), \pi, 0^n)$.)

Assume that V^*, a theorem x and a distinguisher $\mathcal{D}$, so that $\mathcal{D}$ is able to distinguish Hybrid 1 from Hybrid 2 with non negligible probability. We construct $\mathsf{A}^{\mathsf{soa}}$ trying to break the IND-SO-COM security of the underlying commitment scheme.

Sampling Algorithm $\mathcal{M}$**.** Let $w_0 = w$ and $w_1 = 0^n$.

The sampling algorithm takes as input $z' = (w_0, w_1)$ and $N = poly(n)$, and it computes the tree for w_b, for a randomly chosen b, running $\mathsf{VSSCom}(w_b)$. and outputs $\mathbf{M} = P_1^{\mathsf{VSS\text{-}WI}}, \cdots, P_n^{\mathsf{VSS\text{-}WI}}, b$. The last message of $\mathbf{M}$ is the bit b chosen above. The vector $\mathbf{M}$ is sent to the IND-SO-COM experiment.

Re-sampling Algorithm $\mathcal{M}|M_I$**.** The re-sampling algorithm computes the remaining $n-t$ views as follows. $\mathcal{M}|M_I$ picks a random bit σ, and on input the views already opened in M_I tries to compute the remaining views (i.e., $\mathbf{M}^*|M_I$) so that they are consistent with string w_σ. This is possible due to the perfect t-privacy of the VSS VSS-Witn used in VSSCom.

The adversary $\mathsf{A}^{\mathsf{soa}}$**.** The adversary $\mathsf{A}^{\mathsf{soa}}$ simulates Hybrid 2/ Hybrid 3 to V^* using the commitments respectively of w and 0^n received from the IND-SO-COM experiment (and later the openings too).

In order to do that, $\mathsf{A}^{\mathsf{soa}}$, computes $\mathsf{BBCom}(\mathsf{ECC}(V^*))$ and after having received r from V^*, it computes a valid PCPC proof π and commits to it running $\mathsf{BBCom}(\pi)$. Then it obtains the vector $\mathbf{C}$ from the experiment IND-SO-COM, where $\mathbf{C}$ consists of n commitments for the views of VSS-Witn's players, and forward them to V^*.

In the Proof phase, V^* sends queries $\mathcal{I}^M = \{I_1^M, \ldots, I_{\ell_d}^M\}$, and $\mathcal{I}^\pi = \{I_1^\pi, \ldots, I_{\ell_d}^\pi\}$. $\mathsf{A}^{\mathsf{soa}}$ prepares the paths for such queries and additionally it prepares instances of PCPVer by committing to zero strings.

Then, when V^* picks the t positions $p_1, \ldots, p_t$ to be opened for each node and MPC, $\mathsf{A}^{\mathsf{soa}}$ computes the set of indexes I to play in the IND-SO-COM experiment accordingly. Thus it receives views $P_{p_1}^{\mathsf{VSS\text{-}WI}}, \cdots, P_{p_t}^{\mathsf{VSS\text{-}WI}}$. Additionally, $\mathsf{A}^{\mathsf{soa}}$ receives the vector $\mathbf{M}$ containing the remaining $(n-t)$ views of all VSS-WI but without decommitment information. $\mathsf{A}^{\mathsf{soa}}$ use such views to correctly compute views of instances MPC-ϕ. Then it opens the previous commitments of MPC-ϕ according to such views by breaking the computational binding of the underlying commitment scheme.

By looking at $\mathbf{M}$ $\mathsf{A}^{\mathsf{soa}}$ obtains the witness that is claimed to be used to compute the real path, let us denote it by w_{b^*}. When V^* terminates, $\mathsf{A}^{\mathsf{soa}}$ runs the distinguisher $\mathcal{D}$. Let b the output of $\mathcal{D}$. $\mathsf{A}^{\mathsf{soa}}$ then outputs $b^* \oplus b$.

When $\mathsf{A}^{\mathsf{soa}}$ is running in the real world, then $\mathsf{A}^{\mathsf{soa}}$ is simulating Hybrid 2 or Hybrid 3 (according to the bit chosen by $\mathcal{M}$) identically, thus the output of $\mathsf{A}^{\mathsf{soa}}$ relates to the advantage of $\mathsf{Adv}^{\mathsf{WI}} + 1/2$. When $\mathsf{A}^{\mathsf{soa}}$ is running in the ideal world, the bit b^* is randomly chosen, thus the output of $\mathsf{A}^{\mathsf{soa}}$ is also a random bit.

Hybrid 3 corresponds to the simulated transcript.

The following Lemma is therefore proved.

Lemma 4. *If* SHCom *is a IND-SO-COM-secure commitment scheme, if* VSS-Witn, VSS, depth *are perfect t-private VSS and* MPC-in, MPC-ϕ *are perfect t-private MPC protocol, then Protocol 2 is statistical Zero Knowledge.*

F Generalized Size-Hiding Black-Box Proof of Set Membership

In this section we describe in details our construction of Generalized BB Proofs of Set Membership.

In this section we use the word *index* in place of element. Namely, we say that the verifier sends index i, to mean that the verifier wants to check if the element i is not empty in the database. We use this nomenclature just because it seems more intuitive when used in conduction with accessing nodes of a Merkle Tree.

F.1 Overview of the construction

In *size-hiding black-box proofs of membership*, a prover P commits to a *sparse* set D (sparse means that there exist indexes i for which $D[i] = \perp$). In this setting we do not want to consider any polynomial upper bound on the size of the set. As such, a verifier can query an index i in the set $[2^{poly(n)}]$. At the same time a prover has a finite set and supports only indexes belonging to a certain range $[2^{\mathsf{maxIndSize}}]$. In addition to hiding the size of D, namely the number of elements that D actually contains, here we want to hide even the range of the indexes that P is able to answer.

In the proof phase, the verifier challenges P with an index $i \in [2^{poly(n)}]$. P then proves whether the position i in D is not empty. From such proof V learns the following. If i hits a YES instance, then V learns that P supports indexes of length $\geq |i|$. If i hits a NO instance, then V learns only that $D[i]$ is empty, without learning anything about the size of the set D and the range of indexes that P supports.

We next show how Idea 1 and Idea 2 behind the extendable Merkle Tree are used to implement size-hiding BB proofs of membership. Let us setup some notation. A prover P supports range k, means that its set contains element in the range $[2^k]$. In this setting we assume that an index is always in its shortest bit representation (e.g., an index 0110 it is always cut as 110).

In the commit-and-prove setting, P has a full string and when challenged with a set of indexes, it proves that for those indexes the property is satisfied. In the proof of membership setting, P has a sparse set, and when challenged with an index, it proves whether the set contains such index. Thus, the first difference between the two setting is that in the latter, because the set is sparse and potentially huge, P cannot afford to compute the complete tree as before, but it should compute only the paths for non-empty leaves. The second difference is that, when proving membership P should be able to provide proofs for NO-instances.

The most important challenge in this setting though, is that the *range* of the indexes supported by the prover must remain hidden. This becomes problematic when proving YES-instances for short indexes, due to the way merkle trees are built. The reason is the following. A merkle tree is constructed by arranging all the elements on the leaves, and building up paths till the root. Thus, only leaves carry information about the set. Any internal node instead carries only *control* information. Namely they store values that enable checking of consistency (i.e., the hash values). It follows that, to prove any property wrt an element of the set, P must show a path from the root to the leaf. Now, if P supports all queries in the set 2^k, in order to represent all the elements, it must arrange them on a tree of depth k. In particular, if V asks a proof for an element in position i with $|i| << k$, P can only open a path which is at least of depth k. The problem of this approach is that, because all the elements of the set are arranged at level k, the length of the path for a YES-instance depends on the range k.

We need to construct a merkle tree such that, for any index i, the length of the path opened to prove membership of such index depends only on i (and is independent of k).

We do this by building the merkle tree in the following way. P places the actual elements of the set in every level of the tree and not only on the leaves. Namely, level i of the tree contains the non-empty elements for indexes in 2^i. Consequently here also internal nodes carry actual value information (along with the control information necessary for the consistency of the tree). Note that, along a path from a YES instance to the root, there can be nodes which are not in the set (they are NO-instances). As we shall see later, such nodes will carry only control information necessary to prove consistency. This new technique allows us to open paths for YES-instances for indexes which are shorter then the range supported by P, without revealing anything about the range. For indexes which are longer than the range support, we allow P to provided extended paths, using same techniques as before.

In the following we outline the main differences with the previous construction of the extendable merkle tree. Detailed explanations are provided in App. F. Let D be the sparse set that P is committing to, and let T be the length of maximum index in D.

Building the tree.

- Node representation. Any node i carries two types of information: *control* information, is the label l as before; *value*, is the value of the element in position i, and belongs to the set $\{D[i], \perp, \texttt{dead}\}$, where $\perp$ denotes the fact $D[i] = \perp$ but node i is an active node on the path (it has children); `dead` means that $D[i] = \perp$ and node i is a dead node on the path.
  A node in position i here is again a pair $[l^i, \mathsf{VSS}^i]$, the only difference is that now the VSS part is secret sharing a pair $\mathsf{Val}^i = [l^i, v^i]$, where $v \in \{D[i], \perp, \texttt{dead}\}$. The membership proof is tested only over $\mathsf{Val}[2]$.
- Tree Construction. To build a tree, P arranges the elements of D according to their index. Namely, at any level j it arranges elements in D with index $|\gamma| = j$ and computes the nodes accordingly (i.e., it compute $\mathsf{Val}^\gamma = [l^\gamma, D[\gamma]]$ for yes instances, and $\mathsf{Val}^\gamma = [l^\gamma, \texttt{dead}]$ for

siblings which are NO-instances). Given the nodes at level j, P builds the level $j-1$ by first creating nodes for the elements $D[\gamma']$ with $\gamma' \in \{0,1\}^{j-1}$, and then creating the remaining nodes at level $j-1$ which are on the paths from nodes of the level below, by computing $\mathsf{Val}^{\gamma'} = [l^{\gamma'}, \perp]$.

Verifier's Challenge. V sends an index $i \in [2^n]$. Because there is no upper bound over the size of the set, V can ask for any index in any exponentially large set.

Functionalities. $\mathcal{F}_{\mathsf{innode}}$. $\mathcal{F}_{\mathsf{innode}}$ is computed only for YES-instance. Thus, for a node γ, after obtained value Val from the reconstruction of VSS^{γ}, $\mathcal{F}_{\mathsf{innode}}$ first check $\mathsf{Val}[1]$ to check that the innode property holds. Then it checks that $\mathsf{Val}[2] \not= \mathtt{dead}$. Indeed, a $\mathtt{dead}$ node cannot be *on* the path for a YES-instance. $\mathcal{F}_{\mathsf{YES}}$. For a node γ, $[l^{\gamma}, \mathsf{VSS}^{\gamma}]$, functionality $\mathcal{F}_{\mathsf{YES}}$ runs the reconstruction of VSS^{γ}, obtains the value Val and outputs 1 iff $\mathsf{Val}[2] \notin \{\perp, \mathtt{dead}\}$.

Proof of YES-instance. If $D[i] \neq \perp$, then it does exist a real path from the root to $D[i]$. Therefore, P roughly follows the same procedure used in $\mathsf{BBProve}$ and opens a path of length i. It runs $\mathsf{MPC\text{-}Yes}$ only for the node in position i.

Proof of NO-instance. If $D[i] = \perp$ OR $i > T$ then P must prove that the i-th value is not present in the set. There are two cases:

1. node i exists already in the real tree because is on the path of an existing node (e.g., if node 110 is in the database, then also node 11 must exist even if $D[3] = \perp$). In this case, P has to prove that there exists a real consistent path from the root of node i, *and* that $\mathsf{Val}^{i}[2] = \perp$.
2. node i is not in the real tree. In this case i, P builds a fake path from i up to the first "dead" node at position j (with $|j| < |i|$) in the real tree which is on the path from i to the root. The consistency requirements is that 1) the path from root to node $j-1$ is consistent; 2) node j is dead (i.e., $\mathsf{Val}^{j}[2] = \mathtt{dead}$). Thus for NO-instances P runs an MPC-in-the-head for a functionality $\mathcal{F}_{\mathsf{NO}}$ (Fig. 12), which takes as input a *path*, and outputs 1 iff both conditions 1) and 2) hold.

Underlying Commitment Scheme. As we are constructing ZK proof of membership, here we need a commitment scheme which is SOA-secure under the *simulation* based definition. We use the simulation based BB SOA-secure scheme presented in [ORSV13].

Multiple Queries. To handle multiple queries both P and V must be stateful. They have to remember the paths opened so far. If this was not the case, then for each query P will open different views for the same path, and privacy is violated.

MPC-in-the-head in use. Here we use $4t$-private, t-correct MPC-protocols. The reason is that here V is allowed to make multiple queries. Therefore, a node might appear along the path for several queries. In such a case, new MPC-in-the-head are run for the same node, and a new subset of t views must be open to convince the verifier (note that for soundness, it cannot be the case that P opens the same views for every request). The observation here is that, because we are in a binary tree the same node can be used to prove at most 2 paths. More explanations are provided in App. F.2.

F.2 The protocol

In the following we use the shorter notation $\mathtt{Share}(value)$ to denote $\mathsf{VSS}(\mathtt{Share}, value)$.

Procedure $\mathsf{BuildSparseTree}$**.** On input a sparse set D of maximum range $\mathsf{maxIndSize}$, and a hash function h.

Leaves. For every index i such that $|i| = \mathsf{maxIndSize}$ and $D[i] \neq \perp$; compute VSS part of node i

as follows. $[P_1^{\mathsf{VSS}^i}, \ldots, P_n^{\mathsf{VSS}^i}] \leftarrow \mathtt{Share}(0^n|D[i])$. Let $\mathsf{VSS}^i = [P_1^{\mathsf{VSS}^i}, \ldots, P_n^{\mathsf{VSS}^i}]$.

Level j. For $j = \mathsf{maxIndSize} - 1$ to 1:

- Compute nodes for element in D. For every γ such that $|\gamma| = j$ and $D[\gamma] \neq \perp$, compute:
  - Label part.
    - Case 1. If node $\gamma 0$ and $\gamma 1$ both exist. $l^\gamma = [h(P_1^{\mathsf{VSS}^{\gamma 0}}| \ldots |h(P_n^{\mathsf{VSS}^{\gamma 1}})]$;
    - Case 2. If node γb exists and $\gamma \bar{b}$ does not exists. Compute $\mathsf{VSS}^{\gamma\bar{b}} = [P_1^{\mathsf{VSS}^{\gamma\bar{b}}}, \ldots, P_n^{\mathsf{VSS}^{\gamma\bar{b}}}] \leftarrow \mathtt{Share}(\mathtt{dead}|0^n)$. $l^\gamma = [h(P_1^{\mathsf{VSS}^{\gamma 0}}| \ldots |h(P_n^{\mathsf{VSS}^{\gamma 1}})]$;
    - Case 3. If node $\gamma 0$ and $\gamma 1$ do not exist. $l^\gamma = 0^n$.
  - VSS part. Compute $\mathsf{VSS}^\gamma = [P_1^{\mathsf{VSS}^{\gamma\bar{b}}}, \ldots, P_n^{\mathsf{VSS}^{\gamma\bar{b}}}] \leftarrow \mathtt{Share}(l^\gamma|D[\gamma]|)$.
- Compute empty nodes which are ancestors of non-empty nodes. Let γ be a node at level j which is an ancestor of a non-empty node. This means that node γ has both children $\gamma 0$ and $\gamma 1$. Compute Label as Case 1. Compute VSS part as follows. $\mathsf{VSS}^\gamma = [P_1^{\mathsf{VSS}^{\gamma\bar{b}}}, \ldots, P_n^{\mathsf{VSS}^{\gamma\bar{b}}}] \leftarrow \mathtt{Share}(l^\gamma|\perp)$. *Note, an empty node on a real path is stored with symbol $\perp$ and not dead. This means that even if such node is empty, as it is on the path of same YES-instance, it it still must be consistent.*

Protocol 6 (Generalize BB Proofs of Set Membership)**.** *P has in input a possible sparse database D which contains range* maxIndSize*.*

Commitment of the Database. *V sends a randomly chosen $h \in \mathcal{H}$ to P. P runs* $\mathsf{db} \leftarrow \mathsf{SOAcom}(\mathsf{BuildSparseTree}(h, D, \mathsf{maxIndSize}))$*. Send* db *to V.*

Proof First Query. *V sends query q_1 to P.*

- **Commitment of proof starting from** nd **= root.**
  1. *Case 1. YES-instance: $D[q_1] = v$.*
     (a) *Retrieve the path from the node* nd *to node q_1.*
     (b) *For each node* on *the path from q_1 up to* nd*, run* MPC-DBInnode *to prove consistency of the node.*
     (c) *For node q_1 run* MPC-Yes *on public input v.*
     (d) *Send to V, the value v, commitments of each node of the path, one execution of* MPC-DBInnode *for each node* on *the path (starting from node* nd*), and the execution* MPC-Yes *for the node in position q_1.*
  2. *Case 2. NO-instance which is part of the real tree: $D[q_1] = \perp$.*
     (a) *Retrieve the path from the node* nd *s to node q_1.*
     (b) *Run protocol* MPC-No *over the nodes belonging to the path starting from node* nd*.*
     (c) *Send to V, the value $\perp$, commitments of each node of the path and of* MPC-No*.*
  3. *Case 3. NO-instance which does not exist: $D[q_1] = \perp$. In this case P computes a fake path till position q_1 and then prove that q_1 is a NO-instance as is case 2. The path is computed as follows. Let j be the first node along the path between the node* nd *and node q_1 which belongs to the real tree (i.e., which actually exists). Computes the node as follows.*

(VSS part) For each node γ along the path from j to q_1 compute the VSS part, denoted by $\mathsf{VSS}^\gamma = P_1^{\mathsf{VSS}^\gamma}, \ldots, P_n^{\mathsf{VSS}^\gamma} \leftarrow \mathtt{Share}(0^n, \mathtt{dead})$. Namely, construct a dead node.

(Label) For each node γ along the path from j to q_1 compute label as $l^\gamma = [h(P_1^{\mathsf{VSS}^{\gamma 0}}) | \ldots | h(P_n^{\mathsf{VSS}^{\gamma 1}})]$.

- **Challenge.** *V randomly chosen players $p_1, \ldots, p_t$ and send them to P. P opens the requested views for each node of the path, and each MPC-in-the-head committed previously.*
- **Verification.** *V performs the following checks.*

YES-Instance

1. *Check Hash Consistency. For each node i, check that the values opened for the label is the hash of the correspondent views. Namely, check that $l^i[p_j] = h(P_{p_j}^{\mathsf{VSS}^{i0}})$ and $l^i[p_j + n] = h(P_{p_j}^{\mathsf{VSS}^{i1}})$ (for $j = 1, \ldots, t$).*
2. *Check Consistency of Nodes (innode property). This property is checked for all nodes on the path (except for the leaf). For each player $P_{p_j}^{\mathsf{Dbinnode}^i}$ of $\mathsf{MPC\text{-}DBInnode}$ checks that: 1) input consistency: check that player $P_{p_j}^{\mathsf{Dbinnode}^i}$ plays with input the view of the correspondent player in VSS^i and the p_j-th hash of the label associated to such node. Namely, the input consists of the values $l^i[p_j]$, $l^i[p_j + n]$, $P_{p_j}^{\mathsf{VSS}^i}$. 2) View Consistency. Check that the views of all t players are consistent. 3) Output correctness. Check that the output of all the revealed views is 1.*
3. *Check Membership: i.e. $D[q_1] = v$. This property is checked only for node q_1. Check view of players $P_{p_1}^{\mathsf{YES}}, \ldots, P_{p_t}^{\mathsf{YES}}$ as follows. Input: check that $P_{p_j}^{\mathsf{YES}}$ has in input the view $P_{p_j}^{\mathsf{VSS}^{q_1}}$ and the value v, for $j = 1, \ldots, t$. View Consistency. Check that the views of all t players are consistent. Output: check that all t players output 1.*
   If any of this check fails, abort.

NO-Instance

1. *Check Hash Consistency. For each node i, check that the values opened for the label is the hash of the correspondent views. Namely, check that $l^i[p_j] = h(P_{p_j}^{\mathsf{VSS}^{i0}})$ and $l^i[p_j + n] = h(P_{p_j}^{\mathsf{VSS}^{i1}})$ (for $j = 1, \ldots, t$).*
2. *Check NON-membership. For each player $P_{p_j}^{\mathsf{MPC\text{-}No}^i}$ of $\mathsf{MPC\text{-}No}$ checks that: 1) input consistency: check that player $P_{p_j}^{\mathsf{MPC\text{-}No}^i}$ plays with input the view of the correspondent player in VSS for every node on the path, and the corresponding label value. Namely, the input consists of the values: $l^i[p_j]$, $l^i[p_j + n]$, $P_{p_j}^{\mathsf{VSS}^i}$ for $i = 1$ to $i = q_1$. 2) Views consistency. Check that the opened views are consistent with the honest execution of the protocol and with each other. 3) Output correctness. Check that the output of all the revealed views is 1. If any of this check fails, abort.*

Proof i-th Query. *V sends query q_i to P.*

- *Case YES-instance. If $D[q_i] = v$. Consider the longest path from the root to node q_i that has been opened for a YES-instance. Let $\mathsf{q_{prev}}$ the last node over such path. Follow the same procedure shown for the first query case, but set $\mathsf{nd} = \mathsf{q_{prev}}$.*
- *Case NO-instance. If $D[q_i] = \perp$. Consider the longest path from the root to node q_i that has been opened for a YES-instance* **OR** *NO-instance. Let $\mathsf{q_{prev}}$ the last node over such path. Run same procedure as for first query setting $\mathsf{nd} = \mathsf{q_{prev}}$.*

Functionality $\mathcal{F}_{\mathsf{YES}}$.
This functionality is computed for a node q.
Public Input: Index of the leaf q, claimed value $D[q]$

- Input. From player P_j obtains input $P_j^{\mathsf{VSS}^q}$.
- Computation. Run $\mathsf{Val}^q = [l^q, v'] \leftarrow \mathtt{Recon}(P_1^{\mathsf{VSS}^q}, |\cdots|, P_n^{\mathsf{VSS}^q})$. If $v' = v$ output 1 to all players. Else output 0 to all players.

Figure 10: The $\mathcal{F}_{\mathsf{YES}}$ functionality.

Functionality $\mathcal{F}_{\mathsf{innode}}^{\mathsf{db}}$.
This functionality is computed for a node γ.
Public Input: Index of the node γ.

- Input. From player P_j obtains input $P_j^{\mathsf{VSS}^\gamma}$, $l^\gamma[j]$, $l^\gamma[j+n]$.
- Computation. Check the innode property.
  - run $\mathsf{Val}^\gamma = [l^\gamma, v'] \leftarrow \mathtt{Recon}(P_1^{\mathsf{VSS}^\gamma}, |\cdots|, P_n^{\mathsf{VSS}^\gamma})$.
  - If $l^\gamma = l^\gamma[1]|\ldots|l^\gamma[2n]$ AND if v' $\not$dead output 1 to all players.
  - Else, output 0 to all players.

Figure 11: The $\mathcal{F}_{\mathsf{innode}}^{\mathsf{db}}$ functionality.

Observations.

1. YES-instance paths. Any node on a YES-instance path is consistent, i.e., the innode property holds.
2. NO-instance paths. A node on a NO-instance path is not necessarily consistent.
3. We do not need to commit to the size of the tree in advance. The reason is that for YES-instances we always open real paths. For NO-instances the flag that allows to cheat is the first dead node along the path. Thus, it is sufficient to have one dead node to extend any path to any arbitrary length.

The need of MPC with $4t$-privacy. When multiple queries are allowed, a node (which is essentially a VSS execution) can be along the path of multiple queries, thus involved in multiple proofs, and for each independent proof V will ask to open t freshly selected players. In our construction, a node at position i can be involved in at most: 1) one NO-proof, 2) one YES-proof, 3) one hash consistency because is on the path of some query j with $j > i$; and 4) one (YES/NO) proof when node i is the node which is actually queried. Therefore, for a node P will have to show at most $4t$ values.

Functionality $\mathcal{F}_{\mathsf{NO}}$.
This functionality is computed nodes on the path from node nd to node q.
Public Input: Index of the starting node nd, index of the leaf q.

- Input. From player P_j obtains input $P_j^{\mathsf{VSS}^\gamma}$, $l^{\gamma'}[j]$, $l^{\gamma'}[j+n]$, for $\gamma = \mathsf{nd}$ to $\gamma = q$, for $\gamma' = \mathsf{nd}$ to $\gamma' = q/2$.
- Computation. Check the innode property. For $\gamma = \mathsf{nd}$ to $\gamma = q/2$.
  - run $\mathsf{Val}^\gamma = [l^\gamma, v] \leftarrow \mathtt{Recon}(P_1^{\mathsf{VSS}^\gamma}, |\cdots|, P_n^{\mathsf{VSS}^\gamma})$.
  - If $l^\gamma = l^\gamma[1]|\ldots|l^\gamma[2n]$, continue.
  - Else, if $v = \mathtt{dead}$, $\mathtt{break}$.
  - Else, output 0 to all players. (In this case, γ is a node that it is non consistent and not dead. Note that a node in which $v = \perp$, still must be consistent).

  Run $\mathsf{Val}^\gamma \leftarrow \mathtt{Recon}(P_1^{\mathsf{VSS}^q}, |\cdots|, P_n^{\mathsf{VSS}^q})$. If $\mathsf{Val}^\gamma[2] \in \perp, \mathtt{dead}$ output 1 to all players. Else, output 0 to all players.

Figure 12: The $\mathcal{F}_{\mathsf{NO}}$ functionality.

F.3 Soundness

Soundness follows from the collision resistance property of the hash function, the binding of the underlying commitment scheme and the t-correctness of the VSS and MPC protocol being used.

Assume that there exists a PPT malicious prover P^* which outputs a commitment cmDb, and a query q for which P^* opens two accepting proofs (π_q^1, v_q^1) and (π_q^2, v_q^2) and $v_1 \neq v_2$. The proof of divided in two cases.

Case 1. π_q^1 and π_q^2 are both YES-proofs. We consider the following hybrids.
Hybrid 1. In Hybrid 1 the prover sends everything in the clear. In this hybrid to prove that q is a YES-instance and that $D[q] = v$, P simply sends the path path_q from the root till node q. Namely, a YES-proof π_q in this hybrid consists of each node i, $[l^i, \mathsf{VSS}^i]$ (recall that $\mathsf{VSS}^i = P_1^{\mathsf{VSS}^q}, |\cdots|, P_n^{\mathsf{VSS}^q}$) which is *on* the path from the root to node q. A YES proof is **accepting** if, for each node i, it holds that (hash consistency) l^i corresponds to the hash of $\mathsf{VSS}^{i0}, \mathsf{VSS}^{i1}$ and that ($\mathsf{Dbinnode}$ property) l^i is the value obtained when running $\mathtt{Recon}(P_1^{\mathsf{VSS}^i}, |\cdots|, P_n^{\mathsf{VSS}^i})$ and that it is not a $\mathtt{dead}$ node. Because everything is in the clear, V checks both property on its own.

Now assume that P is able to compute $\pi_q^1 = \mathsf{path}_q^1$ and $\pi_q^2 \mathsf{path}_q^2 =$ such that

$$(l_1, v_q^1) \leftarrow \mathtt{Recon}([P_1^{\mathsf{VSS}^q}, |\cdots|, P_n^{\mathsf{VSS}^q}]_{\mathsf{path}_q^1})$$

and

$$(l_2, v_q^2) \leftarrow \mathtt{Recon}([P_1^{\mathsf{VSS}^q}, |\cdots|, P_n^{\mathsf{VSS}^q}]_{\mathsf{path}_q^2})$$

but $v_q^1 \neq v_q^2$.

It follows that $[\mathsf{VSS}^q]_{\mathsf{path}_q^1} \neq [\mathsf{VSS}^q]_{\mathsf{path}_q^2}$ in at least $(n-4t)$ positions (as it was already shown in Lemma 1). Because path_q^1 and path_q^2 share the same root, and because $\mathsf{Dbinnode}$ property holds, it

must be the case the there exists a node i, such that $[l^i]_{\mathsf{path}_q^1}=[l^i]_{\mathsf{path}_q^2}$ but $[\mathsf{VSS}^{ib}]_{\mathsf{path}_q^1} \neq [\mathsf{VSS}^{ib}]_{\mathsf{path}_q^2}$, for a bit $b \in \{0,1\}$. Namely, for a node i on the path, the label l^i is the same in both path_q^1 and path_q^2, but at least one of the children between $i0$ and $i1$, values $[\mathsf{VSS}^{ib}]_{\mathsf{path}_q^1}$ and $[\mathsf{VSS}^{ib}]_{\mathsf{path}_q^1}$ differ in at least $(n-4t)$ views in the two paths. Thus, we found a collision, hence contradicting the collision resistance of family $\mathcal{H}$.

Hybrid 2. In this hybrid the prover sends everything in the clear, but the verifier V refrains to look to all views of the nodes and checks only t views. Hence, P additionally sends execution of MPC-DBInnode for each node of the path, and MPC-Yes for node q.

In this hybrid a YES-proof consists of: 1) t views for every node on the path from root to q; 2) t views for each execution of Dbinnode (one for each node); 3) t views for the execution of MPC-Yes for node q. A YES-proof is accepting if: for each node i (hash consistency), $l^i[p_j] = h(P_{p_j}^{\mathsf{VSS}^{i0}})$ and $l^i[p_j+n] = h(P_{p_j}^{\mathsf{VSS}^{i1}})$; (Dbinnode property): t views of MPC-DBInnode's players pass the verification step of the protocol; (membership): t views of MPC-Yes's players pass the verification step of the protocol. Note that in this hybrid P^* does not know which views are tested.

The difference between Hybrid 1 and Hybrid 2 is in fact that the consistency of the node and the validity of the proof is proved using MPC-in-the-head instead of being verified directly by V. This, in this hybrid we use the t correctness of the MPC protocols being used to argue that P^* has no better advantage of cheating in hybrid 2 wrt hybrid 1.

A bit more in details, assume P^* provides accepting (π_q^1, v_q^1) and (π_q^2, v_q^2).

Due to the t-correctness of protocol MPC-Yes (recall that MPC-Yes implements $\mathcal{F}_{\mathsf{YES}}$, and the task of $\mathcal{F}_{\mathsf{YES}}$ on input v, is to check that the VSS part of node q reconstructs to value v), whp it holds that:

$$(l_1, v_q^1) \leftarrow \mathtt{Recon}([P_1^{\mathsf{VSS}^q}, |\cdots|, P_n^{\mathsf{VSS}^q}]_{\mathsf{path}_q^1})$$

$$(l_2, v_q^2) \leftarrow \mathtt{Recon}([P_1^{\mathsf{VSS}^q}, |\cdots|, P_n^{\mathsf{VSS}^q}]_{\mathsf{path}_q^2})$$

Hence, given that $v_q^1 \neq v_q^2$, due to the hash consistency check it follows that parent label $l^{q'}$ of node q from path_q^1 must differ from the one in path_q^2 in at least $(n-4t)$, if not we found already a collision.

Due to the t-correctness of protocol MPC-DBInnode, which guarantees that each node is consistent whp, the above argument can be extended for each level of the path. Therefore, a P^* successful in hybrid 2, can be reduced to a P^* successful in hybrid 1.

Hybrid 3. This hybrid is the same as before, except that now the prover sends the commitment to the YES-proof and later it opens only the t views requested by V. It follows from the binding of the underlying commitment scheme that a successful P^* in hybrid 3 can be transformed in a successful adversary in hybrid 2.

Case 2. π_q^1 is a NO-proof and π_q^2 is a YES-proof. For better clarity, let us define $\pi_q^{\mathsf{yes}} \stackrel{def}{=} \pi_q^1$ and $\pi_q^{\mathsf{no}} \stackrel{def}{=} \pi_q^2$. We consider the following hybrids.

Hybrid 1. In this hybrid the prover sends everything in the clear.

In this hybrid, a NO-proof π_q^{no} corresponds to a path from the root to node q. A NO-proof is accepting if the following holds: (hash consistency) for each node on the path l^i corresponds to the

hash of VSS part $\mathsf{VSS}^{i0}, \mathsf{VSS}^{i1}$. (Consistency of the path *up to* the first `dead` node) Let j be the first `dead` node along the path from root to q. Check innode property for all nodes above j. (NO membership). For node q check that $(l_q^{\mathsf{no}}, v_{\mathsf{no}}) \leftarrow \mathtt{Recon}(\mathsf{VSS}^q)$ and $v_{\mathsf{no}} \in \{\perp, \mathtt{dead}\}$.

A YES-proof corresponds to the one described in Hybrid 1 for Case 1.

Now assume that P^* is able to compute a NO-proof $\mathsf{path}_q^{\mathsf{no}}$ and a YES-proof $\mathsf{path}_q^{\mathsf{yes}}$ and both are accepting. Because $\mathsf{path}_q^{\mathsf{no}}$ is a NO-proof, than it holds that: $v_{\mathsf{no}} \leftarrow \mathtt{Recon}([P_1^{\mathsf{VSS}^q}, |\cdots|, P_n^{\mathsf{VSS}^q}]_{\mathsf{path}_q^{\mathsf{no}}})$ and $v_{\mathsf{no}} \in \{\mathtt{dead}, \perp\}$ Because $\mathsf{path}_q^{\mathsf{yes}}$ is a YES-proof, than it holds that: $v_{\mathsf{yes}} \leftarrow \mathtt{Recon}([P_1^{\mathsf{VSS}^q}, |\cdots|, P_n^{\mathsf{VSS}^q}]_{\mathsf{path}_q^{\mathsf{yes}}})$ $v_{\mathsf{yes}} \notin \{\mathtt{dead}, \perp\}$. Thus $v_{\mathsf{no}} \neq v_{\mathsf{yes}}$.

Now first note that, for a NO-proof the verifier checks that innode property is verified for only for the nodes on the path from the root to the first `dead` node. Any node which has a dead node as ancestor does not have to be consistent. In a YES-proof instead V checks that *every* node on the path satisfies the innode property *and* it is **not** dead.

Therefore it could have been the case that P^* computed a consistent path $\mathsf{path}_q^{\mathsf{yes}}$ opening to v_{yes} for the YES-proof, and a path $\mathsf{path}_q^{\mathsf{no}}$ opening to a dead node, exploiting the fact that on a path leading to a dead node, some nodes can be inconsistent.

The observation is that, even for a NO-proof, an inconsistent node is accepted only after a dead node was found on the path. Now, because both $\mathsf{path}_q^{\mathsf{no}}$ and $\mathsf{path}_q^{\mathsf{yes}}$ start from the same root and end to the node q, and because on a YES-path every node must be not dead, there must have been a node i along the path such that $(l_i^{\mathsf{no}}, \mathtt{dead}) \leftarrow \mathtt{Recon}([P_1^{\mathsf{VSS}^i}, |\cdots|, P_n^{\mathsf{VSS}^i}]_{\mathsf{path}_q^{\mathsf{no}}})$ while $(l_i^{\mathsf{yes}}, v_i) \leftarrow \mathtt{Recon}([P_1^{\mathsf{VSS}^i}, |\cdots|, P_n^{\mathsf{VSS}^i}]_{\mathsf{path}_q^{\mathsf{yes}}})$. Consequently, $[\mathsf{VSS}^i]_{\mathsf{path}_q^{\mathsf{no}}} \neq [\mathsf{VSS}^i]_{\mathsf{path}_q^{\mathsf{yes}}}$ in at least $(n - 4t)$ views, but they have the same parent label in path $\mathsf{path}_q^{\mathsf{no}}$ and $\mathsf{path}_q^{\mathsf{yes}}$.

Thus we extract a collision.

Hybrid 2. In this hybrid the verifier only checks t views, thus the prover computes the additional MPC-in-the-head: $\mathsf{Dbinnode}, \mathsf{MPC\text{-}Yes}$ for the YES-proof, and $\mathsf{MPC\text{-}No}$ for the NO-proof.

Because the MPC-in-the-head performs the very same tests that V performed by itself in Hybrid 1, and due to the t-correctness of the MPC being used, any P^* successful in this hybrid can be transformed in a P^* successful in hybrid 1.

Hybrid 3. In this hybrid the prover first sends a commitment of the YES/NO proof and then opens t views as requested by V. Due to the binding of the underlying commitment scheme, any P^* successful in this hybrid can be transformed in a P^* successful in hybrid 2.

F.4 Zero Knowledge

Zero knowledge follows from the simulation based SOA-security of the underlying commitment scheme and from the $4t$ privacy of the underlying MPC protocols being used.

The Simulator Sim here will commit to the 0 string in the commitment phase. Then, when receiving a query q, it obtains the value of $D[q]$ from the experiment. It computes the commitments required for the type of proof (i.e., YES-proof or NO-proof). For example, if q is a YES-instance, Sim will compute the commitments for the residual path (i.e., the path contains only nodes that haven't been involved in previous YES-proofs so far), the $\mathsf{MPC\text{-}DBInnode}$ for each node, and finally one execution of $\mathsf{MPC\text{-}Yes}$. Note that, even if $D[q]$ is revealed, still Sim has no information about the other values, so it cannot actually commit to a valid path or valid MPC executions. Thus, Sim will compute the commitments for the proof, by running the SOA-simulator granted by the SOA-security of the commitment scheme.

Then, when V^* sends the positions $p_1, \ldots, p_t$ to be revealed, Sim proceeds as follows. First, it computes the nodes on the residual path so that the hash consistency is preserved. Each of such node is just computed as the VSS of the zero string. Obviously for such nodes the Dbinnode property (or any other property) is not satisfied.

This is not a problem because Sim will compute views of player MPC-DBInnode and MPC-Yes/MPC-No, by exploiting the knowledge of the t selected players, and running the simulator SimMpc granted from the perfect $4t$-privacy of the MPC protocols being used. Namely, Sim runs SimMpc providing values $P_{p_j}^{\mathsf{vss}^i}$ (for each $j \in [t]$) as the inputs for the t malicious parties, and for output 1. It then uses the output of SimMpc to set the views of each MPC-DBInnode and MPC-Yes/MPC-No.

With such views so computed, Sim uses the SOA-simulator to equivocate the commitments sent before.