Resettably Sound Zero-Knoweldge Arguments from OWFs - the (semi) Black-Box way

Rafail Ostrovsky
UCLA, USA

Alessandra Scafuro
UCLA, USA

Muthuramakrishnan Venkitasubramaniam
University of Rochester, USA

Abstract

We construct a constant-round resettably-sound zero-knowledge argument of knowledge based on black-box use of any one-way function.

Resettable-soundness was introduced by Barak, Goldreich, Goldwasser and Lindell [FOCS 01] and is a strengthening of the soundness requirement in interactive proofs demanding that soundness should hold even if the malicious prover is allowed to "reset" and "restart" the verifier. In their work they show that resettably-sound ZK arguments require non-black-box simulation techniques, and also provide the first construction based on the breakthrough simulation technique of Barak [FOCS 01]. All known implementations of Barak's non-black-box technique required non-black-box use of a collision-resistance hash-function (CRHF).

Very recently, Goyal, Ostrovsky, Scafuro and Visconti [STOC 14] showed an implementation of Barak's technique that needs only black-box access to a collision-resistant hash-function while still having a non-black-box simulator. (Such a construction is referred to as *semi black-box*.) Plugging this implementation in the BGGL's construction yields the first resettably-sound ZK arguments based on black-box use of CRHFs.

However, from the work of Chung, Pass and Seth [STOC 13] and Bitansky and Paneth [STOC 13], we know that resettably-sound ZK arguments can be constructed from non-black-box use of any one-way function (OWF), which is the *minimal* assumption for ZK arguments.

Hence, a natural question is whether it is possible to construct resettably-sound zero-knowledge arguments from black-box use of any OWF only. In this work we provide a positive answer to this question thus closing the gap between black-box and non-black-box constructions for resettably-sound ZK arguments.

1 Introduction

Zero-knowledge (ZK) proofs [GMR85] allow a prover to convince a verifier of the validity of a mathematical statement of the form "$x \in L$" without revealing any additional knowledge to the verifier besides the fact that the theorem is true. This requirement is formalized using a simulation paradigm: for every malicious verifier there exists a simulator that having a "special" access to the verifier (special in the sense that the access granted to the simulator is not granted to a real prover) is able to reproduce the view that the verifier would obtain interacting with an honest prover. The simulator has two special accesses to the verifier: black-box access, it has the power of resetting the verifier during the simulation; non-black-box access, it obtains the actual code of the verifier. While providing no additional knowledge, the proof must also be sound, i.e. no malicious prover should be able to convince a verifier of a false statement.

In this work we consider a stronger soundness requirement where the prover should not be able to convince the verifier even when having the power of resetting the verifier's machine (namely, having

black-box access to the verifier). This notion of soundness, referred to as *resettable soundness*, was first introduced by Barak, Goldwasser, Goldreich and Lindell (BGGL) in [BGGL01], and is particularly relevant for cryptographic protocols being executed on embedded devices such as smart cards. Barak et al. in [BGGL01] prove that, unless $\mathbf{NP} \subseteq \mathsf{BPP}$, interactive proofs for **NP** cannot be zero knowledge and resettably-sound at same time, unless the security proof is based on a *non-black-box* simulation strategy. (Indeed, a resetting prover has the same special access to the verifier as a black-box simulator, and it can break soundness by running the simulator's strategy.) In [BGGL01], they also provide the first resettably-sound zero-knowledge arguments for NP which relies on the non-black-box zero-knowledge protocol of [Bar01] and on the existence of collision-resistant hash-functions (CRHFs).

Recently, Chung, Pass and Seth (CPS) [CPS13] showed that the minimal assumption for non-black-box zero-knowledge is the existence of one-way functions (OWFs). In their work they provide a new way of implementing Barak's non-black-box simulation strategy which requires only OWFs. Independently, Bitansky and Paneth [BP13] also showed that OWFs are sufficient by using a completely new approach based on the impossibility of approximate obfuscation.

Common to all the above constructions [BGGL01, CPS13, BP13], beside the need of non-black-box simulation, is the need of non-black-box use of the underlying cryptographic primitives.

Before proceeding, let us explain the meaning of black-box versus non-black-box use of a cryptographic primitive. A protocol makes black-box use of a cryptographic primitive if it only needs to access the input/output interface of the primitive. On the other hand, a protocol that relies on the knowledge of the implementation (e.g., the circuit) of the primitive is said to rely on the underlying primitive in a non-black-box way. A long line of work [IKLP06, Hai08, PW09, CDSMW09, Wee10, Goy11, LP12, GLOV12] starting from the seminal work of Impagliazzo and Rudich [IR88] aimed to understand the power of the non-black-box access versus the black-box access to a cryptographic primitive. Besides strong theoretical motivation, a practical reason is related to efficiency. Typically, non-black-box constructions are inefficient and as such, non-black-box constructions are used merely to demonstrate "feasibility" results. A first step towards making these constructions efficient is to obtain a construction that makes only black-box use of the underlying primitives.

In the resettable setting non-black-box simulation is necessary. In this work we are interested in understanding if non-black-box use of the underlying primitive is necessary as well.

Very recently, [GOSV14a] constructed a public-coin ZK argument of knowledge based on CRHFs in a black-box manner. They provided a non black-box simulator but a black-box construction based on CRHFs. Such a reduction is referred to as a *semi black-box* construction (see [RTV04] for more on different notions of reductions). By applying the [BGGL01] transformation, their protocol yields the first (semi) black-box construction of a resettably-sound ZK argument that relies on CRHFs.

In this paper, we address the following open question:

> *Can we construct resettably-sound ZK arguments under the minimal assumption of the existence of a OWF where the OWF is used in a black-box way?*

1.1 Our Results

We resolve this question positively. More formally, we prove the following theorem.

Theorem 1 (Informal)**.** *There exists a (semi) black-box construction of an O(1)-round resettably-sound zero-knowledge argument of knowledge for every language in NP based on one-way functions.*

It might seem that achieving such result is a matter of combining techniques from [GOSV14a], which provides a "black-box" implementation of Barak's non-black-box simulation and [CPS13],

which provides an implementation of Barak's technique based on OWFs. However, it turns out that the two works have conflicting demands on the use of the underlying primitive which make the two techniques "incompatible".

More specifically, the construction presented in [GOSV14a] crucially relies on the fact that a collision-resistance hash-function is publicly available. Namely, in [GOSV14a] the prover (and the simulator) should be able to evaluate the hash function *on its own* on any message of its choice at any point of the protocol execution. On the other hand, the protocol proposed in [CPS13] replaces the use of hash functions with digital signatures (that can be constructed from one-way functions), namely, the hash values are replaced with signatures. Furthermore, they require the signing key to be hidden from the prover: the only way the prover can obtain a signature is through a "signature slot". Here the prover presents the verifier a fixed number of messages and the verifier responds with the signatures of these messages using a privately chosen signing key. Consequently in [CPS13], the prover: (1) cannot compute signature on its own, (2) cannot obtain signatures at any point of the protocol (but only in the signature slot), and (3) cannot obtain an arbitrary number of signatures.

Next, we explain the prior works in detail.

1.2 Previous Techniques and Their Limitations

We briefly review the works [GOSV14a] and [CPS13] in order to explain why they have conflicting demands. As they are both based on Barak's non black-box simulation technique, we start by describing this technique.

Barak's Non-Black-Box Zero Knowledge [Bar01]. Barak's ZK protocol for an NP language L is based on the following idea: a verifier is convinced if one of the two statements is true: 1) the prover knows the verifier's next-message-function, or 2) $x \in L$. By definition a non-black-box simulator knows the code of the next-message-function of the verifier V^* which is a witness for statement 1, while the honest prover has the witness for statement 2. Soundness follows from the fact that no adversarial prover can predict the next-message-function of the verifier. Zero-knowledge can be achieved by employing a witness-indistinguishable (WI) proof for the above statements. The main bottleneck in translating this beautiful idea into a concrete construction is the size of statement 1. Since zero-knowledge demands simulation of arbitrary malicious non-uniform PPT verifiers, there is no *a priori* bound on the size of the verifier's next-message circuit and hence no strict-polynomial bound on the size of the witness for statement 1. Barak and Goldreich [BG02] in their seminal work show how to construct a WI argument that can *hide* the size of the witness. More precisely, they rely on Universal Arguments (UARG) that can be constructed based on collision-resistant hash-functions (CRHFs) via Probabilistically Checkable Proofs (PCPs) and Merkle hash trees (based on [Kil92]). PCPs allow rewriting of proofs of NP statements in such a way that the verifier needs to check only a few bits to be convinced of the validity of the statement. Merkle hash-trees [Mer89], on the other hand, allow committing to strings of arbitrary length with the additional property that one can selectively open some bits of the string by revealing only a small *fixed* amount of decommitment information. More precisely, a Merkle hash-tree is constructed by arranging the bits of the string on the leaves of a binary tree, and setting each internal node as the hash of its two children: a Merkle tree is a *hash chain*. The commitment to the string corresponds to the commitment to the root of the tree. The decommitment information required to open a single bit of the string is the path from the corresponding leaf in the tree to the root along with their siblings. This path is called *authentication* path. To verify the decommitment, it is sufficient to perform a *consistency check* on the path with the root previously committed. Namely, for each node along the path check if the node corresponds to the hash of the children, till the last node that must correspond to the root.

Merkle trees allow for committing strings of super-polynomial length with trees of poly-logarithmic depth.

In a UARG, on a high-level, the prover first commits to the PCP-proof via a Merkle hash-tree. The verifier responds with a sequence of locations in the proof that it needs to check and the prover opens the bits in the respective locations along with their authentication paths. While this approach allows constructing arguments for statements of arbitrary polynomial size, it is not zero-knowledge because the authentication paths reveal the size of the proof (since the length of the path reveals the depth of the tree). To obtain witness indistinguishability Barak's construction prevents the prover from revealing the actual values on any path. Instead, the prover commits to the paths and then proves that the opening of the commitments corresponds to *consistent* paths leading to *accepting* PCP answers. A standard ZK is sufficient for this purpose as the size of the statement is strictly polynomial (is fixed as the depth of the Merkle tree). Such ZK proofs, however, need the code of the CRHFs used to build the Merkle-hash tree.

Black-box implementation of Barak's non-black-box simulation strategy [GOSV14a]. Recently, Goyal, Ostrovsky, Scafuro and Visconti in [GOSV14a] showed how to implement Barak's simulation technique using the hash function in a black-box manner.

The first observation in [GOSV14a] is the following. In order to use a hash function in a black-box manner, the prover cannot prove the consistency of the paths by giving a proof, but instead it should reveal the paths and let the verifier recompute the hash values and verify the consistency with the root of the tree. The problem is that to pass the consistency check an honest prover can open paths that are at most as long as the real tree, therefore revealing its size. Instead we need to let the verifier check the path while still keeping the size of the real tree hidden.

To tackle this, they introduce the concept of an *extendable* Merkle tree, where the honest prover will be able to extend any path of the tree *on the fly*, if some conditions are met. Intuitively, this solves the problem of hiding the size of the tree — and therefore the size of the proof— because a prover can show a path for any possible proof length.

To implement this idea they construct the tree in a novel manner, as a sequence of *"LEGO"* nodes, namely nodes that can be connected to the tree on the fly. More concretely, observe that checking the consistency of a path amounts to checking that each node of the path corresponds to the hash of the children. This is a *local* check that involves only 3 nodes: the node that we want to check, say A, and its two children say $\mathsf{leftA}, \mathsf{rightA}$, and the check passes if A is the hash of $\mathsf{leftA}, \mathsf{rightA}$. The LEGO idea of [GOSV14a] consists of giving the node the following structure: a node A now has two fields: $\mathsf{A} = [label, encode]$, where *label* is the hash of the children $\mathsf{leftA}, \mathsf{rightA}$, while *encode* is some suitable encoding of the *label* that allows for black-box proof. Furthermore, *label* is not the hash of the *label* part of $\mathsf{leftA}, \mathsf{rightA}$, but it is the hash of the *encode* part. Thus there is not direct connection between the hash values, and the hash chain is broken. The second ingredient to guarantee binding, is to append to each node a proof of the fact that "*encode* is an encoding of *label*". (This property is referred to as $\mathsf{in\text{-}node}$ consistency in [GOSV14a]).

Note that now the tree so constructed is not an hash chain anymore, the chain is broken at each step. However, it is still binding because there is a proof that connects the hash with its encoding. More importantly note that, if one can cheat in the proof added to the node, then one can replace/add nodes. Thus, provided that we introduce a *trapdoor* for the honest prover (and the simulator) to cheat in the proof, the tree is *extendable*. Therefore, now we can let the verifier check the hash value by itself, and still be able to hide the depth of the committed tree.

For the honest prover, which does not actually compute the PCP proof, the trapdoor is the witness for the theorem "$x \in L$". For the simulator, which is honestly computing the PCP proof and hence the Merkle tree, the trapdoor is the size of the real tree (that is committed at the very

beginning). Namely, the simulator is allowed to cheat every time it is required to answer to PCP queries that are beyond the size of the committed PCP.

The final piece of their construction deals with committing the actual code of the verifier which also can be of an arbitrary polynomial size. The problem is that to verify a PCP proof, the verifier needs to read the entire statement (in this case the code of V^*) or at least the size of the statement to process the queries. But revealing the size will invalidate ZK again. [GOSV14a] get around this problem by relying on Probabilistically-Checkable Proof of Proximity (PCPP) [BSGH+06] instead of PCPs which allow the verifier to verify any proof and arbitrary statement by querying only a few bits of the statement as well as the proof. For convenience of the reader, we provide a very simplified version of [GOSV14a]'s protocol in Fig. 1.

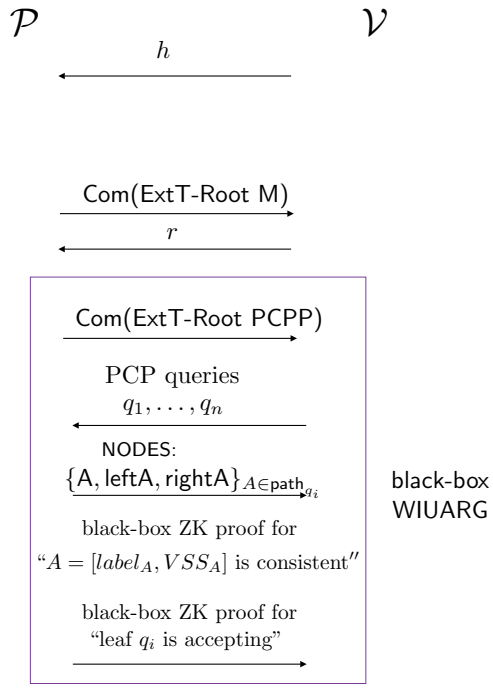

Figure 1: [GOSV14a]

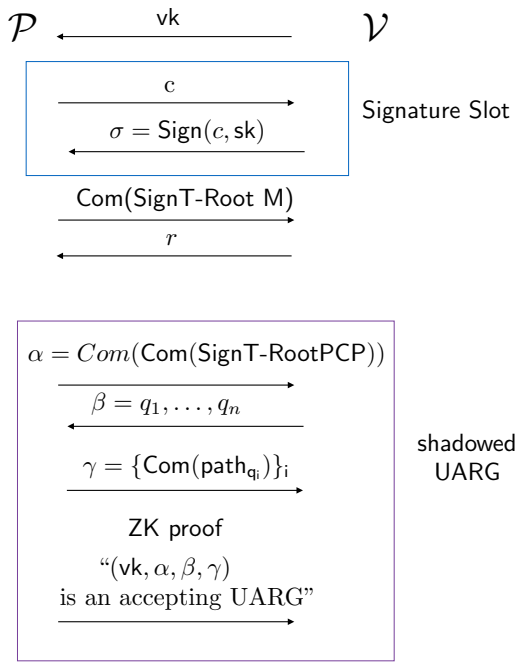

Figure 2: [CPS13]

Summing up, some of the key ideas that allow [GOSV14a] for a black-box use of the hash function are: (1) the prover unfolds the paths of the Merkle tree so that the verifier can directly check the hash consistency, (2) the prover/simulator can arbitrarily extend a path on the fly by computing fake LEGO nodes and cheat in the proof of consistency using their respective trapdoors.

The work of [GOSV14a] retains all the properties of Barak's ZK protocol, namely, is public-coin and constant-round (therefore resettable sound due to [BGGL01]), and relies on the underlying CRHF in a black-box manner.

Non black-box simulation using OWFs [CPS13]. Chung, Pass and Seth [CPS13] showed how to implement the non-black-box simulation technique of Barak using OWFs.

The main idea of CPS is to notice that digital signature schemes — which can be constructed from one-way functions — share many of the desirable properties of CRHFs, and to show how to appropriately instantiate (a variant of) Barak's protocol using signature schemes instead of using CRHFs. More precisely, CPS show that by relying on strong fixed-length signature schemes, one can construct a *signature tree* analogous to the Merkle hash-tree that allows compression of arbitrary length messages into fixed length commitments and additionally satisfies an analogue collision-

resistance property. The soundness of such a construction will have to rely on the unforgeability (i.e. collision-resistance) of the underlying signature scheme. Hence it must be the case that is the verifier the one generating the secret key and the signatures for the prover. Towards this, CPS adds a signature slot at the beginning of the protocol. More precisely, first the verifier generates a signature key-pair sk, vk and sends only the verification key vk to the prover. Next, in a "signature slot", the prover sends a commitment c to the verifier, and the verifier computes and returns a valid signature σ of c (using sk). The simulator constructs the signature tree by rewinding the (malicious) verifier, and then succeeds in the WIUARG proof as in Barak's protocol. While the simulator can use the signature slot to construct the signature tree, we cannot require the honest prover to construct any tree since it is not allowed to rewind the verifier. To address this, CPS uses a variant of Barak's protocol due to Pass and Rosen [PR05], which relies on a special-purpose WIUARG, in which the honest prover never needs to perform any hashing (an early version of Barak's protocol also had this property). The idea here is that since there exist public-coin UARGs, the prover can first engage in a shadowed UARG where the prover merely commits to its messages and then in a second phase proves using a witness-indistinguishable proof that either $x \in L$ or the messages it committed to constitute a valid UARG proving that it possesses a fake witness. This will allow the honest prover to "do nothing" in the shadowed UARG and use the witness corresponding to x in the second phase while the simulator will commit to valid messages in the UARG by first obtaining signatures from the signature slot through rewinding and then use the generated UARG messages as the witness in the second phase.

The resulting protocol is not public-coin, [CPS13] nevertheless shows that it suffices to apply the PRF transformation of BGGL to obtain a protocol that is resettably-sound. We provide a very informal pictorial description of the CPS protocol in Fig. 2. It seems inherent that the CPS protocol needs a shadowed UARG, and hence proving anything regarding this shadowed argument needs to use the underlying OWF in a non-black-box manner.

Competing requirements of [GOSV14a] and [CPS13]. Summing up, in [GOSV14a], in order to use the CRHF in a black-box manner, the prover is required to open the paths of the Merkle Tree corresponding to the PCPP queries and let the verifier check their consistency. To preserve size-hiding, the prover needs the ability to arbitrarily extend the paths of the tree by *privately* generating new nodes and this is possible because the prover can compute the hash function on its own. In contrast, in [CPS13] the prover cannot compute nodes on its own, but it needs to get signatures from the verifier. Therefore the protocol in [CPS13] is designed so that the prover never has to use the signatures.

In this work we show a technique for using the benefits of the signature while relying on the underlying OWF in a black-box manner and we explain this in the next section.

1.3 Our Techniques

Our goal is to implement the resettably-sound ZK protocol based on Barak's non-black-box simulation technique using OWFs in a black-box manner. We have illustrated the ideas of [GOSV14a] to implement Barak's ZK protocol based on extendable Merkle hash-tree that uses a CRHF only as a black-box, and the ideas of [CPS13] that show how to compute a Merkle signature-tree based on (non-black-box use of) OWFs.

The natural first step to reach our goal is then to take the protocol of [GOSV14a] and implement the extendable Merkle tree with signatures instead of CRHF. As mentioned before, this replacement cannot work because the crucial property required to extend the tree is that the prover computes nodes on its own. If we replace CRHF with signatures, then the prover needs to ask signatures from the verifier for every node. This means that any path computed by the prover is already known by

the verifier (even the concept of "opening a path" does not seem make to much sense here). But instead we need the ability to commit to a tree and then extend it *without* the verifier knowing that we are creating new nodes.

We are able to move forward using the following facts underlying [GOSV14a]'s protocol. First, although it is true that the prover needs to extend paths and prove consistency, it does so by cheating in every proof of consistency. Indeed, recall that a node is a pair $(label, encode)$ and the consistency between and *label* and *encode* is proved via ZK, the prover cheats in this proof using the witness for "$x \in L$" as a trapdoor. Under closer inspection we notice that the prover does not have to compute any tree; it just needs to compute the paths on-the-fly when asked for it. Indeed, an equivalent version of [GOSV14a]'s protocol would be the following. The prover commits to the root by just committing to a random string.[1] Then, when it sees the PCPP queries $q_1, \ldots, q_n$ it computes *on-the-fly* the corresponding paths, and is able to prove consistency of the paths with the previously committed roots by cheating in the proofs. Finally, we point out that the hash function is required only to compute the *label* part of the node, while the *encode* part can be computed by the prover on its own.

Armed with the above observations, we present our idea. As in [CPS13] we place a signature slot at the very beginning of the protocol. This signature slot will allow the simulator to get unbounded number of the signatures by rewinding the verifier, and ultimately to construct the extendable Merkle trees. After the signature slot, the prover commits, using an instance-dependent trapdoor commitment, to the roots of the extendable Merkle trees. This trapdoor commitments allows the possessor of the witness to equivocate any commitment. Therefore, the roots committed by the prover – who knows the witness – are not binding. Next, when the prover receives the PCPP queries $q_1, \ldots, q_n$, it computes the paths on-the-fly with the help of the verifier. Namely, it computes the *encoding* part of the nodes (for the prover these encodings will be equivocal commitments to the zero-string), and send them to the verifier who computes the *label* part by "hashing" the encodings (in the proper order), namely, by computing their signature. Therefore, deviating from [CPS13], we introduce a second signature slot where the prover gets the signatures required to construct the paths. Once all the signatures have been computed by the verifier, the paths are finally completed. Now the prover can proceed with proving that the paths just computed are *consistent* with the roots previously committed and lead to accepting PCPP answers (this black-box proof follows the ideas of [GOSV14a]).

Interestingly in this game the verifier knows that the prover is cheating and the paths cannot be possibly consistent, but he is nevertheless happy because the only way the prover can cheat in the consistency proof is by using the witness for $x \in L$, which basically proves that the theorem is true.

Now, let's look at the simulation strategy. The simulator honestly computes extendable Merkle signature-trees to commit to the machine V^* and to the PCPP, using the first signature slot. Then, when the verifier sends PCPP queries $q_1, \ldots, q_n$ it answers in the following way. For the queries that do not concern the real tree, the simulator sends *encode* parts which are just commitments of random strings (in the same way as an honest prover) and later on it will cheat in the proof of consistency by using its *trapdoor* (as in [GOSV14a] the trapdoor for the simulator is the size of the real tree). For the queries that hit the real tree, the simulator sends the same commitments that it sent in the first signature slot and that were used to compute the real tree. Indeed, for these nodes the simulator must prove that they are truly consistent, and it can do so by forcing in the view of

[1] This is not exactly true. In [GOSV14a] the prover actually commits to the hash of two random nodes. However, in our paper we assume that any commitment is computed using an instance-dependent trapdoor commitment, computed on instance x. The prover, that knows the witness w, can just commit to a random string and then equivocate later accordingly.

the verifier the same paths that it already computed for the real tree.

Thus, for the simulation to go through it should be the case that the signatures that the simulator is collecting in the second signature slot, match the ones that it obtained in the first signature slot and that were used to compute the real tree.

Unfortunately, with the protocol outlined above we do not have such guarantee. This is due to two issues. First, the verifier can answer with some probability in the first slot and with another probability in the second slot, therefore skewing the distribution of the output of the simulator. Second, the verifier might not compute the signature deterministically: in this case the signatures obtained in the first slot and used to compute the real tree will not match the signatures obtained in the second slot, where the paths are "re-computed", and thus the simulator cannot prove that the nodes are consistent. We describe the two issue in details and we show how we change the construction to fix each issue.

Issue 1. V^* aborts in the two signature slots with different probabilities. We describe this issue with the following example. Consider a malicious verifier that aborts on all messages that start with bit 0 in the first signature slot and aborts on all sets of messages in the second slot if 3/4 of them start with bit 1. The honest prover will succeed with probability close to a 1/2 since the verifier will abort in the first message with probability 1/2 and not abort with high probability in the second slot.[2] The simulator on the other hand can only obtain signatures of commitments that start with bit 1 in the first slot and has to use the same commitments in the second slot. This means that all the commitments sent by the simulator in the second slot will begin with the bit 1 and the verifier will always abort. Hence the simulator can never generate a view.

The way out is to come up with a simulation strategy ensuring that the distribution fed in the second slot is indistinguishable to the honest prover's messages.

Fixing for issue 1. We first amplify the probability that the verifier gives a signature in the first signature slot by requesting the verifier to provide signatures for $T = O(n^{c+1})$ random commitments instead of just one. Using a Yao-type hardness-amplification, we can argue that if the verifier provides valid signatures with non-negligible probability then we can obtain signatures for at least $1 - \frac{1}{n^c}$ fraction of random tapes for the commitment. Lets call these random tapes good. Now if $k << n^c$ commitments are sent in the second slot, with probability at least $1 - \frac{k}{n^c}$ over random commitments made in the second slot, all of them will be good. This is already promising since the verifier at best will try to detect good messages in the second slot and with high probability all of them are good. However there is still a non-negligible probability (i.e. $\frac{k}{n^c}$) that the verifier could abort in this simulation strategy.

Towards addressing this issue, we make the simulator modify the description of the verifier's code so that it is 2^d times larger and the running time is 2^u times slower than the actual code, where d and u are chosen randomly from $1, \ldots, \log n$. The modified code however would still need to have the same functionality. This can be achieved by padding the description and adding "no-op" instructions. The main advantage the simulator has by doing this is that it has a choice of $\log n$ levels to disguise the verifier's code into (and $\log n$ levels in the proof tree to disguise the proof into). Since the actual size and proof lengths are only committed, the verifier will not apriori know which among the $\log n$ levels is used by the simulator.[3]

Next, when it is time to provide the commitments in the second signature slot, the simulator needs to plug in the actual commitments used to build the trees. Let us focus on one of the trees and the other tree will follow analogously. For every level, commitments to nodes are presented for the k

[2] We assume here that random commitments are equally likely to have their first bits 0 or 1.

[3] Without this modification, the verifier based on its own running time and description will in fact know the levels that the simulator will use.

paths corresponding to the k PCPP queries. We will assume that the number of nodes/commitments sent for each of these levels is a fixed number M. There is a fixed level d^* where the simulator needs to plug-in the actual commitments, i.e., the commitments used in the first signature slot to construct the tree. In contrast a prover sends all freshly generate commitments. If the simulator just plugged them in there while sending all fresh commitments for the nodes of the remaining levels, it might skew the distribution. The reason is that the commitments for d^* where good, i.e. signatures for them were obtainable from the first slot. So by plugging in good commitments only for level d^* in the second slot, the simulator might be increasing the total number of good commitments sent in the second signature slot. This might be detected by the verifier. Therefore, we need to argue that with high-probability the freshly generated commitments sent in the second signature slots would have been good in the first slot as well. If this is true that we can argue that V^* cannot distinguish the transcript where d^* is made of fresh commitments or previously generated good commitments.

Towards arguing this, we construct the following simulator. It will first generate M fresh commitments to dummy nodes for *all* levels, in particular also for level d^* for which it should use actual commitments. Let D be the set of levels that the simulator could have disguised the verifiers code into. So $d^* \in D$ and there are $\log n$ levels in D. Next, it will investigate how many of these levels are good, namely, for which of these levels the fresh commitments generated for that level in the second signature slot, would have been also good when submitted in the first signature slot. We will say that a level is good if all the commitments generated for this level are good commitments, namely signatures for these commitments were obtainable from the first slot. Suppose that some $d' \in D$ is a good level. Then it will swap out the fresh commitments temporarily generated for level d^* and plug them to level d'. It will then re-plug the actual commitments (the ones that allow it to prove consistency) to the level d^*. This swapping will ensure that the total number of good commitments will not increase, M good ones are removed and M good ones are added. Arguing that the exact distribution does not change requires some more work but in essence this trick above will suffice.

Issue 2. V^* does not compute signatures deterministically. Our protocol requires the verifier to compute the signature deterministically, namely, the randomness used to compute the signature must be derived from a PRF whose key is sampled along with the parameters vk, sk and is part of the secret key. However, in the construction that we outlined before we cannot enforce a malicious verifier from signing deterministically. As mentioned before, if there verifier gives different (correct) signatures for the same node in the first and second slot, the simulator cannot proceed with the proof of consistency.

The only way to catch the verifier is to demand from the verifier a proof that the signatures are computed deterministically. Because we can only use the signature scheme in a black-box manner we cannot solve the problem by attaching a standard proof of consistency.

Fixing for issue 2. The high-level idea is that we will force the verifier to be honest using a cut-and-choose mechanism. More precisely, we will require the verifier to provide signatures to n different keys instead of just one in the signature slots. Together with the signature keys the verifier will also attach the commitment of the randomness used in the signature key generation algorithm (the randomness determines the PRG seed that is used to deterministically sign the messages).

After the first signature slot, the prover asks the verifier to reveal $n/2$ keys (by requiring the verifier to decommit to the randomness used to generate the keys) and checks if, for the revealed keys, the signatures obtained so far were computed honestly. The prover proceeds with the protocol using the remaining half of the keys, namely by committing to $n/2$ roots, and obtaining $n/2$ sets of PCPP queries from the verifier. Later, after the second signature slot is completed, the prover will ask to open half of the remaining keys, namely $n/4$ keys, and checks again the consistency of the signatures obtained so far. If all checks pass, the prover is left with paths for $n/4$ trees/PCPP

queries for which he has to prove consistency. Due to the cut-and-choose, we know that most of the remaining signatures were honestly generated, but not all of them. Therefore, at this point we will not ask the prover to prove that all the $n/4$ trees are consistent with the paths. Instead, we allow the prover to choose one coordinate among the $n/4$ left and to prove that the tree in this coordinate is consistent with the paths.

Allowing the prover to choose the tree for which to prove consistency, does not give any advantage compared to the original solution where there was only one tree. On the other hand, having a choice in the coordinate allows the simulator to choose the tree for which it received only consistent signatures and for which it will be able to provide a consistency proof. You can see the coordinate as another *trapdoor*. Namely, in the previous construction, the simulator commits to the real depth of the tree, so that in the consistency proof it can cheat for any path which goes beyond the real depth. We follow exactly the same concept by adding the commitment of the coordinate. The simulator commits to the coordinate in which he wants to provide consistency proof, so, in all the other coordinates he is allowed to cheat. Finally, because we are in the resettable setting, we require the prover to commit in advance to the coordinates that he wants the verifier to open. If this was not the case then the prover can rewind the verifier and get all the secret keys.

2 Protocol

Overview. The protocol starts with a signature slot, where the prover sends T commitments, and the verifier signs all of them. Then, as per Barak's construction, the prover sends a message z, which is the commitment to a machine M, the verifier sends a random string r, and finally the prover sends a commitment to a PCP of Proximity proof for the theorem: "the machine M committed in z is such that $M(z) = r$ in less then $n^{\log n}$ steps". M is the hidden theorem to which the verifier has only oracle access to[4]. The above commitments are commitment to the roots of (extendable) Merkle signature-trees (that we will describe in details later). Next, the verifier sends the PCPP queries. Each PCPP query is a pair of indices, one index for the hidden theorem M and one for the PCPP proof. The verifier expects to see a path for each index.

At this point the prover needs to compute such paths. As we mentioned in the introduction, in a signature tree a path cannot be computed by the prover only: each nodes consists of two parts, the signature of the children, called *label*, and the encoding of the label, that we called *encode*. Thus, the prover continues as follows. For each path, he computes the *encode* parts of the nodes belonging. It then sends all these "half" nodes to the verifier. The verifier computes the *label* parts for each node by signing the *encode* part and send them back to the prover. This is the second signature slot. Once all paths are completed, the prover starts the proof stage. Using a ZK protocol he proves that: (1) the paths are consistent with the root committed before, (2) the leaves of the paths open to accepting PCPP answers, in a black-box manner. How does the prover pass the proof stage? The prover lies by computing all commitments using instance-dependent equivocal commitments and adaptively cheating in the opening. How does the simulator pass the proof stage? The simulator computes the trees for the machine M and the PCPP for real by rewinding the verifier in the first signature slot. On top of this outlined construction we use cut-and-choose to force the verifier to compute the signatures deterministically. This concludes the high-level description of the protocol.

Now we need to show concretely how to compute the proofs of consistency using the signature and the commitment scheme only as *black-box*. This requires to go into the details of the (extendable)

[4]In PCP of Proximity the theorem is a pair (a, y) where a is the public theorem and is read entirely by a verifier, and y is the hidden theorem, that the verifier has only oracle access to. A verifier for a PCP of Proximity queries both the proof and the hidden theorem y.

Merkle signature-tree and the mechanism for size hiding introduced in [GOSV14a]. We provide such details in the next section.

2.1 Details of the Construction

Some of the ideas presented in this section are adapted from [GOSV14a, GOSV14b].

String Representation. To allow black-box proofs about a committed string, the first ingredient is to represent the string with a convenient encoding that enables to give black-box proofs on top. For this purpose, following [GOSV14a, GLOV12] we use Verifiable Secret Sharing (VSS for short). VSS is a two-stage protocol run among $n+1$ players. In the first stage, called $\texttt{Share}(s)$, the dealer distributes a string s among the n players, so that, any t players (where $t = n/c$ for some constant $c > 3$) colluding cannot reconstruct the secret. The output of the `Share` phase is a set of VSS views $\mathsf{S}_1, \ldots, \mathsf{S}_n$ that we call VSS shares. In the second stage, called $\texttt{Recon}(\mathsf{S}_1, \ldots, \mathsf{S}_n)$, any $(n-t)$ players can reconstruct the secret s by exchanging their VSS shares. The scheme guarantees that if at most t players are corrupted the `Share` stage is hiding and the `Recon` stage is binding (this property is called t-robustness). Such VSS is called (t, n)-secure.

To commit to any string s, the prover will proceed as follows. It first run, in his head, an execution of a perfectly (t, n)-secure VSS among $n+1$ imaginary players where the dealer is sharing s, obtaining n views: $\mathsf{S}[1], \ldots, \mathsf{S}[n]$. Then it commits to each share separately using a statistically binding commitment.

Black-box proof of a predicate. With the VSS representation of strings, now the prover can commit to the string and prove any predicate about the committed string, using MPC-in-the-head. Specifically, let s be the string, $[\mathsf{S}[1], \ldots, \mathsf{S}[n]]$ be the VSS shares of s and let ϕ be a predicate. The prover wants to prove that $\phi(s)$ is true without revealing s.

We define $\mathcal{F}_\phi$ as the n-party functionality that takes in input a VSS share $\mathsf{S}[p]$ from each player p, and outputs $\phi(\texttt{Recon}(\mathsf{S}[1], \ldots, \mathsf{S}[n]))$. To prove $\phi(s)$, the prover runs a (t, n)-perfectly secure MPC-in-the-head among n players for the functionality $\mathcal{F}_\phi$. Each player participates to the protocol with input a VSS share of the string s. Then the prover commits to each view of the MPC-in-the-head so computed.

The verifier can check the proof by asking the prover to open t views of the VSS *and* the MPC protocol, and check that such views are consistent (in Appendix A we provide more details on this consistency check) with an honest execution of the MPC protocol. Zero-knowledge follows from the t-privacy and soundness follows from the t-robustness of the MPC/VSS protocols. In a bit more details, t-robustness roughly means that, provided that the predicate to be proved is false and that the prover does not know in advance which views will be opened, corrupting only t players is not sufficient to convince the verifier with consistent views. On the other hand, by corrupting more than t players, the prover can be caught with high probability.

(Extendable) Merkle Signature-Tree. As we discussed in the introduction, a node in an extendable Merkle tree is a pair $[label, encode]$. In our signature Merkle tree, the field *label* is a vector of signatures (computed by the verifier), and the field *encode* is a vector of commitments of VSS shares of *label*.

Specifically, let γ be any node, let $\gamma 0 = [label^{\gamma 0}, \{\mathsf{Com}(\mathsf{S}^{\gamma 0}[i])\}_{i\in n}]$ be its left child, and $\gamma 1 = [label^{\gamma 1}, \{\mathsf{Com}(\mathsf{S}^{\gamma 1}[i])\}_{i\in n}]$ be its right child. Node γ is computed as follows: The label part is: $label^{\gamma} = \{\mathsf{Sign}_{\mathsf{sk}}(\mathsf{Com}(\mathsf{S}^{\gamma b}[i]))\}_{b\in\{0,1\}, i\in n}$. The *encode* part is computed in two steps: First, compute shares $\mathsf{S}^{\gamma}_1, \ldots, \mathsf{S}^{\gamma}_n \leftarrow \texttt{Share}(label)$; next commit to each share separately.

At **leaf level**, the $label^{\gamma} = s[\gamma]$, namely the γ-th bit that we want to commit.

Hiding the size of the tree. The size of the string committed, and hence the depth of the corresponding tree, is not known to the verifier and it must remain hidden. Specifically, the verifier should not know the size of the machine M and of the PCPP proof. Hence, the verifier will send a set of PCPP queries for each possible depth of the tree for the PCPP. Namely, for each possible depth $j \in [\log^2 d]$, V sends $\{q_{i,j}, p_{i,j}\}_{i \in k}$[5], where k is the soundness parameter.

Note that the prover (actually, the simulator) will be able to correctly answer only to the queries hitting the depth of the real tree, and we want that this is transparent to the verifier. This is done by adding the commitment to the depth of the real tree at the beginning, and then proving, for each query's answer, that either the query is correctly answered or the query was not valid for the committed size. This commitment of the depth needs to be in the same format of the *encode* part of the nodes (i.e., it will be a commitment of VSS shares) because it will be used in the black-box proofs of consistency.

Black-box proofs about the leaves of the tree. As it should be clear by now, the proof consists in convincing the verifier that the paths shown are consistent with the committed root of the tree and open to valid answers of PCPP queries. This is done as follows. Attached to each path, there is a proof of consistency. This proof serves to convince the verifier that the path is consistent with the previously committed root. Attached to each set of paths (there is a set of paths for each depth $j = 1, \ldots, \log^2 n$), there is a proof of acceptance. This proof serves to convince the verifier that those paths open to bits of PCPP proof/hidden theorem M that are accepting.

Both the prover and the simulator will cheat in this proof, but in different ways. The prover cheats in all proofs by equivocating the commitments (using the witness as trapdoor). The simulator cheats in all proofs concerning fake paths by using the commitments of the depth as a trapdoor. Namely, the simulator will prove that either the path is consistent/accepting, or the path is not on the real tree. For the paths of the real tree, the prover will honestly compute the proof. We now describe each step of the proof in more details.

Depth. To commit to the depth depth of the tree, the prover/simulator proceeds as follows. First, compute shares: $\{\mathsf{S}_{\mathsf{depth}}[1], \ldots, \mathsf{S}_{\mathsf{depth}}[n]\} \leftarrow \mathtt{Share}(\mathsf{depth})$; then commits to each share.

Path. Let $p_{i,j}$ a query for a tree of depth j. A path is a sequence of nodes: $\left[label^\gamma, \{\mathsf{Com}(\mathsf{S}^\gamma[p])\}_{p \in n}\right]$, with $\gamma \in \cup_{l=j}^{l=0} p^l_{i,j}$, where notation $p^l_{i,j}$ means the first l bits of the bit-expansion of index $p_{i,j}$.

Proof that a path is consistent. Proving consistency of a path for $p_{i,j}$ amounts to prove consistency of each node along the path. For each node $\left[label^\gamma, \{\mathsf{Com}(\mathsf{S}^\gamma[p])\}_{p \in n}\right]$, along the path for $p_{i,j}$, the prover proves that $\mathtt{Recon}(\mathsf{S}^\gamma[1], \ldots, \mathsf{S}^\gamma[n]) = label^\gamma$.

This is done via an MPC-in-the-head protocol, for a functionality $\mathcal{F}_{\mathsf{innode}}$ that takes in input shares: $\mathsf{S}^\gamma[p]$, $\mathsf{S}_{\mathsf{depth}}[p]$ and the string $label^\gamma$ and outputs 1 to all players if either $\mathtt{Recon}(\mathsf{S}^\gamma[1], \ldots, \mathsf{S}^\gamma[n]) = label^\gamma$ or $\mathtt{Recon}(\mathsf{S}_{\mathsf{depth}}[1], \ldots, \mathsf{S}_{\mathsf{depth}}[n]) \neq j$.

The actual proof consists in the commitment of the views of the MPC players. The verifier verifies the proof by checking the consistency of t views.

Proof that a set of paths is accepting. For each level j, the verifier sends queries $\{p_{i,j}, q_{i,j}\}_{i \in [k]}$ and the prover opens a path for each query.

To prove that these queries are accepting, the prover computes a proof that involves the leaves of all paths. That is, the proof involves the nodes in position $\{p_{i,j}, q_{i,j}\}_{i \in [k]}$. The prover runs an MPC-in-the-head for a functionality, that we call $\mathcal{F}_{\mathsf{VerPCPP}}$, that will check that the values in those positions will accepted by a PCPP verifier. $\mathcal{F}_{\mathsf{VerPCPP}}$ takes in input in input shares: $\mathsf{S}^{p_{i,j}}[p]$, $\mathsf{S}^{q_{i,j}}[p]$,

[5] We assume that the length PCPP proof is a power of 2. Also, for the sake of simplifying the notation we use the same index j for the queries to the machine $q_{i,j}$ and the proof $p_{i,j}$, even though in reality the machine M and the corresponding PCPP proof π might lie on different levels.

$\mathsf{S}_{\mathsf{depth}}[p]$ (with $p = 1, \ldots, n$) and the public theorem; it then reconstructs the bits of the PCPP proof $\pi_{i,j} = \mathtt{Recon}(\mathsf{S}^{p_{i,j}}[1], \ldots, \mathsf{S}^{p_{i,j}}[n])$ and of the hidden theorem $m_{i,j} = \mathtt{Recon}(\mathsf{S}^{q_{i,j}}[1], \ldots, \mathsf{S}^{p_{i,j}}[n])$. It finally outputs 1 to all players iff: either the PCPP verifier accepts the reconstructed bits, i.e., $\mathsf{D}_{\mathsf{pcpx}}(m_{i,j}, \pi_{i,j}, q_{i,j}, p_{j,i}) = 1$, or if $\mathtt{Recon}(\mathsf{S}_{\mathsf{depth}}[1], \ldots, \mathsf{S}_{\mathsf{depth}}[n]) \neq j$.

The actual proof consists of the commitment of the views of the MPC players for $\mathcal{F}_{\mathsf{VerPCPP}}$. The verifier verifies the proof by checking the consistency of t views.

Verification of the proof. The verifier receives the commitments of all such views and ask the prover to open t of them. The prover will then decommits t views for *all* MPC/VSS protocol committed before. The prover will cheat in all decomitments: it will use the Simulator of the MPC in the head to open views that the verifier will accept.

The cut-and-choose. The mechanism described above is repeated n times: the verifier provides n signature keys, and the prover will compute n trees. During the protocol $\frac{3}{4}n$ of the secret keys will be revealed, so for those indexes the prover will proceed to the proof phase.

If fact, the prover will prove consistency of only one tree among the $1/4n$ trees left (but of course, we want the verifier to be oblivious about the tree that the prover is using).

Thus, the last ingredient of our construction is to ask the prover to commit to an index J among the remaining $1/4n$ indexes. This commitment is again done via VSS of J and then committing to the view[6]. As expected, such VSS will be used in the computation of $\mathcal{F}_{\mathsf{innode}}$ and $\mathcal{F}_{\mathsf{VerPCPP}}$. The functionalities now will first check if the nodes are part of the J-th tree. If not, it means that the tree is not the one that must be checked, in such a case the functionality outputs 1 to all players.

2.2 The construction

We give the outline of the final protocol. A detailed version of this protocol is provided in App. B. We remark that in any step of the protocol the randomness used by the verifier to compute its messages is derived by the output of a PRF computed on the entire transcript computed so far. *Common Input:* An instance x of a language $L \in \mathbf{NP}$ with witness relation $\mathbf{R}_L$. *Auxiliary input to P:* A witness w such that $(x, w) \in \mathbf{R}_L$.

Cut-and-choose 1.

- P_0: Randomly pick two disjoint subsets of $\{1, \ldots, n\}$ that we denote by J_1, J_2, with $|J_1| = n/2$ and $|J_2| = n/4$. Commit to J_1, J_2 using the equivocal commitment scheme.
- V_0: Run $(\mathsf{sk}_\kappa, \mathsf{vk}_\kappa) \leftarrow \mathsf{Gen}(1^n, r_\kappa)$ for $\kappa = 1, \ldots, n$. Send $\mathsf{vk}_\kappa, \mathsf{Com}(r_\kappa)$ for $\kappa = 1, \ldots, n$ to P.
- **Signature Slot 1**. P_1: Send $\mathsf{T} = O(n^c)$ commitments using the equivocal commitment scheme to 0^v (for some constant c and for v being as the size of the *encode* part). V_1: Signs each commitment.
  Check Signature Slot 1. P opens set J_1. V send sk_κ and decommitment to r_κ, for $\kappa \in J_1$. P checks that all signatures verified under key vk_κ are consistent with $\mathsf{sk}_\kappa, r_\kappa$. If not, abort.

Commitment to the Machine.

- P_2: Send equivocal commitment to the *encode* part of the root of the (extendable) Merkle signature-tree for M, and equivocal commitment to the depth of the tree, this is done for each $\kappa \in \{1, \ldots, n\}/J_1$.
- V_2: Send a random string $r \in \{0,1\}^n$. Let $(r, \mathtt{t})$ be the public theorem for the PCPP for the language: $\mathcal{L}_\mathcal{P} = \{(a = (r, \mathtt{t}), (Y)), \exists M \in \{0,1\}^* \text{ s.t. } Y \leftarrow \mathsf{ECC}(M), M(z) \rightarrow r \text{ within } \mathtt{t}$

[6] Attached to the VSS there will be also a proof that proves that $J \in \{1, \ldots, n\}/\{J_1 \cup J_2\}$.

steps} (where $\mathsf{ECC}(\cdot)$ is a binary error correcting code tolerating a constant fraction $\delta > 0$ of errors, and δ is the proximity factor of PCPP).

Commitment to the PCPP proof.

- P_3: Send equivocal commitment to the *encode* part of the root of the (extendable) Merkle signature-tree for the PCPP proof and the commitment to the depth of such tree. This is done for each $\kappa \in \{1, \ldots, n\}/J_1$.
- V_3: Send the random tapes for the PCPP queries. V and P obtain queries $(q_{i,j}, p_{i,j})$ for $i \in [k]$ and with $j = 1, \ldots, \log^2 n$.
- P_4: Send paths for $p_{i,j}, q_{i,j}$. Namely, send the *encode* part for each node along the paths $p_{i,j}, q_{i,j}$. This is done for each tree $\kappa \in \{1, \ldots, n\}/J_1$, previously committed.

Signature Slot 2 V_4: Sign the *encode* parts received from P.

Cut-and-choose 2. P opens the set J_2. V sends sk_κ and decommitment to r_κ, for $\kappa \in J_2$. P checks that all signatures verifier under key vk_κ are consistent with $\mathsf{sk}_\kappa, r_\kappa$. If not, abort.

Proof.

- P_5: Commit to a random index $J \in \{1, \ldots, n\}/\{J_1 \cup J_2\}$.

  Then compute the proof of consistency of each path, and the proof of acceptance for each set of queries (as explained earlier), for each of the remaining trees. The proofs (i.e., the views of the MPC-in-the-head) are committed using an *extractable* equivocal commitment scheme.
- V_5: Select t players to check: Send indexes $p_1, \ldots, p_t$.
- P_6: Compute the t VSS shares and the t views of the MPC protocols that will make the verifier accept the proof. This is done by running the simulator guaranteed by the perfect t-privacy of the MPC protocols for the proofs. P then equivocates the previously committed views so that they open to this freshly computed views.
- V_6: Accept if all the opened views are consistent and output 1.

Completeness essentially follows from the perfect simulatability of the semi-honest parties in the MPC-in-the-head protocols and equivocability of the equivocal commitment scheme.

Zero-Knoweldge: We provide a high-level description of our simulator. The simulator internally incorporates V^* and proceeds as follows. It commits to sets J_1 and J_2 and proceeds to the first signature slot. After the first signature slot, it opens the set J_1 and obtains from V^* half of the secret keys (and the randomness used to generate the signatures under those keys). If the verifier fails to provide correct or consistent signatures, the simulator aborts. Otherwise, it sets up a procedure $\mathsf{GetSign}$ that will help the simulator to obtain signatures for arbitrary messages. This it achieves via Goldreich-Kahan approach. It estimates the probability with which the verifier replies in the signature slot and then uses that probability to identify the number of times $\mathsf{GetSign}$ needs to rewind in the slot to get a signature. This is the same approach pursued in [CPS13]. However, recall that we need to boost the probability of obtaining signatures. We let the verifier sign T commitments simultaneously. Using a Yao-type amplification, we can show that for most messages, a signature can be obtained. Furthermore, we can show that for at least $n/2 - n/10$ of the remaining keys, the signature obtained on a commitment of m will be consistent with the deterministic signing procedure with probability at least $1 - O(\log^2(n)/n)$. This will essentially follow from the cut-and-choose mechanism.

Once the procedure is established, the simulator generates the Merkle trees for the description of the code. As mentioned earlier, the simulator chooses $d, u \in [\log n]$ randomly and alters the

description size and running time by a factor of 2^d and 2^u respectively. For the altered code, it generates the Merkle signature tree for the description and the proof. Every time a signature needs to be obtained on a message m, the simulator invokes GetSign procedure T times each time requesting a signature on a new commitment of m. This ensures that the simulator will receive a signature of a commitment of m with high-probability for most keys. Once the tree is constructed, the simulator sends the encode part of the root (which is a vector of commitments of VSS shares), and commits to the VSS shares of the depth of the tree. Next, it receives the challenge string r and repeats the same process to construct the Merkle-signature tree for the proof.

Following this, the verifier provides randomness required to generate the PCPP queries for any possible depths. Now, for every depth and every unopened key, the simulator is required to provide commitments in the second signature slot. These commitments contain the nodes in the paths corresponding the PCPP queries for every depth and every unopened key. Recall that, for the simulator to succeed it needs to re-obtain signatures of the commitments in the actual tree for at least one key (and for one depth only).

A naive method to achieve this would be to directly place the commitments used to compute the tree (and for which the simulator already obtained the signatures in the first signature slot), for every tree, and request the signatures. However, as pointed out earlier, this will skew the distribution as these commitments by definition were those on which the verifier gave signatures with high-probability in the first slot and hence might be detectable. To get around this, the simulator first generates random commitments and then inserts it in a manner that will not alter the distribution of the messages sent in the second slot. After the verifier provides the signatures in the second slot and opened the keys in J_2, the simulator finds a key among the remaining unopened ones for which the signatures obtained in the second slot match the ones obtained from the first slot. The cut-and-choose and boosting of the first signature slot will ensure that such a key will exist with high-probability. The simulator commits to the index of this key and the convinces the verifier in the rest of the protocol by creating the MPC-in-the-head views appropriately. The formal description and analysis is provided in Appendix C.2.

Resettable-soundness. We invoke the BGGL transformation to obtain a resettably-sound protocol. More precisely, the verifier in the transformed protocol will generate all the randomness used in the protocol via a PRF applied on the partial transcript. However the proof of soundness does not follow from their work since our verifier utilizes private coins. In [CPS13], they construct a protocol in a hybrid where both the prover and the verifier have access to a Signing Oracle and then show that the Oracle can be implemented via a signature slot. Here we argue the resettable soundness directly. The high-level idea is that using any cheating prover P^* we can construct an adversary A that violates the collision-resistance property of the signatures. A receives as input a verification key and is given access to a signing oracle. Internally, it emulates P^* and supplies messages for the verifier. In a random session opened by P^*, it injects the verification key as one of the n keys. Since it needs to provide a commitment of the randomness used to generate it, A will fake it with a commitment to the 0 string. As long as P^* does not request to open that key via J_1 or J_2, A is safe (unless P^* is breaking the hiding of the commitment). For a randomly chosen session fixed till message P_3, A obtains responses to two independently chosen PCPP queries in V_3. Since P^* cannot obtain a fake witness or cheat in the MPC-in-the-head protocol, it will follow that there must be a collision to the signatures in the paths opened by the two queries. This collision is output by A.

3 Acknowledgments

We thank the anonymous FOCS's reviewers for pointing out an issue with using digital signatures based on one-way functions in a previous version of our work. We thank Kai-Min Chung, Vipul Goyal, Huijia (Rachel) Lin, Rafael Pass and Ivan Visconti for valuable discussions.

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.

[BGGL01] Boaz Barak, Oded Goldreich, Shafi Goldwasser, and Yehuda Lindell. Resettably-sound zero-knowledge and its applications. In *FOCS'01*, pages 116–125, 2001.

[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.

[BP13] Nir Bitansky and Omer Paneth. On the impossibility of approximate obfuscation and applications to resettable cryptography. In *STOC*, pages 241–250, 2013.

[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.

[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.

[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.

[DSMRV13] Dana Dachman-Soled, Tal Malkin, Mariana Raykova, and Muthuramakrishnan Venkitasubramaniam. Adaptive and concurrent secure computation from new adaptive, non-malleable commitments. In *ASIACRYPT (1)*, pages 316–336, 2013.

[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.

[GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *J. Comput. Syst. Sci.*, 28(2):270–299, 1984.

[GMR85] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In *STOC*, pages 291–304, 1985.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989.

[Gol01] Oded Goldreich. *Foundations of Cryptography — Basic Tools*. Cambridge University Press, 2001.

[GOSV14a] Vipul Goyal, Rafail Ostrovsky, Alessandra Scafuro, and Ivan Visconti. Black-box non-black-box zero knowledge. In *STOC*, 2014.

[GOSV14b] Vipul Goyal, Rafail Ostrovsky, Alessandra Scafuro, and Ivan Visconti. Black-box non-black-box zero knowledge. *IACR Cryptology ePrint Archive*, 2014:390, 2014.

[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.

[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.

[IR88] Russell Impagliazzo and Steven Rudich. Limits on the provable consequences of one-way permutations. In *CRYPTO '88*, pages 8–26, 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.

[LP12] Huijia Lin and Rafael Pass. Black-box constructions of composable protocols without set-up. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO*, volume 7417 of *Lecture Notes in Computer Science*, pages 461–478. Springer, 2012.

[Mer89] Ralph C. Merkle. A certified digital signature. In Gilles Brassard, editor, *CRYPTO*, volume 435 of *Lecture Notes in Computer Science*, pages 218–238. Springer, 1989.

[NY89] Moni Naor and Moti Yung. Universal one-way hash functions and their cryptographic applications. In *STOC '89*, pages 33–43, 1989.

[PR05] Rafael Pass and Alon Rosen. New and improved constructions of non-malleable cryptographic protocols. In *STOC '05*, pages 533–542, 2005.

[PTW09] Rafael Pass, Wei-Lung Dustin Tseng, and Douglas Wikström. On the composition of public-coin zero-knowledge protocols. In *CRYPTO '09*, pages 160–176, 2009.

[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.

[RTV04] Omer Reingold, Luca Trevisan, and Salil P. Vadhan. Notions of reducibility between cryptographic primitives. In *TCC*, pages 1–20, 2004.

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

A Definitions

A.1 Computational Indistinguishability

The following definition of computational indistinguishability originates in the seminal paper of Goldwasser and Micali [GM84]. Let X be a countable set of strings. A **probability ensemble indexed by** X is a sequence of random variables indexed by X. Namely, any element of $A = \{A_x\}_{x\in X}$ is a random variable indexed by X.

Definition 1 (Indistinguishability)**.** *Let X and Y be countable sets. Two ensembles $\{A_{x,y}\}_{x\in X,y\in Y}$ and $\{B_{x,y}\}_{x\in X,y\in Y}$ are said to be **computationally indistinguishable over** X, if for every probabilistic machine D (the distinguisher) whose running time is polynomial in its first input, there exists a negligible function $\nu(\cdot)$ so that for every $x \in X, y \in Y$:*

$$|\Pr[D(x, y, A_{x,y}) = 1] - \Pr[D(x, y, B_{x,y}) = 1]| < \nu(|x|)$$

(In the above expression, D is simply given a sample from $A_{x,y}$ and $B_{x,y}$, respectively.)

A.2 Interactive Arguments

Definition 2 (Interactive Arguments)**.** *A pair of interactive algorithms (P, V) is an **interactive argument** for a* **NP** *language L with witness relation R_L if it satisfies the following properties:*

- *Completeness: There exists a negligible function $\mu(\cdot)$, such that for all $x \in L$, if $w \in R_L(x)$,*

$$\Pr[(P(w), V)(x) = 1] = 1 - \mu(|x|)$$

- *Soundness: For all non-uniform polynomial-time adversarial prover P^*, there exists a negligible function $\mu(\cdot)$ such that for every all $x \notin L$,*

$$\Pr[(P, V)(x) = 1] \leq \mu(|x|)$$

*If the following condition holds, (P, V) is an **argument of knowledge**:*

- *Argument of knowledge: There exists an expected* PPT *algorithm E such that for every polynomial-size P^*, there exists a negligble function $\mu(\cdot)$ such that for every x,*

$$\Pr[w \leftarrow E^{P^*(x)}(x); w \in R_L(x)] \geq \Pr[(P^*, V)(x) = 1] - \mu(|x|)$$

A.3 Instance Dependent Equivocal Commitments

Let Com be a statistically binding commitment scheme. Such a commitment scheme can be constructed from OWF.

Let $\mathcal{L}$ be an **NP**-Language and $\mathbf{R}_L$, the associated **NP**-relation. Since the language $\mathcal{L} \in$ **NP**, we can reduce $\mathcal{L}$ to the **NP**-complete problem Hamiltonian Cycle. Thus, given the public input x (which may or may not be in $\mathcal{L}$), we can use a (deterministic) Karp reduction to a graph G which contains a Hamiltonian cycle iff $x \in \mathcal{L}$. Moreover, finding a Hamiltonian cycle H in the graph G, implies finding a trapdoor w such that $\mathbf{R}_L(x, w) = 1$. Let Φ denote the deterministic mapping from strings x to a graphs G induced by the Karp reduction.

The protocol is specified in Figures 3, 4 and has appeared before in [CLOS02]. For completeness, we present it again here and show that it satisfies the properties of an equivocal commitment scheme.

Commitment

$\mathsf{Com}^{\mathsf{x}}(\beta, \tau)$ is defined as:

To commit to $\beta = 1$**:** Choose an $n \times n$ adjacency matrix H of a random n-node Hamiltonian cycle. Commit to each bit of the adjacency matrix H using ExCom.

To commit to $\beta = 0$**:** Choose an $n \times n$ adjacency matrix I which corresponds to a random isomorphism of $G = \Phi(x)$. Commit to each bit of the adjacency matrix I using ExCom.

Decommitment:

To decommit to a 0**:** Open the commitments in M where $M_{i,j}$ is a commitment to 1 and shows that these correspond to a random Hamiltonian cycle.

To decommit to a 1**:** Open the commitments in M to obtain adjacency matrix I and shows an isomorphism from $G = \Phi(x)$ to this graph.

Figure 3: Instance-dependent equivocal commitment scheme.

A.4 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 1 (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.*

Let $\mathsf{Com} = (C, R)$ any commitment scheme with non-interactive opening, which uses one-way-function in a black-box manner. One can construct an extractable commitment scheme $\mathsf{ExCom} = (\mathsf{ExCom}, \mathsf{ExRec})$ as shown in Fig. 5.

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

Equivocal Commitment

Define $\mathsf{EQCom}^{\mathrm{x}}(0^v) = \mathsf{Com}^{\mathrm{x}}(0)$

Adaptive Opening

$\mathsf{Adapt}(x, w, c, v)$, where $c = \mathsf{EQCom}^{\mathrm{x}}(0^v)$

Decommit $v = 0$ as follows: Open all the commitments in the matrix M to reveal adjacency matrix I and shows an isomorphism from $G = \Phi(x)$ to this graph.

Decommit $v = 1$ as follows: Use w to open the commitments in the matrix M that correspond to the Hamiltonian cycle in $G = \Phi(x)$ and shows that these correspond to a random Hamiltonian cycle.

Figure 4: Equivocation.

Extractable Commitment

Commitment Phase:

1. (**Commitment**) 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 $\{c_i^0 = \langle C(r_i^0, \omega_i^0), R\rangle, c_i^1 = \langle C(r_i^1, \omega_i^1), R\rangle\}_{i\in[k]}$.
2. (**Challenge**) ExRec sends a random k-bit challenge string $r' = (r'_1, \ldots, r'_k)$.
3. (**Response**) ExCom decommits $\{c_i^{r'_i}\}_{i\in[k]}$ (i.e., non-interactively opens k of previous commitments, one per pair) 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.

Figure 5: Extractable Commitment.

receiver. The running time of the extractor is polynomial (as can be argued following arguments similar to [PW09])

Remark 1. *In the extractable commitment used in our protocol, we instantiate the underlying commitment scheme* Com *with* $\mathsf{EQCom}^{\mathrm{x}}$*, the instance-dependent Equivocal Commitment shown in Fig. 3.*

A.5 Zero Knowledge

We start by recalling the definition of zero knowledge from [GMR89].

Definition 3 (Zero-knowledge [GMR89])**.** *An interactive protocol (P, V) for language L is **zero-knowledge** if for every PPT adversarial verifier V^*, there exists a PPT simulator S such that the following ensembles are computationally indistinguishable over $x \in L$:*

$$\{\mathsf{View}_{V^*}\langle P, V^*(z)\rangle(x)\}_{x\in L, z\in\{0,1\}^*} \approx \{S(x,z)\}_{x\in L, z\in\{0,1\}^*}$$

A.6 Resettably Sound Zero Knowledge

Let us recall the definition of resettable soundness due to [BGGL01].

Definition 4 (Resettably-sound Arguments [BGGL01])**.** *A resetting attack of a cheating prover P^* on a resettable verifier V is defined by the following two-step random process, indexed by a security parameter n.*

1. *Uniformly select and fix $t = poly(n)$ random-tapes, denoted $r_1, \ldots, r_t$, for V, resulting in deterministic strategies $V^{(j)}(x) = V_{x,r_j}$ defined by $V_{x,r_j}(\alpha) = V(x, r_j, \alpha)$,[7] where $x \in \{0,1\}^n$ and $j \in [t]$. Each $V^{(j)}(x)$ is called an incarnation of V .*

2. *On input 1^n, machine P^* is allowed to initiate $poly(n)$-many interactions with the $V^{(j)}(x)$'s. The activity of P^* proceeds in rounds. In each round P^* chooses $x \in \{0,1\}^n$ and $j \in [t]$, thus defining $V^{(j)}(x)$, and conducts a complete session with it.*

Let (P, V) be an interactive argument for a language L. We say that (P, V) is a resettably-sound argument *for L if the following condition holds:*

- Resettable-soundness*: For every polynomial-size resetting attack, the probability that in some session the corresponding $V^{(j)}(x)$ has accepted and $x \notin L$ is negligible.*

Just as in [CPS13, COP+14] we will consider the following weaker notion of resettable soundness, where the statement to be proven is fixed, and the verifier uses a single random tape (that is, the prover cannot start many independent instances of the verifier).

Definition 5 (Fixed-input Resettably-sound Arguments [PTW09])**.** *An interactive argument (P, V) for a* **NP** *language L with witness relation R_L is **fixed-input resettably-sound** if it satisfies the following property: For all non-uniform polynomial-time adversarial prover P^*, there exists a negligible function $\mu(\cdot)$ such that for every all $x \notin L$,*

$$\Pr[R \leftarrow \{0,1\}^\infty; (P^{*V_R(x,\mathsf{pp})}, V_R)(x) = 1] \leq \mu(|x|)$$

This is sufficient because it was shown in [CPS13] that any zero-knowledge *argument of knowledge* satisfying the weaker notion can be transformed into one that satisfies the stronger one, while preserving zero-knowledge (or any other secrecy property against malicious verifiers.

Claim 1. *Let (P, V) be a fixed-input resettably sound zero-knowledge (resp. witness indistinguishable) argument of knowledge for a language $L \in$ **NP** . Then there exists a protocol (P', V') that is a (full-fledged) resettably-sound zero-knowledge (resp. witness indistinguishable) argument of knowledge for L.*

[7] Here, $V(x, r, \alpha)$ denotes the message sent by the strategy V on common input x, random-tape r, after seeing the message-sequence α.

A.7 Strong Deterministic Signature

In this section, we define strong, fixed-length, deterministic secure signature schemes that we rely on in our construction. Recall that in a strong signature scheme, no polynomial-time attacker having oracle access to a signing oracle can produce a valid message-signature pair, unless it has received this pair from the signing oracle. The signature scheme being fixed-length means that signatures of arbitrary (polynomial-length) messages are of some fixed polynomial length. Deterministic signatures don't use any randomness in the signing process once the signing key has been chosen. In particular, once a signing key has been chosen, a message m will always be signed the same way.

Definition 6 (Strong Signatures)**.** *A* strong, length-ℓ, signature scheme SIG *is a triple* $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ *of* PPT *algorithms, such that*

1. *for all* $n \in \mathcal{N}, m \in \{0,1\}^*$,

$$\Pr[(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{Gen}(1^n), \sigma \leftarrow \mathsf{Sign}_{\mathsf{sk}}(m); \mathsf{Ver}_{\mathsf{vk}}(m, \sigma) = 1 \wedge |\sigma| = \ell(n)] = 1$$

2. *for every non-uniform* PPT *adversary* A*, there exists a negligible function* $\mu(\cdot)$ *such that*

$$\Pr[(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{Gen}(1^n), (m, \sigma) \leftarrow A^{\mathsf{Sign}_{\mathsf{sk}}(\cdot)}(1^n); \mathsf{Ver}_{\mathsf{vk}}(m, \sigma) = 1 \wedge (m, \sigma) \notin L] \leq \mu(n),$$

where L *denotes the list of query-answer pairs of* A*'s queries to its oracle.*

Strong, length-ℓ, **deterministic** signature schemes with $\ell(n) = n$ are known based on the existence of OWFs; see [NY89, Rom90, Gol01] for further details. In the rest of this paper, whenever we refer to signature schemes, we always means strong, length-n signature schemes.

Let us first note that signatures satisfy a "collision-resistance" property.

Claim 2. *Let* $\mathsf{SIG} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ *be a strong (length-n) signature scheme. Then, for all non-uniform PPT adversaries* A*, there exists a negligible function* $\mu(\cdot)$ *such that for every* $n \in \mathcal{N}$*,*

$$Pr[(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{Gen}(1^n), (m_1, m_2, \sigma) \leftarrow A^{\mathsf{Sign}_{\mathsf{sk}}(\cdot)}(1^n, \mathsf{vk}); \mathsf{Ver}_{\mathsf{vk}}(m_1, \sigma) = \mathsf{Ver}_{\mathsf{vk}}(m_2, \sigma) = 1] \leq \mu(n)$$

A.8 Secure Multiparty Computation.

A secure multi-party computation (MPC) [BOGW88, AL11] scheme allows n players to jointly and correctly compute an n-ary function based on their private inputs, even in the presence of t corrupted players. More precisely, let n be the number of players and t denotes the number of corrupted players. Under the assumption that there exists a synchronous network over secure point-to-point channels, [BOGW88] shows that for every n-ary function $\mathcal{F} : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$, there exists a t-secure MPC protocol Π_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 Π_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 Π_f other than the output.

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 $IDEAL_{f,\mathsf{SimMpc}(z),I}(\bar{x})$ the analogous output of SimMpc and honest parties after an ideal execution with a trusted party computing $\mathcal{F}$.

Definition 2. *Let $\mathcal{F} : (\{0,1\}*)^n \to (\{0,1\}*)^n$ be an n-ary functionality and let Π be a protocol. We say that Π (n,t)-perfectly securely computes $\mathcal{F}$ if for every probabilistic adversary A in the real model, there exists a probabilistic adversary S of comparable complexity[8] 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 $\mathcal{F}$ 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 $\mathcal{F}$ in the presence of a static malicious adversary for any $t < n/3$.*

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 3 (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.*

A.9 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 $\texttt{Recon}(\mathsf{S}_i, \ldots, \mathsf{S}_n)$ 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].

Definition 4 (VSS Scheme)**.** *An $(n+1,t)$-perfectly secure VSS scheme consists of a pair of protocols $\texttt{VSS} = \langle \texttt{Share}, \texttt{Recon} \rangle$ that implement respectively the sharing and reconstruction phases as follows.*

$\texttt{Share}(s)$*. 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, $1 \le i \le n$, runs on input a randomness r_i. During this phase players can send (both private and broadcast) messages in multiple rounds.*

$\texttt{Recon}(\mathsf{S}_1, \ldots, \mathsf{S}_n)$*. 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.*

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

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`.

Implementations of $(n+1, \lfloor n/3 \rfloor)$-perfectly secure VSS schemes can be found in [BOGW88, CDD+99]. We are interested in a deterministic reconstruction procedure, therefore we adopt the scheme of [GIKR01] that implements an $(n+1, \lfloor n/4 \rfloor)$-perfectly secure VSS scheme.

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 MPCZK 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 MPCZK 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. 3) 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 an input that is a view of a VSS player S_i. $\mathcal{F}_\phi$ collects all such views, and outputs 1 if and only if $\phi(\mathtt{Recon}(\mathsf{S}_1, \ldots \mathsf{S}_n)) = 1$. This idea is crucially used in [GOSV14a].

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 $x \neq x' \in \{0,1\}^k, |\{i | \mathsf{ECC}(x)_i \neq \mathsf{ECC}(x')_i\}| \geq \delta\ell(k)$.

A.10 Probabilistically Checkable Proof (PCP)

The following definition is taken verbatim from [BG08]. A probabilistically checkable proof (PCP) system consists of a probabilistic polynomial-time verifier having access to an oracle which represents a proof in redundant form. Typically, the verifier accesses only a few of the oracle bits, and these bit positions are determined by the outcome of the verifier's coin tosses. It is required that if the assertion holds, then the verifier always accepts (i.e., when given access to an adequate oracle); whereas if the assertion is false, then the verifier must reject with high probability (as specified in an adequate bound), no matter which oracle is used.

Definition 5 (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 6 (PCP – auxiliary properties).)**.** *Let V be a PCP verifier with error $\epsilon : \mathcal{N} \to [0,1]$ 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 $?_x$ 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.10.1 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.

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 $y[q_i]$, $\pi[p_i]$.

Definition 7 (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 8 (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 9 (PCPP verifier for a pair language)**.** *For functions $s, \delta : \mathcal{N} \to [0,1]$, 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 3 ((Theorem 1 of [DSK12]))**.** *Let $T : \mathcal{N} \to \mathcal{N}$ 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 the PCP of Proximity satisfies the **efficient-resampling** property [BSCGT12].

B Details of our resettable-sound ZK protocol

In this section we describe our protocol in details.

We start with setting up some useful notation.

Notation for VSS views. We use S to denote a view of a VSS player, that we call share. The notation $\mathsf{S}^{\gamma}_{\mathsf{string}}[i]$ refers to the share of the i-th player, participating in the VSS for the *label* of node in position γ, in the Merkle tree committing the string string. For instance: $\mathsf{S}^{\gamma}_{M}[i]$ denotes the view of the VSS player P_i sharing the label of node γ, of the tree hiding the string M.

Notation for MPC-in-the-head views. We use $P^{\gamma}_{\mathsf{prot}}[i]$ to denote the view of player P_i participating in the MPC protocol prot, executed to prove some property of the node γ. For example, $P^{\gamma}_{\mathsf{in}}[i]$ denotes the view of player P_i, participating in protocol $\mathsf{MPC\text{-}Innode}$ (that we describe in the next section), which is executed on top of the VSS of node γ.

Queries of the PCPP verifier. Size-hiding demands that V has no information on the actual size of the theorem/proof committed by P. Consequently, V must compute a the PCPP queries for each possible size of the proof. Assuming that the size of the PCPP proof is a power of 2, we have that there are at most $\log^2 n$ possible PCPP proofs. Therefore, for each lenght $j = 1, \ldots, \log^2 n$, V sends queries $\{q_{i,j}, p_{i,j}\}_{i \in k}$, where k is the soundness parameter of the PCPP.

Some Parameters. We let $\ell_d = \log^2 n$. For every query of the PCPP verifier, the prover P must exhibit a path, i.e., a sequence of nodes. We shall denote by N the total number of *encode* parts that P prepares in Step P_4 (see Protocol described in Sec. 2). As the number of queries is $2k$ for each PCPP proof, and as there are ℓ_d possible "proofs", $N = \sum_{j=1}^{j=\ell_d} \sum_{i=1}^{i=k} \log q_{i,j} \cdot \log p_{i,j}$.

Value representation. Any value v involved in the proof (e.g., a node of the tree, the depth of the real tree) is represented as a vector of views of VSS players. Specifically let v be any string that P will use in the proof. Instead of working directly with v, P will execute (in his head) a VSS protocol among $n+1$ players in which the dealer shares the string v. 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 v, but any subset of only t views does not reveal anything about v. This vector of views is the "VSS representation" of v.

Definition 10 (VSS representation)**.** *The VSS representation of a string $s \in \{0,1\}^*$, is the vector $[\mathsf{S}[1], \ldots, \mathsf{S}[n]]$, where $\mathsf{S}[i]$ is the view of player i, participating in the protocol $\mathtt{Share}(s)$.*

Definition 11 (innode property of a node)**.** *A node in position γ is the pair $\left[label^{\gamma}, \mathsf{Com}(\mathsf{S}^{\gamma}_{label}[1]), \ldots, \mathsf{Com}(\mathsf{S}^{\gamma}_{label}[n])\right]$. The innode property of γ is satisfied iff $\mathtt{Recon}(\mathsf{S}^{\gamma}_{label}[1], \ldots, \mathsf{S}^{\gamma}_{label}[n]) = label^{\gamma}$.*

Definition 12 (Node Connection)**.** *Let $\mathsf{VSS}^{\gamma 1} = \{\mathsf{S}^{\gamma 0}[i]\}_{i \in [n]}$ and $\mathsf{VSS}^{\gamma 1} = \{\mathsf{S}^{\gamma 1}[i]\}_{i \in [n]}$ then the label for node γ is the vector: $label^{\gamma} = [\mathsf{Sign}(\mathsf{Com}(\mathsf{S}^{\gamma 0}[1])) \mid \cdots \mid \mathsf{Sign}(\mathsf{Com}(\mathsf{S}^{\gamma 0}[n]) \mid \mathsf{Sign}(\mathsf{Com}(\mathsf{S}^{\gamma 1}[1])) \mid \cdots \mid \mathsf{Sign}(\mathsf{Com}(\mathsf{S}^{\gamma 1}[n]))]$. Each VSS view is* signed separately.

PCP of Proximity. We use PCP of Proximity for the following pair language:
$\mathcal{L}_{\mathcal{P}} = \{(a = (r, \mathtt{t}), (Y)), \exists M \in \{0,1\}^* \text{ s.t. } Y \leftarrow \mathsf{ECC}(M), M(z) \rightarrow r \text{ within } \mathtt{t} \text{ steps}\}$
(where $\mathsf{ECC}(\cdot)$ is a binary error correcting code tolerating a constant fraction $\delta > 0$ of errors). Where a is the *public* theorem, and will be known to both P and V by the end of the trapdoor-generation phase. In our case $a = (r, \mathtt{t})$.

B.1 MPC-in-the-head used

We now define more in details the MPC-in-the-head that are computed by the Simulator/Prover.

Functionality $\mathcal{F}_{\mathsf{innode}}$.
Public input: Index of the node: γ; index of the path: j; the label part of node γ: $label^{\gamma}$, the index of the tree in examination T.
Secret Inputs: VSS shares: $\mathsf{S}^{\gamma}_{\mathsf{string}}[i], \mathsf{S}_{\mathsf{depth}}[i], \mathsf{S}_{\mathsf{tree}}[i]$, received from player P_i with $i \in [n]$.

Functionality:

1. run $d = \mathtt{Recon}(\mathsf{S}_{\mathsf{depth}}[1], \ldots, \mathsf{S}_{\mathsf{depth}}[n])$. If the reconstruction fails output 0 to all players and halt. If $d \notin \{1, \ldots, \log^2 n\}$ output 0 and halt. Else, if $j \neq d$ output 1 and halt.

2. run $J = \mathtt{Recon}(\mathsf{S}_{\mathsf{tree}}[1], \ldots, \mathsf{S}_{\mathsf{tree}}[n])$. If the reconstruction fails output 0 to all players and halt. If $J \notin \{1, \ldots, n\}$ output 0 and halt. Else, if $T \neq J$ output 1 and halt.

3. run $v = \mathtt{Recon}(\mathsf{S}^{\gamma}_{\mathsf{string}}[1], \ldots, \mathsf{S}^{\gamma}_{\mathsf{string}}[n])$. If the reconstruction fails output 0 to all players and halt. If $v = label^{\gamma}$, output 1 to all players and halt. Else, output 0.

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

For every node, P prepares a proof to convince the verifier that any internal node either is "consistent" or that some trapdoor condition holds. For any **leaf** P proves that either the value of the leaf is accepted by the PCPP Verifier, or some trapdoor condition holds.

Namely, P proves to the verifier the following two properties.

- **Node consistency**. For every node γ, over a path of length j for tree *tree* (recall that, because of the cut-and-choose there are $n/4$ trees), P wants to prove that either $j \neq \mathsf{depth}$, where depth is the size of tree, or that the $j = \mathsf{depth}$ *and* the innode property (see Def. 11) for node γ is satisfied, or that $tree \neq J$.

  P proves the consistency of a node running in its head the protocol $\mathsf{MPC\text{-}Innode}$. $\mathsf{MPC\text{-}Innode}$ is a perfectly t-robust and perfectly t-private (n, t)-MPC protocol for the functionality $\mathcal{F}_{\mathsf{innode}}$ depicted in Fig. 6.

- **PCPP Verifier's acceptance**. For every set of queries $(q_{i,j}, p_{i,j})_{i \in [k]}$ for a depth j, P proves that either j is not the real depth of the committed trees, OR the PCPP Verifier accepts the values of the tree in positions $(q_{i,j}, p_{i,j})_{i \in [k]}$, OR $tree \neq J$.

  P proves this property by running in its head protocol $\mathsf{MPC\text{-}PCPVer}$. Protocol $\mathsf{MPC\text{-}PCPVer}$ is a perfectly t-robust, perfectly t-private (n, t)-MPC protocol for the functionality $\mathcal{F}_{\mathsf{VerPCPP}}$ depicted in Fig. 7.

B.2 The Protocol in details

We are now ready to show our protocol in details.

Let $(\mathsf{Com}^{\mathsf{x}}, \mathsf{EQCom}^{\mathsf{x}})$ be an instance-dependent equivocal commitment described in Sec. A.3, defined on the instance $x \in L$. Let $\mathsf{ExCom} = (\mathsf{ExCom}, \mathsf{ExRec})$ be an extractable commitment instantiated with the instance-dependent equivocal commitment $(\mathsf{Com}^{\mathsf{x}}, \mathsf{EQCom}^{\mathsf{x}})$ (as shown in Fig. 5). Let $T \stackrel{def}{=} \text{polylog}n$, let $\ell_d = \log^2 n$, and v be the size of a VSS share. We denote by $(\mathsf{Q}_{\mathsf{pcpx}}, \mathsf{D}_{\mathsf{pcpx}})$ a PCPP verifier (as defined in App. A.10.1), with query complexity k and proximity parameter δ. Let $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ a deterministic strong-signature scheme.

Functionality $\mathcal{F}_{\mathsf{VerPCPP}}$.
Public input: Queries: $q_{i,j}, p_{i,j}$; public theorem: (a, r); the index T of the tree in examination.
Secret inputs: $\mathsf{S}_M^{q_{i,j}}[i]$, $\mathsf{S}_\pi^{p_{i,j}}[i]$, $\mathsf{S}_{d\pi}[i], \mathsf{S}_{\mathsf{tree}}[i]$ with $i \in [n]$.

Functionality:

1. run d= $\mathtt{Recon}(\mathsf{S}_{dM}[1], \ldots, \mathsf{S}_{dM}[n])$. If the reconstruction fails output 0 to all players and halt. Else, if $j \neq d$ output 1 and halt.
2. run $J = \mathtt{Recon}(\mathsf{S}_{\mathsf{tree}}[1], \ldots, \mathsf{S}_{\mathsf{tree}}[n])$. If the reconstruction fails output 0 to all players and halt. Else, if $T \neq J$ output 1 and halt.
3. run $m_{j,i} = \mathtt{Recon}(\mathsf{S}_M^{q_{i,j}}[1], \ldots, \mathsf{S}_M^{q_{i,j}}[n])$ and $\pi_{i,j} = \mathtt{Recon}(\mathsf{S}_\pi^{p_{i,j}}[1], \ldots, \mathsf{S}_\pi^{p_{i,j}}[n])$. If any of the reconstructions fails output 0 to all players and halt. If $\mathsf{D}_{\mathsf{pcpx}}(a, m_i^j, \pi_{i,j}, q_{i,j}, p_{i,j}) = 1$ output 1 to all players. Else output 0.

Figure 7: The $\mathcal{F}_{\mathsf{VerPCPP}}$ functionality.

Common Input: An instance x of a language $L \in \mathbf{NP}$ with witness relation $\mathbf{R}_L$.

Auxiliary input to P: A witness w such that $(x, w) \in \mathbf{R}_L$.

Cut-and-choose 1.

- P_0: Randomly pick two disjoint subsets of $\{1, \ldots, n\}$ that we denote by J_1, J_2, with $|J_1| = n/2$ and $|J_2| = n/4$. Commit to J_1, J_2 using the equivocal commitment scheme $\mathsf{Com}^{\mathrm{x}}$.
- V_0: Run $(\mathsf{sk}_\kappa, \mathsf{vk}_\kappa) \leftarrow \mathsf{Gen}(1^n, r_\kappa)$ for $\kappa = 1, \ldots, n$. Send $\mathsf{vk}_\kappa, \mathsf{Com}(r_\kappa)$ for $\kappa = 1, \ldots, n$ to P.
- **Signature Slot 1**.
  - P_1: Send $\mathsf{T} = O(n^c)$ commitments to the zero-string (for some constant c), that must be signed with each signature key sk_κ with $\kappa \in [n]$. Specifically, for each $\kappa \in [n]$, computes commitments $c_{\kappa,l} \leftarrow \mathsf{Com}^{\mathrm{x}}(0^v)$ with $l \in [\mathsf{T}]$ where v is the size of a VSS share. To simplify the notation we omit the auxiliary information τ computed in all equivocal commitments. Send $c_{\kappa,1}, \ldots, c_{\kappa,\mathsf{T}}$ to V.
  - V_1: For each $\kappa \in [n]$: send $\sigma_{\kappa,1}, \ldots, \sigma_{\kappa,\mathsf{T}}$ to P where $\sigma_{\kappa,l} = \mathsf{Sign}_{\mathsf{sk}_\kappa}(c_{\kappa,l})$ with $l = 1, \ldots, T$. P aborts if any of the signatures is not valid.
- **Check Signature Slot 1**.
  - $P_{1.2}$: Open set J_1.
  - $V_{1.2}$: For $\kappa \in J_1$, send sk_κ and the decommitment to r_κ.
  - $P_{1.3}$: Check that all signatures verified under key vk_κ are consistent with $\mathsf{sk}_\kappa, r_\kappa$. If not, abort.

Commitment to the Machine.

- P_2: Send equivocal commitment to the *encode* part of the root of the (extendable) Merkle

signature-tree for M, and equivocal commitment to the depth of the tree.

Namely, for each tree $\kappa \in [n]/J_1$, P computes equivocal commitments (that we denote by tc meaning, trapdoor commitment) to each VSS view l of the *encode* part of the root:

$$\mathsf{tc}_M[\kappa, l] = \mathsf{EQCom}^{\mathrm{x}}(0^v)$$

and to each VSS view S_l of the depth of the tree as:

$$\mathsf{tc}_{dM}[\kappa, l] == \mathsf{EQCom}^{\mathrm{x}}(0^v)$$

For each tree κ, send $L_M^{\lambda}[\kappa] = \{\mathsf{tc}_M[\kappa, l]\}_{l\in[n]}$ as the root of the tree of the machine, and $\mathsf{tc}_{dM}[\kappa, l]$ as commitment to the depth.

- V_2: Send $r \xleftarrow{\$} \{0,1\}^n$ to P.
  P and V set the public theorem as $a = (r, \mathsf{t})$.

Commitment to the PCPP proof.

- P_3: Send equivocal commitment to the *encode* part of the root of the (extendable) Merkle signature-tree for the PCPP proof and the commitment to the depth of such tree. Namely, for each tree κ, with $\kappa \in [n]/J_1$, as before, computes commitments $\mathsf{tc}_\pi[\kappa, l] = \mathsf{EQCom}^{\mathrm{x}}(0^v)$ and $\mathsf{tc}_{d\pi}[\kappa, l] = \mathsf{EQCom}^{\mathrm{x}}(0^v)$ (depth of the tree) for $l \in [n]$. For each tree κ, let $L_\pi^{\lambda} = \{\mathsf{tc}_\pi[\kappa, l]\}_{l\in[n]}$ be the root of the tree of the PCP proof, and $\mathsf{tc}_{d\pi}[\kappa, l]$ the commitment to the depth.
- V_3: Pick a uniformly chosen random tape r_j for every possible depth j of the PCPP proof. Send $r_1, \ldots, r_{\ell_d}$ to P.

  P and V compute the queries of the PCPP Verifier: $(q_{i,j}, p_{i,j}) = \mathsf{Q}_{\mathsf{pcpx}}(a, r_j, i)$ with $i \in [k]$ (k is the soundness parameter for the PCPP).
- P_4: For each tree $\kappa \in [n]/J_1$;
  for each level j of tree κ;
  let $p_{i,j}, q_{i,j}$ be the PCPP queries for level j; send the *encode* part for each node γ_j, β_j along the paths $p_{i,j}, q_{i,j}$ of tree κ. Namely, for each node γ_j in the κ-th tree of the PCPP proof, P sends trapdoor commitments to n VSS views:

  $$\mathsf{tc}_\pi^{\gamma_j}[\kappa, l] = \mathsf{EQCom}^{\mathrm{x}}(0^v)$$

  Similarly, for each node β_j in the κ-th tree for the machine M, P sends trapdoor commitments to n VSS views

  $$\mathsf{tc}_M^{\beta_j}[\kappa, l] = \mathsf{EQCom}^{\mathrm{x}}(0^v)$$

  Note. For different levels, say levels $j, j+1$, of the *same* tree, the paths opened for queries $(p_{i,j}, q_{i,j})$ and $(p_{i,j+1}, q_{i,j+1})$ are computed independently. Namely, the nodes in $p_{i,j}$ which are also on the path for $p_{i,j+1}$ are not reused but freshly recomputed. Instead, within the *same level* j, the nodes common to two queries, say $p_{1,j}$ and $p_{2,j}$, are reused.

Signature Slot 2 V_4: Sign the *encode* parts received from P.

For each $\kappa \in \{1, \ldots, n\}/J_1$; send $\sigma_\pi^{\gamma_j}[\kappa, l] = \mathsf{Sign}_{\mathsf{sk}_\kappa}(\mathsf{tc}_\pi^{\gamma_j}[\kappa, l])$ (do the same with β_j on the tree $_M$)

Cut-and-choose 2

- $P_{4.1}$: Open the set J_2.
- $V_{4.1}$: Send $\mathsf{sk}_\kappa, r_\kappa$, for $\kappa \in J_2$.
- $P_{1.3}$: Check that all signatures verifier under key vk_κ are consistent with $\mathsf{sk}_\kappa, r_\kappa$. If not, abort.

Proof.

- P_5: Send equivocal commitments to the VSS of a random index $J \in 1, \ldots, n/\{J_1 \cup J_2\}$. Namely, P sends: $\mathsf{tc}_{\mathsf{tree}}[1], \ldots, \mathsf{tc}_{\mathsf{tree}}[n]$ with $\mathsf{tc}_{\mathsf{tree}}[l] = \mathsf{EQCom}^{\mathrm{x}}(0^v)$.
  Compute the proof of consistency of each path, and the proof of acceptance for each set of queries (as explained earlier), for each of the remaining trees. Namely.
  For each tree $\kappa \in [n]/\{J_1 \cup J_2\}$;
  For each level j of tree κ;
  For each node γ_j, β_j on the paths for queries let $p_{i,j}, q_{i,j}$, P prepares equivocal commitments to views of the MPC-in-the-head MPC-Innode:

$$\mathsf{tc}_{\mathsf{in}}^{\gamma_j}[\kappa, l] = \mathsf{EQCom}^{\mathrm{x}}(0^v)$$

  and

$$\mathsf{tc}_{\mathsf{in}}^{\beta_j}[\kappa, l] = \mathsf{ExCom}(0^v)$$

  and send them to V.
  For the leaves nodes in positions $(q_{i,j}, p_{i,j})$, which represent the bits of the machine/PCP queried by V; P sends commitment to the views of the MPC-in-the-head MPC-PCPVer:

$$\mathsf{tc}_{\mathsf{PCPVer}}^{j}[\kappa, l] = \mathsf{ExCom}(0^v)$$

  Where ExCom is an extractable *and* equivocal commitment scheme.
- V_5: Select t players to check: Send indexes $p_1, \ldots, p_t$.
- P_6: At this point the prover knows which are the players (i.e., VSS players, MPC-Innode's players and MPC-PCPVer's players) that the verifier wants to check. It will proceed to compute views that will look correct to the verifier.
  1. **Compute valid VSS shares (but still, shares of the zero-string)** To generate correct VSS views, it will run the honest procedure `Share` on input 0^n. The goal of the prover is just to build correct views that can be given in input to the honest players of the MPC-in-the-head: MPC-Innode and MPC-PCPVer.
     (a) Compute valid shares for the VSS of depths. Namely, for each tree $\kappa \in \{1, \ldots, n\}/\{J_1 \cup J_2\}$ run:
     $\{\mathsf{S}_{d_M}^{\kappa}[1], \ldots, \mathsf{S}_{d_M}^{\kappa}[n]\} \leftarrow \mathtt{Share}(0^n)$ and $\{\mathsf{S}_{d_\pi}^{\kappa}[1], \ldots, \mathsf{S}_{d_\pi}^{\kappa}[n]\} \leftarrow \mathtt{Share}(0^n)$.
     (b) Compute valid shares of 0 for the VSS of J. Namely run:
     $\{\mathsf{S}_{\mathsf{tree}}[1], \ldots, \mathsf{S}_{\mathsf{tree}}[n]\} \leftarrow \mathtt{Share}(0^n)$.
  2. **Equivocate views of MPC-Innode (proof of consistency of each node)** . For each tree κ, for each node γ_j, β_j on the paths for queries $p_{i,j}, q_{i,j}$ do:
     (a) Compute valid shares of 0 for each node γ_j on the tree of the machine (resp. β_j for the tree of M) that was revealed as follows.

$$\{\mathsf{S}_M^{\kappa,\gamma_j}[1], \ldots, \mathsf{S}_M^{\kappa,\gamma_j}[n]\} \leftarrow \mathtt{Share}(0^n); \{\mathsf{S}_\pi^{\kappa,\beta_j}[1], \ldots, \mathsf{S}_\pi^{\kappa,\beta_j}[n]\} \leftarrow \mathtt{Share}(0^n)$$

(b) Equivocate commitments in position p_j so that they open to the shares just generated. Compute

$$\mathsf{Adapt}(x, w, \mathsf{tc}_{\pi}^{\gamma_j}[\kappa, p_j], \mathsf{S}_{\pi}^{\kappa,\gamma_j}[\kappa, p_j])$$

(resp., $\mathsf{Adapt}(x, w, \mathsf{tc}_{M}^{\beta_j}[\kappa, p_j], \mathsf{S}_{M}^{\kappa,\beta_j}[\kappa, p_j])$ for any β_j on the tree of the machine). Also equivocate the commitments to the VSS views for the tree J:

$$\mathsf{Adapt}(x, w, \mathsf{tc}_{\mathsf{tree}}[p_j], \mathsf{S}_{\mathsf{tree}}[p_j])$$

and for the VSS views for the depths of the real trees:

$$\mathsf{Adapt}(x, w, \mathsf{tc}_{d\pi}[p_j], \mathsf{S}_{d\pi}^{\kappa})$$

$$\mathsf{Adapt}(x, w, \mathsf{tc}_{dM}[p_j], \mathsf{S}_{dM}^{\kappa})$$

(c) Compute accepting views for players $p_1, \ldots, p_t$ by using the Semi-honest MPC Simulator SimMpc associated to the MPC protocol in. For every γ, β along the paths for queries $q_{i,j}, p_{i,j}$: Set

$$P_{\mathsf{in}}^{\kappa,\gamma_j}[p_1], \ldots, P_{\mathsf{in}}^{\kappa,\gamma_j}[p_t] = \mathsf{SimMpc}(p_1, \ldots, p_t, \gamma_j, q_{i,j}, \mathsf{S}_{d\pi}^{\kappa}[p_j], \mathsf{S}_{\pi}^{\kappa,\gamma_j}, \mathsf{S}_{\mathsf{tree}}[p], \sigma_{\pi}^{\gamma_j}[p], \text{out1})$$

recall that $\sigma_{\pi}^{\gamma_j}$ is the *label* part of the node γ_j, and is a public input of the functionality $\mathcal{F}_{\mathsf{innode}}$.
The same is done for the node β_j.

(d) Equivocate the commitments for the MPC views in position p_j so that they open to the above generated views as follows:

$$\mathsf{Adapt}(x, w, \mathsf{tc}_{\mathsf{in}}^{\gamma_j}[\kappa, p_j], P_{\mathsf{in}}^{\kappa,\gamma_j}[p_j])$$

with $p_j \in [t]$ (resp., for node β_j).

3. Equivocate views for $\mathsf{MPC\text{-}PCPVer}$ on public input $a = (r, \mathsf{t})$. For each set of queries $(q_{1,j}, p_{1,j}, \ldots, q_{k,1}, p_{k,j})$ asked for level j, compute the views for $\mathsf{MPC\text{-}PCPVer}$'s players as follows:

$$P_{\mathsf{PCPVer}}^{\kappa,j}[p_j] = \mathsf{SimMpc}(p_j, \mathsf{S}_{M}^{q_{i,j}}[p_j], \mathsf{S}_{\pi}^{p_{i,j}}[p_j], \mathsf{S}_{d\pi}^{\kappa}[p_j], \mathsf{S}_{dM}^{\kappa}[p_j], \mathsf{S}_{\mathsf{tree}}[p_j], a, \text{out1})$$

Equivocate commitments for views of $\mathsf{MPC\text{-}PCPVer}$ in position p by running: $\mathsf{Adapt}(x, w, \mathsf{tc}_{\mathsf{PCPVer}}^{j}[\kappa, p_j],\ P_{\mathsf{PCPVer}}^{\kappa,j}[p_j])$ with $p_j \in [t]$.

Verification. V_6: The verifier obtains the opening to all the views of players in positions $p \in p_1, \ldots, p_t$. Namely the views for all the VSS and the MPC-in-the-head that the prover has computed for players $p_1, \ldots, p_t$. Now, V has to check the views are consistent (as per Def. 3) and the VSS views corresponds to the input of the MPC players involved in the consistency proofs. More formally, V performs the following checks:

1. **VSS consistency.** For every node γ_j, β_j, check that the views $\mathsf{S}_{\pi}^{\kappa,\gamma_j}[p_1], \ldots, \mathsf{S}_{\pi}^{\kappa,\gamma_j}[p_t]$ are consistent with each other.

   Also check that VSS views $\mathsf{S}_{d\pi}^{\kappa}[p_1], \ldots, \mathsf{S}_{d\pi}^{\kappa}[p_t]$ are consistent with each other. Finally, check that $\mathsf{S}_{\mathsf{tree}}[p_1], \ldots, \mathsf{S}_{\mathsf{tree}}[p_t]$ are also consistent.

2. **Node consistency.** For every tree κ; for every level j;
Then for each player p in $p_1, \ldots, p_t$ check that the view of the MPC player for $\mathsf{MPC\text{-}Innode}$: $P_{\mathsf{in}}^{\kappa,\gamma_j}[p]$ has in the **input** the views $\mathsf{S}_{\pi}^{\kappa,\gamma_j}[p_1]$, $\mathsf{S}_{d\pi}^{\kappa}[p]$, $\mathsf{S}_{\mathsf{tree}}[p]$, and has as **output** 1, and that all views are **consistent** with each other.
3. **PCPP verifier's acceptance.** For every set of nodes $(q_{i,j}, p_i^j)_{i\in[k]}$, check that the view $P_{\mathsf{PCPVer}}^{j}[p]$ has in **input** $\mathsf{S}_{M}^{\kappa,q_{i,j}}[p]$, $\mathsf{S}_{\pi}^{\kappa,p_{i,j}}[p], \mathsf{S}_{d\pi}[p], \mathsf{S}_{\mathsf{tree}}[p]$ and has in **output** is 1, and that all the opened views $P_{\mathsf{PCPVer}}^{j}[p_1], \ldots, P_{\mathsf{PCPVer}}^{j}[p_t]$ are consistent with each other.

C Security Proof

In this section we provide the formal proof of security of the following theorem.

Theorem 4. *There exists a (semi) black-box construction of a resettably-sound zero-knowledge argument of knowledge based on one-way functions.*

Completeness follows from the perfect simulatability of the semi-honest parties in the MPC-in-the-head protocols and equivocability of the equivocal commitment scheme.

C.1 Resettable Soundness and Argument of Knowledge

Proof Sketch of resettable soundness: Analogous to [CPS13], it suffices to prove *fixed-input* resettable-soundness of the protocol without loss of generality. Assume for contradiction, there exists a PPT adversary P^*, sequences $\{x_n\}_{n\in\mathcal{N}} \subseteq \{0,1\}^*/L$, $\{z_n\}_{n\in\mathcal{N}} \subseteq \{0,1\}^*$ and polynomial $p(\cdot)$ such that for infinitely many n, it holds that P^* convinces V on common input $(1^n, x_n)$ and private input z_n with probability at least $\frac{1}{p(n)}$. Fix an n, for which this happens.

First, we consider a hybrid experiment HYB, where we run the adversary P^* on input (x_n, z_n) by supplying the messages of the honest verifier with a small modification. For all the randomness used by the verifier in the protocol via a PRF applied on the transcript, we instead supply truly random strings. In particular, the signature keys generated in the first message, the challenge string in message V_2, the random strings in V_3 and random t indices in V_5 are sampled truly randomly. By the pseudo-randomness of the PRF, we can conclude that P^* convinces the emulated verifier in this hybrid with probability at least $\frac{1}{p(n)} - \nu(n) > \frac{1}{2p(n)}$ for some negligible function $\nu(\cdot)$.

The high-level idea is that using P^*, we construct an oracle adversary A that violates the collision-resistance property of the underlying signature scheme. In more details, $A^{\bullet}$ is an oracle-aided PPT machine that on input vk, n and oracle access to a signing oracle $\mathsf{SIG}_{\mathsf{sk}}(\cdot)$ proceeds as follows: It internally incorporates the code of P^* and begins emulating the hybrid experiment HYB by providing the verifier messages. Recall that P^* is a resetting prover and can open arbitrary number of sessions by rewinding the verifier. A selects a random session i.

- For all the unselected sessions, A simply emulates the honest verifier.
- For the selected session, A proceeds as follows.

  (1) In the first message of the protocol, $A^{\mathsf{SIG}_{\mathsf{sk}}(\cdot)}$ chooses a random index $f \in [n]$ and places the vk in that coordinate. More precisely, it sets $(\mathsf{vk}_f, \sigma_f) = (\mathsf{vk}, c)$ where c is a commitment to the 0 string using Com. We remark here that since A does not have the secret key or the randomness used to generate $(\mathsf{sk}_f, \mathsf{vk}_f)$, it commits to the 0 string as randomness. It then emulates the protocol honestly. In either of the signature slots, whenever A needs to provide a signature under sk_f for any message m, A queries its oracle on m. For all

the other keys, it possesses the signing key and can generate signatures on its own. If the sets J_1 or J_2 revealed in $P_{1.2}$ and $P_{4.2}$ contains f, then A simply halts.

(2) If P^* fails to convince the verifier in the selected session, A halts. If it succeeds, then A stalls the emulation. Let C_1 contain the messages exchanged in session i. A then rewinds the prover to the message V_3. Let τ be the partial transcript of the messages exchanged until message V_3 in session i occurs.

(3) Next, A uses fresh randomness to generate V_3 continues the execution from τ until P^* either convinces the verifier in that session or aborts. If it aborts then A halts.

(4) Otherwise, using the values revealed by P^* in the two continuations from the point τ, A will try to extract a collision if one exists and halts otherwise. Let C_2 denote the messages exchanged in session i in the rewinding.

We describe below how the adversary A obtains a collision from two convincing transcripts starting from τ and argue that it can do so with non-negligible probability. Thus, we arrive at a contradiction to collision-resistance property of the signature scheme and will conclude the proof of resettable-soundness.

First, we consider a hybrid experiment HYB', where a hybrid adversary A is provided with the actual commitment c to the randomness used to generate the signing key whose signing oracle it has access to. Besides that A' proceeds identically to A. By construction the internal emulation by A in HYB' is identical to that of HYB. Below we analyze A's success in hybrid experiment HYB'. Finally, we claim that A will succeed with probability close to hybrid experiment HYB' because the commitment scheme is hiding.

In a convincing session, for every PCPP query and every unopened signature key f, there is an associated set of paths that the prover reveals. More precisely, for every node γ in the paths, it provides $label^{\gamma}, \{\mathsf{Com}(\mathsf{S}^{\gamma}[i])\}_{i\in[n]}$ where $label^{\gamma}$ is supposed to be the label associated with the node and the commitments are in the *encode* part of the node. The nodes corresponding to paths revealed for every PCPP query is presented in the signature slot. If a node γ is supposed to be the child of the node γ', then it must be the case that $label^{\gamma'}$ contains the valid signatures of $\{\mathsf{Com}(\mathsf{S}^{\gamma}[i])\}_{i\in[n]}$.

Suppose that, in two convincing continuations from τ, for some pair of parent and child node γ' and γ, $label^{\gamma'}$ associated with γ' is the same in both the continuations but the commitments in the *encode* part of γ are different. This means that there is at least one signature in the $label^{\gamma'}$ that verified correctly on two different values of $encode^{\gamma}$. Therefore, if A finds one such pair of nodes for the key sk_f, then it obtains a collision. We show below that with non-negligible probability, A will obtain a collision in this manner.

Towards this, we first show that with non-negligible probability A will obtain two random continuations from τ where P^* succeeds in convincing the verifier in session i. Recall that in HYB', the probability of P^* convincing the verifier in some session is at least $\frac{1}{2p(n)}$. By an averaging argument, we know that, with probability at least $\frac{1}{4p(n)}$ over partial transcripts until message V_3 in some session i', P^* convinces V^* on a random continuation from that message in the same session with probability at least $\frac{1}{4p(n)}$. P^* runs in polynomial time and can open at most polynomially many sessions $m(n)$. Therefore, the probability that A picks i to be that session is at least $\frac{1}{p_1(n)} = \frac{1}{m(n)}$. Next, we estimate the probability that A halts because $f \in J_1 \cup J_2$. J_1 is revealed once and J_2 twice (one for each continuation from τ). The probability that f is in any of these sets is $\frac{1}{2}$ because the location f is statistically hidden.[9] So, A halts because of this with probability at most $\frac{1}{8}$. Therefore, the overall probability that A reaches τ from which random continuations are convincing with probability at least $\frac{1}{4p(n)}$ is at least $\frac{1}{p_2(n)} = \frac{1}{p_1(n)} \times \frac{1}{8} \times \frac{1}{4p(n)}$ which is non-negligible.

[9]This is true even if the prover equivocates and gives two different J_2's.

Next, we analyze the set of bad events when A receives two convincing continuations from τ. These bad events prevent A from finding a collision. So we bound the probabilities of these events happening. Fix a τ for which a random continuation yields a convincing session i with probability at least $\frac{1}{4p(n)}$. (Recall that, from the preceeding argument, this happens with non-negligible probability).

B_1: P^* **equivocates the commitments.** Given that $x_n \notin L$, the commitment scheme used in the construction is statistically binding. Therefore this event can happen only with negligible probability. More precisely, the commitments to (a) J_1, J_2, (b) the root nodes of the Merkle-signature trees of the description and proof, (c) the VSS shares of the depths of the two trees, (d) the commitments in the second signature slot, (e) the commitments to the MPC-in-the-head views and VSS shares of the key are all statistically binding in both continuations C_1 and C_2 except with probability $\nu_1(n)$ for some negligible function $\nu_1(\cdot)$.

B_2: P^* **cheats in the MPC-in-the-head.** Since B_1 occurs with negligible probability, the MPC-in-the-head views in both continuations are statistically binding except with negligible probability. Hence from the t-robusteness of the MPC protocols, we have that in a random continuation from τ, the views for all nodes are consistent except with negligible probability. Now, it follows that, for each key, there exists tc_{d_M} and tc_{d_π}, the depths of the two trees and can be obtained by reconstructing the VSS shares committed to in P_2 and P_4 in τ.[10] There also exists $encode_M$ and $encode_\pi$ that represent the root values of the corresponding trees. Furthermore, all the functions in the MPC-in-the-head protocols are computed correctly. Hence, the values in all the nodes presented by the prover satisfy their respective conditions (as part of $\mathsf{MPC\text{-}Innode}$ and $\mathsf{MPC\text{-}PCPVer}$). Let the answers to the PCPP queries for depths tc_{d_M} and tc_{d_π} in the two continuations be $(pa_1, \ldots, pa_k, qa_1, \ldots, qa_k)$ and $(\tilde{pa}_1, \ldots, \tilde{pa}_k, \tilde{qa}_1, \ldots, \tilde{qa}_k)$. Let the keys committed in P_5 in the two continuations be f_1 and f_2.

B_3: P^* **commits to a machine M that predicts V's next message.** In this case P^* computes consistent trees and convinces the verifier using the same algorithm of the simulator. However, because the string r is chosen at random, the probability that this case happens is close to 2^{-n}, therefore is negligible.

We now estimate the probability that $f_1 = f_2 = f$. Let p be the probability that for a random continuation from τ, B_1, B_2 and B_3 dont occur. From the preceeding argument we know that $p \geq \frac{1}{4p(n)} - \nu_2(n) > \frac{1}{8p(n)}$ for some negligible function $\nu(\cdot)$. Since C_1 and C_2 are random continuations, it holds with probability at least p^2 that $f_1 = f_2$. f is chosen uniformly at random by A and is completely hidden. Hence with probability at least $\frac{p^2}{n}$, $f_1 = f_2 = f$. Whenever this happens, we claim that A can find two nodes γ' and γ that will yield two strings with the same signature under key sk_f. This is because from B_3, we know that the probability that the leaf values $(pa_1, \ldots, pa_k, qa_1, \ldots, qa_k)$ and $(\tilde{pa}_1, \ldots, \tilde{pa}_k, \tilde{qa}_1, \ldots, \tilde{qa}_k)$ came from consistent trees is negligible. However, the root nodes in both continuations are the same. This means for some node a collision must occur.

Thus, A obtains two random convincing continuations with probability at least $\frac{1}{p_2(n)}$ and with probability at least $\frac{1}{8(p(n))^2}$, it obtains a collision. Therefore, overall A obtains a collision with non-negligible probability and this is a contradiction.

Proof sketch of argument of knowledge: Proving argument of knowledge will essentially follow from the same approach as the proof of soundness. Assume P^* convinces a verifier on a statement

[10] This is not efficiently obtainable. Here we claim that there exists values because the commitments are binding and the VSS shares are consistent.

x. In the soundness proof, we crucially relied on the fact that the prover cannot equivocate any of the commitments and they were statistically binding. While proving argument of knowledge, this does not hold as x is in L. Instead, we observe that if the commitments were binding then P^* can find collisions to the signature scheme and that will happen only with negligible probability. Hence, it must be the case that P^* is equivocating the commitments. Since the commitments are instance based-commitments, if a commitment is opened to two different values, a witness can be extracted. The only witness for all the instance-based commitments used in the protocol is the witness to the input statement x. Since P^* equivocates with non-negligible probability, we can extract a witness with non-negligible probability. The extractor essentially runs the same algorithm as A does from the proof of soundness with the exception that it tries to find equivocated commitments as opposed to collision in the signatures.

C.2 Zero-Knowledge.

In this section, we describe our simulator.

High-level description:

We provide a high-level description of our simulator. The simulator internally incorporates V^* and proceeds as follows. It commits to sets J_1 and J_2 and proceeds to the first signature slot. After the first signature slot, it opens the set J_1 and obtains from V^* half of the secret keys (and the randomness used to generate the signatures under those keys). If the verifier fails to provide correct or consistent signatures, the simulator aborts. Otherwise, it sets up a procedure GetSign that will help the simulator to obtain signatures for arbitrary messages. This it achieves via Goldreich-Kahan approach. It estimates the probability with which the verifier replies in the signature slot and then uses that probability to identify the number of times GetSign needs to rewind in the slot to get a signature. This is the same approach pursued in [CPS13]. However, recall that we need to boost the probability of obtaining signatures. We let the verifier sign T commitments simultaneously. Using a Yao-type amplification, we can show that for most messages, a signature can be obtained. More precisely, we can show that for at least $n/2 - n/10$ of the remaining keys, the signature obtained on a commitment for m will be consistent with the deterministic signing procedure with probability at least $1 - O(\log^2(n)/n)$. This will essentially follow from the cut-and-choose mechanism.

Once the procedure is established, the simulator generates the Merkle trees for the description of the code. As mentioned earlier, the simulator chooses $d, u \in [\log n]$ randomly and alters the description size and running time by a factor of 2^d and 2^u respectively. For the altered code, it generates the Merkle signature tree for the description and the proof. Every time a signature needs to be obtained on a message m, the simulator invokes GetSign procedure T times each time requesting a signature on a new commitment of m. This ensures that the simulator will receive a signature of a commitment of m with high-probability for most keys. Once the tree is constructed, the simulator sends the encode part of the root (which is a vector of commitments of VSS shares), and commits to the VSS shares of the depth of the tree. Next, it receives the challenge string r and repeats the same process to construct the Merkle-signature tree for the proof.

Following this, the verifier provides randomness required to generate the PCPP queries for any possible depths. Now, for every depth and every unopened key, the simulator is required to provide commitments in the second signature slot. These commitments contain the nodes in the paths corresponding the PCPP queries for every depth and every unopened key. Recall that, for the simulator to succeed it needs to re-obtain signatures of the commitments in the actual tree for at least one key (and for one depth only).

A naive method to achieve this would be to directly place the commitments used to compute the tree (and for which the simulator already obtained the signatures in the first signature slot), for every

tree, and request the signatures. However, as pointed out earlier, this will skew the distribution as these commitments by definition were those on which the verifier gave signatures with high-probability in the first slot and hence might be detectable. To get around this, the simulator first generates random commitments and then inserts it in a manner that will not alter the distribution of the messages sent in the second slot. After the verifier provides the signatures in the second slot and opened the keys in J_2, the simulator finds a key among the remaining unopened ones for which the signatures obtained in the second slot match the ones obtained from the first slot. The cut-and-choose and boosting of the first signature slot will ensure that such a key will exist with high-probability. The simulator commits to the index of this key and the convinces the verifier in the rest of the protocol by creating the MPC-in-the-head views appropriately.

We now proceed to a more formal description.

The simulator Sim internally emulates the code of V^* and proceeds as follows:

Prover move P_0**:** Sim picks two disjoint subset of $\{1,\ldots,n\}$, J_1 and J_2 of size $n/2$ and $n/4$ respectively, and commits to them using $\mathsf{Com}^{\mathrm{x}}$. Recall that the honest prover uses $\mathsf{EQCom}^{\mathrm{x}}$.

Verifier move V_0 : Sim receives $(\mathsf{vk}_\kappa, \mathsf{C}_\kappa)$ for $\kappa = 1,\ldots,n$, from V^* where vk_κ is the κ^{th} verification key and C_κ) is a commitment to the randomness used to generate the key.

Prover move P_1**:** For each $\kappa \in [n]$, Sim sends $\{c_{\kappa,l}\}_{l\in[T]}$ to V^* where for each l, $c_{\kappa,l}$ is a commitment to 0^v using $\mathsf{Com}^{\mathrm{x}}$.

Verifier move V_1**:** V^* either aborts or produces signatures $\{\sigma_{\kappa,l}\}_{l\in[T],\kappa\in[n]}$

- We say that $\{\sigma_{\kappa,l}\}_{l\in[T],\kappa\in[n]}$ is *valid* if for all l and κ, $\mathsf{Ver}_{\mathsf{vk}_\kappa}(c_l, \sigma_{\kappa,l}) = 1$. If the set of signatures are not valid or the Verifier aborts, then the simulation halts immediately and outputs the transcript up to that point.
- Otherwise, Sim stalls the current execution and repetitively queries V^* at the Signing Slot with fresh commitments to 0^v (where v is the size of a VSS share), until it obtains $2n$ sets of valid signatures (i.e. $2n$ rewindings where the verifier did not abort in the signing slot or return any invalid signature). Let t be the number of queries Sim makes. If $t \geq 2^{n/2}$, then Sim aborts outputting fail_1.
- Next, it sets up a sub-procedure $\mathsf{GetSign}$ which is described below: Let c be a commitment to a message $m \in \{0,1\}^v$ and $\kappa \in [n]$.
  Procedure $\mathsf{GetSign}(c, m, \kappa)$: Initialize $SIGS = \phi$ to be an empty list.
  - For $i = 1$ to $2n^2tT^2$

    Choose i uniformly at random from $[T]$. Select $T-1$ commitments to m, $c_1,\ldots,c_{i-1}$, $c_{i+1},\ldots c_T$ where each of them are obtained by running $\mathsf{Com}^{\mathrm{x}}$ on message m. Set $c_i = c$. Rewind V^* to message P_1 and feed $(c_1,\ldots,c_T)$ (the first Signing Slot). If V^* replies a valid set of signatures $(\sigma'_1,\ldots,\sigma'_T)$ under verification key vk_κ, then σ'_i is a valid signature of $c_i = c$. Add σ'_i to the list $SIGS$.
  - If fewer than n signatures are in the list $SIGS$, then return $\perp$.
  - Let σ^* be that signature that occurs more than half the time in $SIGS$. Return σ^*. If no such signature exists return $\perp$.

Prover move $P_{1.2}$**:** Open set J_1.

Verifier and Prover moves $V_{1.2}$ and $P_{1.3}$: The simulator proceeds as the honest prover would. For every opened key $(\mathsf{sk}_\kappa, r_\kappa)$, it checks if the signatures obtained for the κ-th key are valid and deterministically signed. If not, it aborts.

Prover move P_2: Choose $i \leftarrow [\log n], j \leftarrow [\log n]$. Create a machine V^{**} that is functionally equivalent to V^* but is made to perform some no-op's at the end of the computation and the description is padded with an appropriate number of 0's so that, the running time of V^{**} is slower than V^* by a factor of 2^j and the size of the description of V^{**} is exactly 2^i times the size of V^*. Create a (extendable) Merkle signature-tree for $\mathsf{ECC}(V^{**})$ for every unopened signing key $\kappa \in [n]/J_1$. More precisely, to obtain any signature for any message m using any key κ, it proceeds as follows:

- Sim makes T calls to $\mathsf{GetSign}(c, m, \kappa)$ each time with a fresh commitment c to m. If any of the T trials succeed then it has obtained a valid signature of a commitment to m.

Finally for all trees computed with keys $\kappa \in [n]/J_1$, the simulator sends the values stored in the encode part of the root. If for any message m and key $\kappa \in [n]/J_1$, GetSign fails to return a signature in T tries, then the simulator adds κ to the list BadKeys and just sends dummy values for the roots of those trees. To create dummy values it generates VSS shares of the 0 string and commits to them. Finally, Sim commits to the depth $\log |V^*| + i$ by computing the VSS of the depth and committing to each share.

Verifier move V_2: Sim receives the challenge string $r \in \{0,1\}^n$.

Prover move P_3: Compute the PCP of Proximity proof, to prove that V^{**} predicts r. Create an (extendable) Merkle-signature tree for π. Again this is created for every key $\kappa \in [n]/(J_1 \cup \mathsf{BadKeys})$. To build this tree, the simulator obtains signatures just as in the previous step. As before, Sim sends the values stored in the encode part of the root of the tree if one is available for that key or dummy value if the GetSign procedure failed on some message. Finally, Sim commits to the depth $\log |\pi|$ by computing the VSS of the depth and committing to each share. As in step P_2, it appends BadKeys with keys for which the simualator fails to obtain the required signatures.

Verifier move V_3: Sim receives random strings $r_1, \ldots, r_{\ell_d} \in \{0,1\}^n$ from V^*, and computes queries: $(q_{i,j}, p_{i,j}) = \mathsf{Q}_{\mathsf{pcpx}}(a, r_j, i)$ with $i \in [k]$ and $j \in [\log^2 n]$.

Prover move P_4: For every unopened key $\kappa \in [n]/J_1$, V expects to see a sequence of nodes, for paths defined by queries $q_{i,j}$, $p_{i,j}$. More precisely, they are the commitments stored in the encode part of all the nodes in the paths. For the keys $\kappa \in \mathsf{BadKeys}$, the simulator simply presents commitments to 0^v. For every $\kappa \in [n]/(J_1 \cup \mathsf{BadKeys})$, the simulator proceeds as follows. Let N be the total number of commitments that it has to send for every key. Denote the commitments by $c_1, \ldots, c_N$. Define $\mathsf{VS}_j \subset [N]$ to be the subset of indexes that contain the commitments that the verifier expects to see for $q_{i,j}$, $p_{i,j}$ for $i = 1, \ldots, k$. The commitments for a particular key κ are generated as follows:

We know that there exists a j^* in $[l_d]$ which is the depth of the real tree committed in $(\mathsf{tc}_{dM}, \mathsf{tc}_{d\pi})$, sent respectively in step P_2 and P_3.[11] For the queries $(q_{i,j^*}, p_{i,j^*})_{i \in [k]}$, Sim has the values of nodes from the Merkle trees generated before. Let these commitments be denoted by

[11] Precisely, there are two levels j_1^*, j_2^* where j_1^* is the depth of the tree of the machine M and j_2^* is depth of the PCPP proof. However, for the sake of simplifying the notation, we use one depth only, that we denote by j^* and we assume that both the real tree of the machine M and the PCPP proof π, are of depth j^*.

$\{c'_\beta\}_{\beta \in \mathsf{VS}_{j^*}}$. Let D denote the set of all possible depths that simulator would use. So $j^* \in D$ and D contains $\log n$ depths.

Sim first generates $\tilde{c}_1, \ldots, \tilde{c}_N$ where each $\tilde{c}_i$ is a commitment to 0^v using $\mathsf{Com}^{\mathrm{x}}$.

- **CHECK GOOD:** For every $j \in D, \beta \in \mathsf{VS}_j$ run $\mathsf{GetSign}(\tilde{c}_\beta, 0^v, \kappa)$ and see if it returns a valid signature. If for any j, a signature was obtained for all $\tilde{c}_\beta$'s where $\beta \in \mathsf{VS}_j$, we say j is a good depth. If none of the depths were good then the simulator outputs fail_3. Otherwise, let j' be a random good depth.

It then replace all commitments in $\mathsf{VS}_{j'}$ by those in VS_{j^*} and all the commitments in VS_{j^*} with the actual commitments of the nodes, i.e. $\{c'_\beta\}_{\beta \in \mathsf{VS}_{j^*}}$.[12]

Finally, it sends the commitments for all keys in $[n]/J_1$ to V^*.

Verifier move V_4: The simulator receives all the signatures from the verifier. If the set of signatures are not valid or the Verifier aborts, then the simulation halts immediately and outputs the transcript up to that point. If for all $\kappa \in [n]/(J_1 \cup \mathsf{BadKeys})$, it holds that the signatures obtained for $\{c'_\beta\}_{\beta \in \mathsf{VS}_{j^*}}$ were different from the ones the simulator used to construct the respective Merkle trees then the simulator halts outputting fail_4. If not, let κ^* be the good key for which all the signatures match.

Prover move $P_{4.1}$: Open set J_2.

Verifier and Prover moves $V_{4.1}$ and $P_{4.3}$: The simulator proceeds as the honest prover would. For every opened key $(\mathsf{sk}_\kappa, r_\kappa)$, it checks if the signatures obtained for the κ-th key are valid and deterministically signed. If not, it aborts.

Prover move P_5: Send commitment to the VSS of κ^*. For every unopened key $\kappa \in [n]/J_1 \cup J_2$, Sim generates valid views for the MPC-in-the-head computation for all paths. More precisely, for key κ^*, Sim generates valid views for the MPC-in-the-head computation for every node of the paths corresponding to queries $(q_{i,j}, p_{i,j})_{\{j \in [\ell_d], i \in [k]\}}$. For every node belonging to the path of queries (q_{i,j^*}, p_{i,j^*}), Sim computes the MPC-in-the-head for $\mathcal{F}_{\mathsf{innode}}$ proving that nodes are consistent, and thus satisfy condition 2 of $\mathcal{F}_{\mathsf{innode}}$ in Fig. 6. For every other node, Sim successfully computes views of players of $\mathsf{MPC\text{-}Innode}$ for $\mathcal{F}_{\mathsf{innode}}$ by using the views of VSS of the depth j^* to prove condition 1 of $\mathcal{F}_{\mathsf{innode}}$. Sim acts similarly to compute views of protocol $\mathsf{MPC\text{-}PCPVer}$ for $\mathcal{F}_{\mathsf{VerPCPP}}$.

For all the keys $\kappa \in [n]/(J_1 \cup J_2)$ and $\kappa \neq \kappa^*$, it generates the MPC-in-the-head views for all nodes by using the VSS for κ^* as a trapdoor.

Verifier move V_5: Sim receives $\{p_1, \ldots, p_t\} \subseteq [n]$.

Prover move P_6: Sim opens the commitments to reveal the respective t views for every VSS, $\mathsf{MPC\text{-}Innode}$ and $\mathsf{MPC\text{-}PCPVer}$.

To analyze Sim, we introduce some notation. Let $p(m)$ be the probability that $\tilde{V}^*$ on query a random commitment $c = \mathsf{Com}(m, \tau)$ of $m \in \{0,1\}^l$ at the Signing Slot, returns a valid signature of c. Let $p = p(0^l)$.

Running time of the simulator: We first argue that the simulator runs in expected polynomial time. To start, note that Sim aborts at the end of the Signature Slot I with probability $1 - p$,

[12] We assume without loss of generality the number of commitments for each j are the same because each depth can be padded with the extra commitments to the 0 string.

and in this case, Sim runs in polynomial time. With probability p, Sim emulates V^* only a strictly polynomial number of times and size of V^* is bounded by $T_{\tilde{V}^*}$. Thus, Sim runs in some $T' = poly(T_{\tilde{V}^*})$ time and makes at most T queries to its $\mathsf{GetSign}$ procedure, which in turn runs in time $t \cdot poly(n)$ to answer each query. Also note that Sim runs in time at most 2^n, since Sim aborts when $t \geq 2^{n/2}$. Now, we claim that $t \leq 10n/p$ with probability at least $1 - 2^{-n}$, and thus the expected running time of Sim is at most

$$(1-p) \cdot poly(n) + p \cdot T' \cdot (10n/p) \cdot poly(n) + 2^{-n} \cdot 2^n \leq poly(T_{\tilde{V}^*}, n).$$

To see that $t \leq 10n/p$ with overwhelming probability, let $X_1, \ldots, X_{10n/p}$ be i.i.d. indicator variables on the event that V^* returns valid signatures for a random commitments to 0^s. If $t \leq 10n/p$ then via a standard Chernoff bound, we can conclude that $\sum_i X_i \leq 2n$ happens with probability at most 2^{-n}.

Indistinguishability of simulation:

Towards proving indistinguishability of the transcripts generated respectively by P and Sim, we will consider a sequence of intermediate hybrids $H_0, H_1, H_2, H_3, H_3^a, H_4, H_4^a, H_5, H_6, H_7$; starting from H_0, the view of the verifier V^* in the real interaction with the honest prover to H_7, the view output by the simulator Sim.

Hybrid H_1: This hybrid experiment proceeds identically to H_0 (the real interaction), i.e. the hybrid simulator using the witness to the statement follows the honest prover's strategy to generate all the prover messages with the exception that after receiving the verifiers V_1 message (i.e. signatures to a commitments to 0^v) it performs some additional computation. If the verifier provides valid signatures in V_1, it stalls the main simulation and then proceeds like Sim would after receiving V_1. More precisely, it will try to obtain $2n$ sets of valid signatures of commitments to 0^v, compute t and abort if $t \geq 2^{n/2}$. As shown in the running time analysis, using a Chernoff bound this happens with negligible probability. Since H_1 outputs a distribution identical to H_0 except when it aborts outputting fail_1, the distribution of the views output in H_0 and H_1 are statistically close. The running time of the hybrid simulator in this hybrid (and subsequent hybrids) is expected polynomial time following the same argument made for the actual simulator.

Hybrid H_2: This hybrid experiment proceeds exactly as the previous hybrid, with the exception that the hybrid simulator follows Sim's strategy in P_2 with a slight modification (described below). More precisely, to generate the message in P_2, it samples random i, j and sets up the verifier's code V^{**} using V^*, computes the error-correcting code $M = \mathsf{ECC}(desc(V^{**}))$ and generates the Merkle hash tree corresponding to M using the $\mathsf{GetSign}$ procedure just like the real simulator Sim with a slight modification. Instead of using $\mathsf{Com}^{\mathrm{x}}$ as the real simulator would, the hybrid simulator will use $\mathsf{EQCom}^{\mathrm{x}}$ for generating the tree. First, we note that P_2 generated by the hybrid simulator in this step is identically distributed to the hybrid simulator of H_1 since the distribution of the root node is identical when using $\mathsf{EQCom}^{\mathrm{x}}$. Second the rest of the messages are (and can be) generated according to the honest prover's strategy since all the later commitments are equivocated and revealed (only in P_6) according to the honest prover's strategy.

However, this hybrid could abort more often than H_1 because the hybrid simulator in H_2 could fail to get some signatures when generating the Merkle hash tree, i.e. fail_2. We show that this happens with negligible small probability in Claim 3. Hence, the views output in H_2 and H_1 are statistically-close.

Hybrid H_3: This experiment proceeds identically to H_2 with the exception that we compute the PCPP proof and generate the Merkle-hash tree corresponding to it. As in H_2 the hybrid simulator uses $\mathsf{EQCom}^{\mathrm{x}}$ instead of $\mathsf{Com}^{\mathrm{x}}$. It follows exactly as in the previous hybrid that the view output in H_3 and H_2 are statistically-close.

Hybrid H_3^a: This experiment proceeds exactly as H_3 with the exception that the hybrid simulator generates the message for P_4 differently. Instead of proceeding honestly, i.e. picking uniformly random commitments of 0, it will instead proceed like the real simulator with a small exception. If CHECK GOOD is true, it will simply swap the commitments in $\mathsf{VS}_{j'}$ and VS_{j^*}. The only thing we will have to argue here is that the distribution of the commitments fed as part of P_3 are not altered. Observe that, conditioned on not outputting fail_3, there will exist a good depth j'. Since j^* is uniformly chosen, this means that conditioned on not outputting fail_3, the distributions are identical since each commitment is independently chosen and the commitments in $\mathsf{VS}_{j'}$ are shuffled to a random position j^*. Therefore, it suffices to show that fail_3 is output with negligible probability. This is proved Claim 4.

We remark here that the honest prover simply generates each of the commitments for P_4 independently. The real simulator on the other hand needs to insert some fixed commitments. However, the other commitments of the simulator can be arbitrary as the simulator has a trapdoor to fake the nodes in those commitments. In particular, while the paths of different depths need to lead to the same root committed to in steps P_2 and P_3, by committing to different nodes the simulator is still able to fake the consistency of paths corresponding to irrelevant depths with a trapdoor. We crucially rely on the fact that the commitments made in P_2 and P_3 are the encode part of the root node and the connection from the label to the encode as part of the proof of consistency can be faked on irrelevant paths (and irrelevant keys). This is also what allows that simulator to replace the commitments in $\mathsf{VS}_{j'}$ with those in VS_{j^*}.

Hybrid H_4: This experiment proceeds exactly as H_3^a with the exception that the hybrid simulator will use the real simulator's strategy for P_4 (without any exceptions). Arguing H_4 is statistically close to H_3^a in this hybrid it suffices to show that the distribution of messages fed by the hybrid simulator to the verifier in P_4 is identical in hybrids H_3 and H_4. Recall that the only difference between H_3^a and H_4 is in generating the message P_3. More precisely, it replaces the commitments in VS_{j^*} with the actual commitments from the Merkle trees. Recall that signatures were obtained for all the actual commitments. This means that these commitments already passed the CHECK GOOD test by the mere fact that $\mathsf{GetSign}$ provided signatures for them. Therefore, the distribution fed will be identical because we are replacing one set of random commitments that passed the CHECK GOOD test with another set that passed the same test.

Hybrid H_4^a: This experiment proceeds exactly as H_4, with the exception that the hybrid simulator proceeds like the actual simulator to generate the views in P_5 with the exception that the commitments are generated using $\mathsf{EQCom}^{\mathrm{x}}$ instead of $\mathsf{Com}^{\mathrm{x}}$. To argue that the hybrids H_4 and H_4^a are statistically close, it suffices to show that the simulator outputs fail_4 with negligible probability. This happens in H_4^a only if for all the keys in $[n]/(J_1 \cup J_2 \cup \mathsf{BadKeys})$, a bad signature was obtained for some commitment at depth j^*. From the Cut-and-choose 2, we know that the probability that verifier answers incorrectly for more than s keys is negligible. This means that for all but s keys in $[n]/(J_1 \cup J_2 \cup \mathsf{BadKeys})$ the verifier answered in a deterministic manner. From Part(2) of Claim 3, we know that for at least $n/2 - n/10$ of the keys, any signature used by the simulator was correctly signed with probability at least

$1 - \frac{2s}{n}$ for $s \in \omega(\log n)$. Since J_2 is revealed, there will now be at least $n/4 - n/10 - s = \theta(n)$ remaining keys. We compute the probability that any of these are bad. By a union bound, for any of these keys, the probability that one of the $log^e(c)$ commitments from depth j^* yielded a bad signature is at most $\frac{2s \log^e(n)}{n}$. Since there are $\theta(n)$ remaining keys, the probability that all of them are bad is negligible. Hence the probability that the simulator fails to find a key for which the signature match and outputs fail_4 is negligible.

Hybrid H_5: We proceed identically to the previous hybrid with the exception of what values the equivocal commitments are revealed to in P_6. The honest prover reveals the VSS shares for any node as the t views generated by sharing 0^v and the MPC views generated by using the semi- honest simulator. In this hybrid, we instead will use Sim's strategy. More precisely, for the relevant paths, the simulator has generated the entire tree, it can reveal the corresponding values. For the irrelevant paths, the simulator will use the trapdoor (i.e. the length/running-time) to generate the MPC views. From the perfect t-privacy of the MPC protocols and VSS-sharing scheme, it follows that the view output by this hybrid is distributed identically to the previous hybrid.

Hybrid H_6: Observe that Hybrid H_5 is essentially the Sim's strategy with the exception that all commitments generated for messages P_2 through P_6 are made using $\mathsf{EQCom}^{\mathsf{x}}$ instead of $\mathsf{Com}^{\mathsf{x}}$. In hybrid H_6 we will change all these commitments to $\mathsf{Com}^{\mathsf{x}}$ and the indistinguishability will follow from the hiding property of the commitment scheme. The slight technicality here is that since our simulator is expected polynomial time, we might have to switch superpolynomially many commitments and indistinguishability will fail to hold. However, we employ the standard technique of considering the truncated experiments of H_5 and H_6 where we cut-off the simulation if it runs for more than $4t(n)/q$ steps where q is the distinghuishing probability of the hybrids and $t(n)$ is a bound on the expected running time of the simulator. By Markov inequality and a Union bound, we can conclude that the distinguishing probability of the truncated experiments is at least $\frac{p}{2}$. The indistinguishability of the truncated experiments follows directly from the hiding property of the commitment scheme. (This proof is essentially the same as presented in [DSMRV13])

Claim 3. *The probability that hybrid simulator in H_2 outputs fail_2 is negligible.*

Proof. We prove this in three parts: Let $s \in \omega(\log n)$.

1. For each signature key, we define a good set of random tapes G_n. Then we show that on a good tape the $\mathsf{GetSign}$ procedure obtains n signatures with high-probability.
2. We show that the probability of a random tape being good is at least $1 - n/T$.
3. We say that a signature key is good if $\mathsf{GetSign}$ fails to return a signature for at most $2s/n$ fraction of the good tapes. Recall that it returns a signature only if it obtains n signatures and there exists a majority. We show that there exists at least $n/2 - n/10$ keys that are good.

Assuming the above conditions, it holds that there are at least $n/2 - n/10$ keys for which the probability that a random tape is good with probability least $1 - n/T - s/n = 1 - 2s/n$. Since the simulator calls $\mathsf{GetSign}$ T times for any message, it will yield a signature for every message in the $n/2 - n/10$ coordinates with high-probability.

Part (1) We first note that by applying a Chernoff bound (just as in the analysis of the running time of Sim) $n/p \leq t \leq 2^{n/2}$ with probability at least $1 - 2^{-\Omega(n)}$. In this case, for every

$m \in \{0,1\}^s$, $p(m) \geq p - \nu \geq p/2$ implies that $t \geq n/2p(m)$. Fix a message m and a key. Define G_n to be the set of random tapes τ such that the probability that V^* returns a signature on (c, c_{-i}) where $i \leftarrow [n]$, $c = \mathsf{Com}^{\mathrm{x}}(m;\tau)$ and c_{-i} are random $T-1$ commitments for m is at least $\frac{1}{2tT^2}$. This means that, for any $\tau \in G_n$, in $2ntT^2$ tries, the probability that $\mathsf{GetSign}$ fails to return a single signature is at most e^{-n}. Since $\mathsf{GetSign}$ makes $n(2ntT^2)$ attempts, it obtains n signatures except with negligible probability.

Part (2) We argue that a random τ is in G_n with probability at least $1 - \frac{n}{T}$. Assume for contradiction the fraction of tapes in G_n was smaller than $1 - \frac{n}{T}$. We now estimate the probability that V^* returns a signature on random commitments. There are two cases:

Case 1: For a random set of commitments to 0^v, some commitment is not in G_n. Conditioned on this event, the probability that V^* honestly provides signatures is at most $\frac{T^2}{2tT^2}$.

Case 2: All commitments are in G_n. The probability this occurs is at most $(1 - \frac{n}{T})^T \leq e^{-n}$.

Overall the probability that V^* answers is at most $\frac{1}{2t} + e^{-n} < \frac{1}{t} < p(m)$ which is a contradiction. Therefore G_n must contain at least $1 - \frac{n}{T}$ fraction of the tapes.

Part (3) We need to argue that for most messages there will be a majority when n signatures are obtained. We will show that for most messages the n signatures have a majority and the popular signature will be the one that is deterministic signed. At the end of the first signature slot, the verifier opens the randomness and signing key used in the half the coordinates, i.e. those in J_1. Since J_1 is committed using an equivocal commitment in this hybrid, it is statistically hiding. So, given the commitments sent by the prover, J_1 is completely hidden in Hybrid H_2. Therefore, we can conclude that the probability that the verifier gives signatures that were not deterministically signed by more than s keys is at most $2^{-O(s)}$. Using an averaging argument, it holds that the probability that there exists more than $n/10$ keys such that the probability that the verifier gives "incorrect" signatures in those coordinates with probability bigger than $\frac{s}{10n}$ over messages sent in P_1 is negligible. That is, at this point $|\mathsf{BadKeys}| < n/10$ with high-probability. For the remaining $n/2 - n/10$ keys, at most $s/10n$ fraction of possible P_1 messages yield incorrect signatures. We argue next that a commitment to a message m will yield an incorrect signature with probability at most s/n. Assuming this holds, for any message m, the probability that $\mathsf{GetSign}$ return a deterministically signed signature for any of the $n/2 - n/10$ keys is at least $1 - \frac{s}{n}$. We now proceed to prove this claim.

The intuition is that if fewer than $1 - s/n$ fraction of the commitments yielded the correct signature with majority, then the messages containing this commitment will be concentrated in a small fraction. Recall that, each message sent in P_1 has T commitments and at least $1 - s/10n$ fraction are signed deterministically. Hence with probability at least $1 - s/n$ more than half of the T commitments must be deterministically signed and these commitments yield correct signatures with majority. Now suppose that fewer than least $1 - s/n$ fraction of commitments yielded the deterministically generated signatures in majority. Then it must hold that $(1 - s/n)^{T/2} \geq (1 - s/n)p(m)$. Since $T = O(n^c)$, we can set c sufficiently large ($c = 5$ will suffice) to arrive at a contradiction since $p(m) > \frac{n}{2^{n/2}}$.

This concludes proof of this part and the claim.

□

Claim 4. *The probability that the simulator outputs* fail_3 *in* H_3^a *is negligible.*

Proof. We need to show that CHECK GOOD fails to hold only with negligible probability. This happens only if all the $\log n$ depths fail to pass in the CHECK GOOD test. For any particular depth, at most $\log^e(n)$ commitments needs to be sent for some constant e. From Claim 3, we know for $n/2 - n/10$ keys, GetSign retruns a signature on a commitment with probability at least $1 - 2s/n$. By a union bound, the probability that at least one of the $\log^e(n)$ commtiments corresponding to a particular depth does not yield a signature through GetSign is at most $\frac{2s\log^e(n)}{n}$. Hence a particular depth will not pass the test with probability at most $\frac{2s\log^e(n)}{n}$. Since there are $\log n$ possible depths in D, the probability all these depths fail the test simultaneously is negligible.

□