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 show how to construct a $O(1)$-round resettably-sound zero-knowledge argument of knowledge based on one-way functions where additionally the construction and proof of security is black-box. Zero-knowledge proofs (ZK) are fundamental cryptographic constructs used in numerous applications. Formalized using a "simulation" paradigm, ZK requires that for every malicious verifier there exists a "simulator" that can indistinguishably reproduce the view of the verifier in an interaction with the honest prover. Resettable-soundness introduced by Barak, Goldreich, Goldwasser and Lindell (FOCS 01) additionally demands the soundness property to hold even if the malicious prover is allowed to "reset" and "restart" the verifier. Using the breakthrough non-black-box technique of Barak (FOCS 01) they also provided a constant-round construction of a resettably-sound ZK argument relying on the existence of collision-resistance hash-functions. This construction and subsequent constructions all rely on the underlying cryptographic primitive in a non black-box way. Recently, Goyal, Ostrovsky, Scafuro and Visconti (STOC 14) showed how to extend the Barak's technique to obtain a construction and proof of security that relies on the collision-resistant hash-function in a black-box manner while still having a non black-box simulator. Such a construction is referred to as *semi black-box*. From the work of Chung, Pass and Seth (STOC 13) we know that the minimal assumption required to construct resettably-sound ZK argument is the existence of one-way functions.

In this work we close the gap between (semi) black-box and non black-box constructions by showing a black-box (round-efficient) resettably-sound argument relying on one-way functions only.

1 Introduction

Zero-knowledge (ZK) interactive protocols [GMR89] allow one player (called the Prover) to convince another player (called the Verifier) of the validity of a mathematical statement $x \in L$, while providing *zero additional knowledge* to the Verifier. We are here interested in a stronger notion of zero-knowledge arguments known as *resettably-sound zero-knowledge*. This notion, first introduced by Barak, Goldwasser, Goldreich and Lindell (BGGL)[BGGL01], additionally requires the soundness property to hold even if the malicious prover is allowed to "reset" and "restart" the verifier. This model is particularly relevant for cryptographic protocols being executed on embedded devices, such as smart cards. BGGL provided a construction of a resettably-sound zero-knowledge argument for NP based on the existence of collision-resistant hash-functions. More recently, Bitansky and Paneth [BP12] presented a resettably-sound zero-knowledge argument based on the existence of an *oblivious transfer (OT) protocol*. Finally, Chung, Pass and Seth (CPS) [CPS13] showed how to construct based on the minimal assumption of the existence of one-way functions (OWFs).

It was shown in [BGGL01] that resettably-sound ZK arguments inherently require non-black-box simulation. In this work we are interested in understanding how the underlying cryptographic primitive is used. Constructions based on general cryptographic assumptions such as existence of OWFs or CRHs come in two flavors: black-box or non-black-box. The former requires that the construction refer to only the input/output behavior of the underlying primitive, while the latter requires to use the code of the functionality implementing the primitive.

An important theoretical question is to understand the power and limitations of black-box constructions in cryptography. A long line of work starting from the seminal work of Impagliazzo and Rudich [IR88] have tried to understand this for various primitives. Besides strong theoretical motivation, a practical reason is related to efficiency. Typically, non-black-box usage results in inefficient constructions due to inefficient NP-reductions, and this impacts both the computational and communication complexity. As such, non-black-box constructions are used only to demonstrate "feasibility" results. The first step towards efficiency would be to achieve the same result with a construction that makes only black-box usage of the underlying primitives.

Most of the previous constructions of resettably-sound ZK arguments crucially make non black-box usage of the underlying primitive. Very recently, [GOSV14] showed how to construct a public-coin ZK argument of knowledge based on CRHs in a black-box manner. They provide a non black-box simulator but a black-box construction and proof of security based on CRHs. Such a reduction is referred to as a *semi black-box* construction (see [RTV04] for more on different notions of reductions). Applying the [BGGL01] transformation to this protocol yields the first (semi) black-box construction resettably-sound ZK argument that relies on CRHs. In this paper, we address the following open question:

> *Can we construct resettably-sound ZK arguments under minimal assumptions where the underlying primitive is used in a black-box way?*

1.1 Our Results

We resolve this question and provide a black-box construction for resettably-sound ZK argument based on the minimal assumption of the existence of one-way function. 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.*

We know from the lower bound of [BGGL01] that constructing such protocols inherently requires non-black-box simulation. Hence the best we can hope for is a semi black-box construction.

It might seem that achieving such result is just a matter of combining techniques from [GOSV14] and [CPS13]. However, it turns out that the two works have conflicting demands on the use of the underlying primitive which make the two techniques "incompatible".

In more detail, the construction presented in [GOSV14] crucially requires the honest prover to use the collision-resistant hash-function for generating its messages. In particular, it uses the ability of the prover of evaluating the hash function *on its own*, based on the partial transcript of messages exchanged. The resettably-sound zero-knowledge protocol proposed in [CPS13], roughly, replaces the hash function with digital signatures (which can be based on one-way functions). However, the construction of [CPS13] requires that the honest prover never has to use the signatures for generating its messages, as it cannot compute signatures on it own.

Our starting point will be the constant-round resettably-sound ZK argument of [GOSV14] based on CRHs. Our central contribution is to show how to adapt the techniques of [CPS13] to obtain an analogous construction relying on one-way functions only.

1.2 Previous works and why they don't work

We briefly review some of the relevant previous works that rely on Barak's non black-box approach.

Barak's ZK simulation and protocol [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 commits to a machine that predicts the verifier's next message, or 2) $x \in L$. By definition the (non-black-box) ZK 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 without knowing the code/randomness of the verifier. The main bottleneck in translating this beautiful idea to a concrete construction is the size of the 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 ZK proof that can *hide* the size of the witness. More precisely, they rely on Universal Arguments (UARG) which they show can be constructed based on collision-resistant hash-functions (CRHs) 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 on the other hand allow committing to strings of arbitrary length with the additional property that to verify the bit in any position of the string only a small amount of supplementary decommitment information needs to be provided to verify the bit. 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 responds with the bits in the respective locations along with the supplementary information. For those familiar with Merkle hash-trees, the additional information is called authentication path and consists of all nodes and their sibling in the path from the leaves to the root. This approach reveals the size of the proof as the tree size can be determined from the length of the path. While this approach allows constructing arguments for statements of arbitrary polynomial size, it is not zero-knowledge because the authentication path reveals more information about the proof. To get ZK, Barak's construction requires that the prover never reveals the supplementary information in the clear, but instead only commit to the supplementary information and prove via standard ZK that there are valid openings to the commitments that prove consistency of the bits revealed in the PCP. A standard ZK can be employed as the size of this statement is strictly polynomial. However, since Merkle-hash trees rely on CRHs, proving consistency in zero-knowlege inherently relies on the code of the hash function and is thus non black-box.

Summing up, the two ingredients required in Barak's ZK protocol are: (1) the proof can be compressed so that the running time required to *verify* the proof is polynomial and (2) the supplementary information required to verify bits in the proof must be hidden.

Black-box implementation of Barak's NBB simulation [GOSV14]. Recently, Goyal, Ostrovsky, Scafuro and Visconti in [GOSV14] showed how to implement Barak's simulation technique using the hash function in a black-box manner.

The first observation in [GOSV14] is that in order to use a hash function as a black-box, the prover must unfold each path of the Merkle tree corresponding to the PCP queries and let the verifier check the hash consistency of the path with the committed root by recomputing the hash values on its own. However, as mentioned before, we can not hope to achieve ZK if we reveal the path, since the size of the revealed path reveals the size of the proof.

To tackle this, they introduce the concept of *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 using the hash function in a black-box manner, 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, recall that to prove consistency of a leaf of the tree, one has to reveal all the nodes along the path between the leaf and the root, and prove their connections. This can be viewed as *locally* verifying for each node the connection with its two children. Consider any parent node, verifying a connection with its 2 children in the hash-tree entails: (1) computing the hash of the two children; (2) verifying that the parent node "corresponds" to the hash value just computed. In the standard Merkle tree, the parent node is computed as the hash of the children. Hence, to verify (2) it is sufficient to check that the hash computed in (1) is equal to the value of the parent node. In [GOSV14], the parent node instead of being simply the hash of its children, is a *redundant representation* of it (looking ahead, it is the secret-sharing of it). The crucial observation of [GOSV14] is that while checking (1) requires the use of the hash function and the verifier directly checking the hash-computations, condition (2) can be proved indirectly using a black-box ZK protocol. With this abstraction of nodes, to allow an honest prover (and the simulator) to create new nodes on the fly, it is sufficient to provide them with a *trapdoor* to cheat in the ZK proof for (2). 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 he is required to answer to PCP queries that are greater than the size of the committed PCP. The actual construction of [GOSV14] uses VSS for the representation of the nodes, and the MPC-in-the-head due to Ishai, Kushilevitz, Ostrovsky and Sahai [IKOS07] for the ZK proof for condition (2).

The final piece of their construction deals with committing the actual code of the verifier which also can be of an arbitrary polynomial size. This can be achieved by using the same LEGO approach to commit to the code. However, a minor issue here is that to verify a PCP the verifier needs to read the statement (in this case the code of the machine) or at least the size of the statement to process the queries. But revealing the size will invalidate ZK again. [GOSV14] get around this problem by relying on PCP of Proximity (PCPP) 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.

Summing up, some of the key ideas that allow [GOSV14] 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 [GOSV14] 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

CRH in a black-box manner. A detailed overview of the approach can be found in App. B.

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. We recall some of the ideas from this work. Recall that hash functions are needed in two locations in Barak's protocol. First, since there is no *a-priori* polynomial upper-bound of the length of the code of V^*, the simulator needs to commit to the hash of the code of V^*. Secondly, since there is no *a-priori* polynomial upper-bound on the running-time of V^*, the use of universal arguments is required (and such constructions are only known based on the existence of collision-resistant hash functions).

The main idea of CPS is to notice that digital signature schemes—which can be constructed based on one-way functions—share many of the desirable properties of CRHs, and to show how to appropriately instantiate (a variant of) Barak's protocol using signature schemes instead of using CRHs. More precisely, CPS show that by relying on strong fixed-length signature schemes, one can construct *signature tree* analogous to the Merkle hash-tree that could be used to compress arbitrary length messages into a signature of fixed length and 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 the verifier generates the secret key and provide signatures of messages to the prover. Towards this, CPS adds a signature slot at the beginning of the protocol. More precisely, in an initial stage of the protocol, 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 of some message to the verifier, and the verifier computes and returns a valid signature σ of c (using sk). This is used by the simulator to construct a signature tree through rewinding the (malicious) verifier as a fake witness for WIUARG in an analogous way as before. 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 we can construct public-coin UARG, the prover can first engage in a shadowed UARG where the prover merely commits to its messages and then in a second phase proves that either $x \in L$ or the messages it committed to consistute a valid UARG proving that the prover knows 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.

Formalizing this idea turns out to be tricky. First, a malicious verifier might fail to provide signatures for specific messages thereby preventing the simulator from generating the signature tree. To address this, following the works of Lin and Pass [LP12], they consider signatures of commitments of messages thereby preventing the verifying from aborting based on the messages. Second this protocol is not public-coin, CPS nevertheless shows that it suffices to apply the PRF transformation of BGGL to obtain a protocol that is resettably-sound.

It seems inherent in the work of CPS that they need the shadowed UARG and hence proving anything regarding this shadowed argument will rely on the underlying OWF in a non black-box manner.

Competing requirements of [GOSV14] and [CPS13]. Recall that in [GOSV14], in order to use the CRH in a black-box manner, the prover is required to actually show the paths corresponding to the PCPP queries to the verifier. To preserve size-hiding, the prover needs the ability to arbitrarily extend the paths of the tree, by "secretly" generating fake nodes and new connections, and this is possible because the prover can compute the hash function on its own. In contrast, in [CPS13] the honest prover never uses signatures.

Therefore, there are competing requirements between the works of [GOSV14] and [CPS13]. While [GOSV14] requires the honest prover to generate hash-computations on-the-fly, [CPS13] requires the honest prover to not rely on any signatures as the only way it can obtain one is through the verifier. The contribution of our work is to come up with a technique that will allow us to use 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

At the core of the issue of combining the techniques of [GOSV14] and [CPS13] there is the requirement in [GOSV14] that the prover collects the signatures required to compute any tree (which can not be previously bounded), while in [CPS13] the prover must not use the signatures, as it cannot obtain an unbounded number of signatures from the verifier (without rewinding).

The first observation we make is that the number of paths that are actually revealed in [GOSV14] is bounded, in fact, poly-logarithmic. Also, the prover of [GOSV14] does not really need to compute the entire tree, but he needs to compute only the paths that later must be revealed. Therefore, the number of signatures that the prover needs to get from the verifier is fixed and known in advance. Now, given that the LEGO technique of [GOSV14] allows the prover to cheat in the proof of consistency of the node connections, one could envision the following approach. In the signature slot, which is placed at the very beginning of the protocol, the prover sends all the nodes[1] required to construct the paths that will be used later to answer to the PCPP queries. Then, once the PCPP queries are revealed, the prover proves that exactly those nodes (sent at the very beginning), form accepting paths for the PCPP queries just asked, cheating in the proof using its trapdoor (i.e., the knowledge of the witness).

However, this approach introduces the following problem. In an execution with the honest prover, the verifier sees a bunch of nodes in the first slot, then after the PCPP queries are revealed, it sees the proof of consistency of exactly those nodes. Indeed, the honest prover can prove the consistency of any path using its witness as a trapdoor, and he is not actually proving that those paths are consistent to the PCPP queries.

Now, the same thing should happen in the simulated transcript, namely, the simulator should be able to prove consistency of exactly the same nodes that appear in the signature slot. However, the simulator cannot pass the proof of consistency using the witness for $x \in L$, as the prover does. Instead in order to provide an accepting proof, he needs to open paths that are actually consistent with the PCPP queries and the real tree. But, the simulator will know such paths only after the PCPP queries are revealed, namely, only after he is already committed with the nodes shown in the signature slot.

To get around this problem, we could propose to place the signature slot in a suitable place so that we can achieve both these requirements. Suppose, we placed it right after the PCPP queries. The honest prover can still obtain the required signatures, while the simulator can place the relevant nodes after seeing the PCPP queries. The problem of this approach is that the simulator needs to play with fake prover messages (i.e., to compute a fake tree) to reach the signature slot,

[1] More precisely, the commitments to the nodes.

and then rewind all the way up before the commitment of the root and change such commitment. At first, this seems to be possible since the verification key is fixed in the first message and the signatures are unique. However, the simulator in two random continuations can have different, or worse, correlated strategies to abort in the signature slot. This means that the messages for which it received signatures in one continuation might not be accepted in another continuation. While [CPS13] uses commitments to avoid the aborting issue in their protocol, it is not applicable here as the aborting strategy could depend on the commitment itself. Before we explain this problem in more detail, we explain our approach since we will encounter the same issue again.

Our approach is to use two signature slots: one at the very beginning, where the verifiers signs only one message, and one after the PCPP queries are revealed, where the verifier signs the bounded number of messages corresponding to the number of nodes to be shown in the paths. The honest prover obtains its signatures in the second slot. The simulator on the other hand will use the first slot to generate the entire tree, while in the second signature slot, after the PCPP queries are known, he re-submits the same nodes used to compute the real tree (note that for the queries that are beyond the real tree, the simulator will just compute fake nodes on the fly as in [GOSV14]).

As mentioned before, this approach also suffers the correlated attack. Consider a malicious verifier that aborts on all messages that start with bit 0 in the first slot and aborts on all sets of messages in the second slot if more than half 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.

Our final approach is the following. First we amplify the probability that the verifier gives a signature in the first slot. We will ask the prover to provide $T = O(n^{c+1})$ random commitments to the same message in the first slot. 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.

The final piece of the puzzle is to amplify the success probability. Towards this we recall that in [GOSV14], to ensure size hiding, the verifier supplies PCPP queries corresponding to every possible depth of the tree (this is at least 1 and at most $\log^2 n$). Hence, the prover will be required to commit to the actual depth of the tree along with the commitment of the root node and the simulator will have to provide correct responses to the PCPP queries only for this depth. In our final approach:

- The simulator will choose a random i and j from $\{0, \ldots, \log n\}$ and use a modified version of the code of the verifier whose size is 2^i bigger and whose running time is 2^j slower. This

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

it does through padding the description and adding no-op instructions and commit to the appropriate depth.

- In the second signature slot, for all tree depths that are not the actual depth, the paths corresponding to the PCPP queries will be answered with fake paths, computed on the fly, while the path corresponding to the actual depth needs to be the commitments the simulator used to generate the tree earlier. Plugging in the actual commitments directly might *skew* the distribution and could be detected by the verifier. Instead, it will "test the waters" first: it will first generate random commitments and swap these commitments with the actual commitments if the following condition is met: More precisely, the simulator, for each random commitment, will rewind the verifier to first signature slot and feed this commitment in the first slot to check if it returns a signature. If for any of these commitments the first slot fails to produce a signature then the simulator rewinds the verifier to end of first signature slot and starts all over again, but now with new values for i and j.

Roughly the intuition is that if the first slot provided a signature for all these random commitments, the simulator can swap them with the actual commitment and then feed them to the verifier. This *swapping* argument ensures that the distribution fed in the second message is identically distributed to randomly generated commitments because we replace one good commitment with the other. Making this preceeding argument work requires a little more effort. First, we need to show that the simulator does not have to repeat several times. Second, the running time analysis requires combining the Yao-amplification with the rewinding strategy of CPS (that in turn, relies on the Goldreich-Kahan [GK96] strategy). The formal proof can be found in Appendix D

Finally, we remark that in [GOSV14], CRHs are required for constructing commitment scheme that are secure against selective-opening attacks (SOA). SOA-security is necessary in their construction (and in ours) for the following reason. At some point of the protocol, the verifier will ask the prover to reveal t out of n commitments, and the security proof crucially relies on the hiding of the remaining unopened commitments. Statistically-hiding commitments satisfy a (weaker) notion of hiding in presence of selective opening attacks which is sufficient for [GOSV14] (and for us). Unfortunately, we do not know how to construct constant-round statistically-hiding commitments from OWFs. To avoid this round complexity, we observe that we can workaround the selective opening issue by using equivocal commitments. In our final construction, we use instance-dependent equivocal commitment schemes, introduced by Lindell and Zarosim [LZ11] which can be constructed based on one-way functions. On a high-level, such commitment schemes, will allow a witness-holder to equivocate but is computationally-binding to any adversary that does not have the witness. In our case, the honest prover will possess a witness to equivocate all commitments, while the cheating prover (and the simulator) cannot equivocate all commitments. We also require extractable (equivocal) commitments to allow for extraction in proving soundness and proof-of-knowledge property of our protocol.

2 Protocol

Interestingly, our ZK protocol is not designed for the prover to prove that "$x \in L$". The entire protocol is instead designed for the simulator to prove knowledge of the next-message-function of the verifier (also called the trapdoor theorem).

Our protocol consists of a sequence of steps in which the prover sends instance-dependent equivocal commitments[3] – which essentially are commitments to 0 – except for the last step, where P is required to open some of the commitments in a convincing manner.

The key idea is that, by the time the prover reaches the last step, it knows exactly what the verifier expects to see. Specifically, the (equivocal) commitments sent in the previous rounds were supposed to be nodes of the extendable Merkle tree along with their relative proof of consistency. In an extendable Merkle tree[4], a node is vector of VSS shares, and a proof of consistency is a vector of views of MPC-in-the-head executions. Therefore, the verifier expects to see t accepting and consistent views of VSS and MPC players.

The prover successfully passes the last step as follows. First, it computes t accepting views by running the simulator guaranteed by the t-privacy of the underlying MPC/VSS protocol. Second, it equivocates the necessary commitments to open to such views, using the knowledge of the witness.

We now proceed with an high-level description of the protocol. A detailed description of the protocol is provided in App. C, while the detailed description of the simulator and the proof of security are shown in App. D.

Zero-knowledge Argument of Knowledge.

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 $(x, w) \in \mathbf{R}_L$.

V_0: $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{Gen}(1^n)$. Send vk to P

Stage One (Signature Slot 1):

- P_1: Send $\mathsf{T} = O(n^c)$ commitments to 0^n (for some constant c).
- V_1: Signs each commitment.

Stage Two (Commitment to the extendable Merkle Tree of the Trapdoor theorem)

P_2: Send equivocal commitments (i.e., commitments to nothing), which are supposed to be the commitments to the root of the extendable Merkle tree for the trapdoor theorem (i.e., $\mathsf{ECC}(M)$), and the commitments to VSS of the depth of such tree.

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: $\mathcal{L}_\mathcal{P} = \{(a = (r, \mathtt{t}), (Y)),\ \exists M \in \{0,1\}^*$ s.t. $Y \leftarrow \mathsf{ECC}(M),\ M(z) \rightarrow r$ within $\mathtt{t}$ 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).

Stage Three (Proof):

- P_3: **(Commitment to the extendable Merkle Tree of the PCPP proof)** Send equivocal commitments (i.e., commitments to nothing) which are supposed to be the commitments of the root of the extendable Merkle tree representing the PCPP proof, and the VSS of its depth.

[3] Instance-dependent equivocal commitments for an instance x can be equivocated using the witness for $x \in L$; such commitments can be constructed for instances of the Graph Hamiltonicity language, and black-box use of a OWF (see Section A.3).

[4] See App. B for more details on extendable Merkle tree.

- V_3: Send the random tapes for the PCPP queries.

  V and P obtain queries (q_i^j, p_i^j) for $i \in [k], j \in [\log^2 n]$, where j represents each possible length of the trapdoor theorem/PCPP proof, and k is the query complexity of the PCPP system. Each of such queries defines a path of a virtual tree. For example, q_i^j identifies a path of virtual depth j in the tree of the trapdoor theorem (i.e., $\mathsf{ECC}(M)$), while p_i^j identifies a path of virtual depth j in the tree of the PCPP proof. A path is a sequence of nodes. In the extendable Merkle tree, a node is a concatenation of VSS shares.
- P_4: Send equivocal commitments which are supposed to be the commitments of the nodes (i.e., the VSS shares) along the paths defined by the PCPP queries.
- V_4: (**Signature Slot 2**) Sign each commitment received from P.
- P_5: The obtained signatures are the *labels* of the nodes sent in step P_4. Therefore, now each path previously committed is completed with labels, which makes the path "canonical". At this point it is possible to provide a proof of consistency for each path. A proof of consistency for a path of the extendable Merkle tree, consists of a sequence of MPC-in-the-head computations, executed on top of the VSS of each node. Thus, a proof of consistency for a path consists of an MPC-in-the-head execution for each node. Such MPC-in-the-head are committed to using an *extractable* equivocal commitments (See Fig. 3).

  However, again P instead of sending commitments of views of MPC players proving consistency of the nodes previously sent, it sends a sequence equivocal commitments, i.e., commitments to nothing.
- V_5: Select t among n players to check out, namely, $p_1, \dots, p_t$.
- P_6: P now computes all the VSS shares and the views of the MPC protocols that will make the verifier accept the proof. This can be done thanks to the knowledge of the t players that V wants to check. Indeed, P at this point runs the simulator guaranteed by the perfect t-privacy of the MPC protocols used in the proof of consistency. Using the simulator P obtains the views for players $p_1, \dots, p_t$, that will make the verifier accept the proof. Finally, P uses the witness w to equivocate every commitment of the views corresponding to the players in positions $p_1, \dots, p_t$.

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. Soundness relies on the collision-resistance of the signature scheme and the soundness of the PCPP. We rely on ideas from [GOSV14, CPS13, BG02] to prove argument of knowledge. However, for proving both soundness and argument of knowledge, we require commitments to be extractable. More specifically, we require that only the commitments sent in step P5 are extractable and equivocal. There reason it is sufficient that only commitments in P5 are extractable is that they are commitments of MPC views which *contain* also the values of the nodes (the VSS).

Obtaining Resettable-soundness. The protocol above is a (semi) black-box construction of an "almost" public-coin ZK argument of knowledge. It is public-coin except for the signature slots. We can obtain resettable-soundness by applying the BGGL transformation to the protocol presented above. Just as in [CPS13], we transform the protocol (P, V) described in the previous section as follows. The verifier will initially pick a seed for a pseudo-random function family (PRF), then apply the PRF on the partial transcript to generate each verifier message except the messages in the signature slot. The only difference between this and the original protocol is in how the verifier challenges are chosen. Arguing that the protocol is a resettably-sound zero-knowledge argument of knowledge essentially follows from [BGGL01].

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.

[BP12] Nir Bitansky and Omer Paneth. From the impossibility of obfuscation to a new non-black-box simulation technique. In *FOCS*, 2012.

[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.

[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.

[GK96] Oded Goldreich and Ariel Kahan. How to construct constant-round zero-knowledge proof systems for np. *J. Cryptology*, 9(3):167–190, 1996.

[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.

[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.

[GOSV14] Vipul Goyal, Rafail Ostrovsky, Alessandra Scafuro, and Ivan Visconti. Black-box non-black-box zero knowledge. *to appear.* In *STOC*, 2014.

[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.

[LZ11] Yehuda Lindell and Hila Zarosim. Adaptive zero-knowledge proofs and adaptively secure oblivious transfer. *J. Cryptology*, 24(4):761–799, 2011.

[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.

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 non-interactive 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 $\mathcal{L} \in \mathbf{NP}$, 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. 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 1, 2 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 1: Non-interactive, instance-dependent equivocal commitment scheme.

Equivocal Commitment

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

Adaptive Opening

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

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 2: Equivocation.

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. 3.

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

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 3: Extractable Commitment.

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

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 $V_{x,r_j}(\alpha) = V(x, r_j, \alpha)$,[5] where $x \in \{0,1\}^n$ and $j \in [t]$. Each $V^{(j)}(x)$ is called an incarnation of V .*

[5] 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.

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.*

A.7 Strong 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 $\mathcal{F} : (\{0,1\}^*)^n \rightarrow (\{0,1\}^*)^n$, 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 $\mathcal{F} : (\{0,1\}*)^n \rightarrow (\{0,1\}*)^n$ 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[6] 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.

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

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 $\mathtt{Share}(s)$, 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 $\mathtt{Recon}$, players exchange their views of the share stage, and reconstruct the value s. We use notation $\mathtt{Recon}(\mathsf{S}_i, \ldots, \mathsf{S}_n)$ 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* $\mathtt{VSS} = \langle \mathtt{Share}, \mathtt{Recon} \rangle$ *that implement respectively the sharing and reconstruction phases as follows.*

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

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

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* $\mathtt{Recon}$*.*

- **Correctness:** *if the dealer is honest throughout the protocols then each honest player will output the shared secret* s *at the end of protocol* $\mathtt{Recon}$*.*

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 $\phi(\texttt{Recon}(\mathsf{S}_1, \ldots \mathsf{S}_n)) = 1$. This idea is crucially used in [GOSV14].

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 $\epsilon : \mathbb{N} \rightarrow [0,1]$ 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 The extendable Merkle Tree of [GOSV14]

For completeness, in this section we provide an overview of the techniques introduced in [GOSV14] to achieve black-box construction for Barak's ZK simulation strategy.

The main contribution of [GOSV14] is to provide a black-box size-hiding commit-and-prove scheme. In nutshell, this primitives allows one to commit to an arbitrary long string (with size upper bounded by $n^{\log n}$), and later prove some property about a subset of bits of the string in a succinct and size-hiding manner.

This primitive is extremely useful in Barak's ZK protocol where the simulator first has to commit to the code of V^*, which cannot be bounded by any polynomial, then it has to compute to the PCP proof, which cannot be bounded as well. Then, one uses the black-box proof phase to prove to a verifier that the PCP verifier would have accepted the string committed.

The main ingredient to achieve BB commit-and-prove, is a novel construction of the Merkle tree, called extendable Merkle tree. In this tree, each node is represented as a pair $[label, VSS]$, where VSS is the vector of the n shares of $label$ computed using VSS. This redundant representation allows the prover to partially open every node over a committed path, by revealing t out of n shares of the VSS, and letting V verify the correctness of the share, and the hash consistency with the parent node. We provide more details on the construction of the extendable merkle tree in Sec. B.1.

Finally, the ZK protocol of [GOSV14] uses PCP of Proximity instead of regular PCP. This is necessary to avoid that the decision function of the PCP verifier reads the entire theorem, as the size of such function depends on the size of the theorem. Instead, with PCP of Proximity, the PCPP verifier needs to read only few bits of the theorem as well, therefore allowing for size-hiding proof of the PCPP predicate.

B.1 Extendable Merkle Tree

The crucial building block that we use also in our construction is the extendable Merkle Tree. An extendable merkle tree allows to commit to a string of arbitrary size and then prove a property about some bits of such string without revealing anything about the string, not even the size.

The extendable Merkle tree is built according to the following key ideas:

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* $\mathtt{Share}(s)$*. In the paper we use* $\mathsf{S}^{\gamma}_{\mathsf{tree}}[i]$ *to denote the* i*-th share of the VSS representation of the label of a node* γ*, along the tree* tree*, or* $\mathsf{S}_{\mathsf{string}}[i]$ *to denote the* i*-th share of the VSS of the value* string*.*

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

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

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:* $\mathsf{L}^{\gamma} = [h(\mathsf{S}^{\gamma 0}[1]) \mid \cdots \mid h(\mathsf{S}^{\gamma 0}[n]) \mid h(\mathsf{S}^{\gamma 1}[1]) \mid \cdots \mid h(\mathsf{S}^{\gamma 1}[n])]$*. Each VSS view is* hashed separately*.*

Proving a predicate about a node. To prove any predicate ϕ about a node $[label^{\gamma}, \mathsf{S}^{\gamma}[1], \ldots, \mathsf{S}^{\gamma}[n]]$, the prover computes an MPC-in-the-head for a functionality that takes in input the shares of each players $\mathsf{S}^{\gamma}[i]$, reconstructs their secret $label^{\gamma}$ and outputs 1 iff $\phi(label^{\gamma})$ is satisfied. For an example of such functionality, the reader is referred to Fig. 4 and Fig. 5.

This technique of computing MPC on top of VSS (already used in [GLOV12]), allows the prover to prove any statement about the nodes in a *BB manner*: the prover commits to the views of such MPC (along with the VSS shares that were possibly committed earlier), then the verifier checks the validity of the statement by looking at only t views of the MPC.

We stress that the computation of any proof concerning the nodes, can be done independently from the computation of the node itself. Namely, the prover can compute a bunch of nodes, and only when required, it proves that they are consistent to each other, or that they satisfy some predicate.

Achieving input size-hiding. The size of the string committed, and hence the depth of the corresponding tree is not known to the verifier. Therefore, when there verifier wants to test some bits of the string, it will send a set of position to test for *each possible depth* of the tree.

For instance, in the ZK protocol of [GOSV14] and of ours, V will send one set PCPP queries for each possible depth j of the tree of the PCPP. Namely, V sends $\{q_i^j, p_i^j\}_{i \in k}$.

C Details of our resettable-sound ZK protocol

In this section we provide more details on the protocol shown in Section 2

Recall that, in our protocol the prover commits to the root of two (extendable) Merkle Trees, one for the code of a machine, the other for the PCPP proof. Later, after the PCPP queries are revealed, the prover sends the paths of the trees corresponding to the PCPP queries.

Finally, recall that in an extendable Merkle tree (Sec. B.1), the label of a node is represented as a vector of VSS shares, and a proof of consistency of a node consists of a vector of MPC-in-the-head views.

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 $\mathsf{S}^{\gamma}_{\mathsf{tree}}[i]$ refers to the share of the i-th player, participating in the VSS for the *label* of node in position γ, in the tree $\mathtt{tree}$. 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 compute for 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 $\mathsf{MPC\text{-}Innode}$, 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 set of PCPP queries for each possible size. Thus, following [GOSV14], in our protocol V asks for sets $\{q_i^j, p_i^j\}_{i \in k}$, where k is the soundness parameter of the PCPP, and $j = 1, \ldots, \ell_d$.

Parameters. Let $\ell_d = \log^2 n$ be the upper bound on the size of a PCPP proof. For every query of the PCPP verifier, the prover P must exhibit a path, i.e., a sequence of nodes. We denote by N the total number of nodes that the prover has to prepare in Step 4 (see Protocol described in Sec. 2) to answer to the queries of the verifier. V sends a set of $2k$ queries for each possible depth of the Machine/PCPP. Therefore, formally $N = \sum_{j=1}^{j=\ell_d} \sum_{i=1}^{i=k} \log q_i^j \cdot \log p_i^j$.

Trapdoor theorem and PCPP. We use PCP of Proximity for the following pair language:

$$\mathcal{L}_{\mathcal{P}} = \{(a = (r, \mathtt{t}), (Y)),\ \exists M \in \{0,1\}^* \text{ s.t. } Y \leftarrow \mathsf{ECC}(M),\ M(z) \rightarrow r \text{ within } \mathtt{t} \text{ steps}\}$$

Functionality $\mathcal{F}_{\mathsf{innode}}$.
Parameter: Node γ, for some query q_i^j, label $\mathsf{L}^{\gamma}_{tree} = \sigma^{\gamma}_{tree}[1], \ldots, \sigma^{\gamma}_{tree}[n]$.
Secret Inputs: views $\{\mathsf{S}^{\gamma}_{tree}[i], \mathsf{S}_{\mathsf{depth}}[i]\}_{i \in [n]}$.

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

Figure 4: The $\mathcal{F}_{\mathsf{innode}}$ functionality.

(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 $a = (r, \mathtt{t})$.

C.1 MPC-in-the-head used

We now define more in details the MPC-in-the-head that are computed by the Simulator/Prover. For each node γ, P executes an MPC protocol (that we define below) on top of the VSS representation of the label of node γ.

Recall that, in the proof, for every node, P prepares a proof to convince the verifier that any internal node either it is "consistent" or that some trapdoor condition holds. For any leaf node, 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 (such properties are used already in [GOSV14]).

- **Node consistency**. For every node γ, over a path of length j, P wants to prove that either $j \neq \mathsf{depth}$, where depth is the size of the real tree committed; or that the $j = \mathsf{depth}$ *and* the innode property (see Def. 11) for node γ is satisfied.

  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 correct, perfectly t private (n, t) MPC protocol for the functionality $\mathcal{F}_{\mathsf{innode}}$ depicted in Fig. 4.

- **PCPP Verifier acceptance**. For every queries $(q_i^j, p_i^j)_{i \in [k]}$, 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 $(q_i^j, p_i^j)_{i \in [k]}$.

  P proves this property by running in its head protocol $\mathsf{MPC\text{-}PCPVer}$. Protocol $\mathsf{MPC\text{-}PCPVer}$ is a perfectly t correct, perfectly t private (n, t) MPC protocol for the functionality $\mathcal{F}_{\mathsf{VerPCPP}}$ depicted in Fig. 5.

Functionality $\mathcal{F}_{\mathsf{VerPCPP}}$.
Parameters: Queries q_i^j, p_i^j, a.
Secret inputs: $\mathsf{S}_M^{q_i^j}[i]$, $\mathsf{S}_\pi^{p_i^j}[i]$, $\mathsf{S}_{d\pi}[i]$.

Functionality:

1. for $i = 1$ to n: obtain the secret input $\mathsf{S}_M^{q_i^j}[i]$, $\mathsf{S}_\pi^{p_i^j}[i]$, $\mathsf{S}_{d\pi}[i]$.
2. 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.
3. run $m_i^j = \mathtt{Recon}(\mathsf{S}_M^{q_i^j}[1], \ldots, \mathsf{S}_M^{q_i^j}[n])$ 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.
4. Else output 0 and halt.

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

C.2 Our "public-coin" ZK Protocol

In this section we describe the protocol of Section 2 in greater details. This protocol will be "public-coin" except for the signature slots. Let $(\mathsf{Com}^{\mathsf{x}}, \mathsf{EQCom}^{\mathsf{x}})$ be an instance-dependent equivocal commitment as defined 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 nstance-dependent equivocal commitment $(\mathsf{Com}^{\mathsf{x}}, \mathsf{EQCom}^{\mathsf{x}})$ (as shown in Fig. 3). Let $T \stackrel{def}{=}$ polylog$n$, let $\ell_d = \log^2 n$, and v be the size of a view of a VSS player. 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.

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 $(x, w) \in \mathbf{R}_L$.

V_0: $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{Gen}(1^n)$. Send vk to P.

Stage One (Signature Slot 1):

P_1: For all $i \in [\mathsf{T}]$, $c_i \leftarrow \mathsf{Com}^{\mathsf{x}}(0^v, \tau_i)$. Send $c_1, \ldots, c_{\mathsf{T}}$ to V.

V_1: Send $\sigma_1, \ldots, \sigma_T$ to P where $\sigma_i = \mathsf{Sign}_{\mathsf{sk}}(c_i)$. P aborts if any of the σ_i's is not a valid signature of c_i.

Stage Two (Trapdoor theorem):

P_2: **Equivocal commitments to the root of the tree of the machine and its depth.** Send $\mathsf{tc}_M^0[i] = \mathsf{EQCom}^{\mathsf{x}}(\cdot)$ (root's left child); $\mathsf{tc}_M^1[i] = \mathsf{EQCom}^{\mathsf{x}}(\cdot)$ (root's right child), $\mathsf{tc}_{dM}[i] = \mathsf{EQCom}^{\mathsf{x}}(\cdot)$ (depth of the tree) for $i \in [n]$ to V.
Set $L_M^\lambda = \{\mathsf{tc}_M^b[i]\}_{i \in [n], b \in \{0,1\}}$ as the root of the tree of the machine.

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

Stage Three (Proof):

P_3: **Equivocal commitments to the root of the tree of the PCPP proof and its depth.**
Send $\mathsf{tc}^0_\pi[i] = \mathsf{EQCom}^{\mathsf{x}}(\cdot)$ (root's left child); $\mathsf{tc}^1_\pi[i] = \mathsf{EQCom}^{\mathsf{x}}(\cdot)$ (root's right child), $\mathsf{tc}_{d\pi}[i] = \mathsf{EQCom}^{\mathsf{x}}(\cdot)$ (depth of the tree) for $i \in [n]$. Let $L^\lambda_\pi = \{\mathsf{tc}^b_\pi[i]\}_{i\in[n],b\in\{0,1\}}$ be the root of the tree of the PCP proof.

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: $(q^j_i, p^j_i) = \mathsf{Q}_{\mathsf{pcpx}}(a, r_j, i)$ with $i \in [k]$ (k is the soundness parameter for the PCPP).

P_4: **Commitments to fake nodes**. Let N be the number of nodes required to answer to the PCPP queries $\{q^j_i, p^j_i\}_{j\in[\ell_d],i\in[k]}$. For every node γ to be shown on the paths of the tree of the machine, and for every node β to be shown on the paths of the tree of the PCPP proof, P sends: $\mathsf{tc}^\gamma_M[i] = \mathsf{EQCom}^{\mathsf{x}}(\cdot)$, $\mathsf{tc}^\beta_\pi[i] = \mathsf{EQCom}^{\mathsf{x}}(\cdot)$ to V.

V_4: Sign every commitment for P. Send $\sigma^\gamma_M = \mathsf{Sign}(\mathsf{sk}, \mathsf{tc}^\gamma_M[i])$ and $\sigma^\beta_\pi = \mathsf{Sign}(\mathsf{sk}, \mathsf{tc}^\beta_\pi[i])$.

P_5: **Commitment to the fake MPC-in-the-head that prove consistency/PCPP acceptance.** For every node γ, P prepares commitments to views of the MPC-in-the-head $\mathsf{MPC\text{-}Innode}$: P sends $\mathsf{tc}^\gamma_{\mathsf{in}}[p] = \mathsf{ExCom}(\cdot)$ to V, with $p \in [n]$.

For the nodes in position (q^j_i, p^j_i), which represent the bits of the machine/PCP queried by V for level j, P sends commitment to the views of the MPC-in-the-head $\mathsf{MPC\text{-}PCPVer}$: $\mathsf{tc}^{j,i}_{\mathsf{PCPVer}}[p] = \mathsf{ExCom}(\cdot)$, with $p \in [n]$. Where ExCom is an extractable and equivocal commitment scheme.

V_5: Randomly choose t players: $p_1, \ldots, p_t$ and send them to P.

P_6: **Equivocation using the witness as trapdoor**.

1. Equivocate views of $\mathsf{MPC\text{-}Innode}$.
For each node γ, β computed above:
1) Compute valid shares of 0 for the VSS of depths. Namely run:
$\{\mathsf{S}_{dM}[1], \ldots, \mathsf{S}_{dM}[n]\} \leftarrow \mathtt{Share}(0^n)$ and $\{\mathsf{S}_{d\pi}[1], \ldots, \mathsf{S}_{d\pi}[n]\} \leftarrow \mathtt{Share}(0^n)$.
2) Compute valid shares of 0 for each node γ on the tree of the machine (resp. β for the tree of the PCPP) that was revealed as follows.

$$\{\mathsf{S}^\gamma_M[1], \ldots, \mathsf{S}^\gamma_M[n]\} \leftarrow \mathtt{Share}(0^n); \{\mathsf{S}^\beta_\pi[1], \ldots, \mathsf{S}^\beta_\pi[n]\} \leftarrow \mathtt{Share}(0^n)$$

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

$$\mathsf{Adapt}(x, w, \tau, \mathsf{tc}^\gamma_M[p_j], \mathsf{S}^\gamma_M[p_j])$$

with $j \in [t]$, for any node γ in the tree of the machine (resp., $\mathsf{Adapt}(x, w, \tau, \mathsf{tc}^\beta_\pi[p_j]$,$\mathsf{S}^\beta_\pi[p_j]$ for any β on the tree of the PCPP proof).
4) Compute accepting views for players $p_1, \ldots, p_t$ by using the Semi-honest MPC Simulator SimMpc associated to the MPC protocols. For every γ, β along the paths for queries q^j_i, p^j_i: Run

$$\mathsf{SimMpc}(p_1, \ldots, p_t, \gamma, q^j_i, \mathsf{S}^\gamma_M[p], \mathsf{S}_{dM}[p], \sigma^\gamma_M[p], 1)$$

and obtains views

$$P^\gamma_{\mathsf{in}}[p_1], \ldots, P^\gamma_{\mathsf{in}}[p_t]$$

(resp., for β, use share $\mathsf{S}_{\pi}^{\beta}[p]$, β, p_i^j, $\mathsf{S}_{d\pi}$ and $\sigma_{\pi}^{\beta}[p_j]$ and obtain views $\{P_{\mathsf{in}}^{\gamma}[p_j]\}_{j\in[t]}$).
5) Equivocate the commitments for the MPC views in position p_j so that they open to the above generated views as follows:

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

with $j \in [t]$ (resp., for node β).

2. Equivocate views for $\mathsf{MPC\text{-}PCPVer}$ on public input $a = (r, \mathtt{t})$. For each set of queries $(q_1^j, p_1^j, \ldots, q_k^j, p_k^j)$ asked for level j:

   Run $\mathsf{SimMpc}(p_1, \ldots, p_t, \mathsf{S}_M^{q_i^j}[p_1], \mathsf{S}_{\pi}^{p_i^j}[p_1], \ldots, \mathsf{S}_M^{q_i^j}[p_t], \mathsf{S}_{\pi}^{p_i^j}[p_t], 1)$ for $i \in [k]$, and obtain views $P_{\mathsf{PCPVer}}^{j}[p_1], \ldots, P_{\mathsf{PCPVer}}^{j}[p_t]$.
   Equivocate commitments for views of $\mathsf{MPC\text{-}PCPVer}$ in position p by running: $\mathsf{Adapt}(x, w, \tau, \mathsf{tc}_{\mathsf{PCPVer}}^{j,i}[p], P_{\mathsf{in}}^{j,i}[p])$, with $p \in [t]$.

V_6: Verification. For $p \in p_1, \ldots, p_t$ performs checks:

1. Node consistency.
   For every node γ, β, for every player p check that the view $P_{\mathsf{in}}^{\gamma}[p]$ (resp. β) is **consistent**, has in **input** the value $\mathsf{S}_M^{\gamma}[p], \mathsf{S}_{dM}[p]$ (resp., $\mathsf{S}_{\pi}^{\beta}[p], \mathsf{S}_{d\pi}[p]$) and in **output** 1.
2. PCPP verifier acceptance. For every set of nodes $(q_i^j, p_i^j)_{i\in[k]}$, check that the view $P_{\mathsf{PCPVer}}^{j}[p]$ is **consistent**, has in **input** $\mathsf{S}_M^{q_i^j}[p]$, $\mathsf{S}_{\pi}^{p_i^j}[p], \mathsf{S}_{d\pi}[p]$ (for $i \in [k]$) and the **output** is 1.

D Security Proof

In this section we provide the formal proof of security of our protocol. We begin with a protocol that is a (semi) black-box construction of a zero-knowledge argument of knowledge that is almost public-coin. Every message except the signature slots will be public-coin. Finally we argue that applying the BGGL transformation to this protocol will yield a resettably-sound zero-knowledge argument of knowledge.

Completeness of (P, V) follows directly from simulatability of semi-honest adversary in the MPC protocol.

Soundness. We argue soundness of our protocol by breaking the unforgeability of the underlying signature scheme. Our proof essentially follows from Barak and Goldreich [BG08]. More precisely, we will show that if there exists non-uniform PPT adversary P^* and $x \in \{0,1\}^n \backslash L$ such that P^* convinces V on input x with probability $\frac{1}{p(n)}$ there exists an oracle algorithm CF that violates the unforgeability of the underlying signature scheme given oracle access to the signing oracle.

We show how to construct CF using P^*. First we note that since $x \notin L$, all the commitments made with $\mathsf{Com}^{\mathsf{x}}$ by P^* have to be statistically binding. In particular this means that the length of the code and the running times committed to by the prover are fixed and there has to exist an $i^* \in [\ell_d]$ corresponding to this length and running-time such that MPC functionality corresponding to the nodes in the paths obtained from query r_{i^*} can only be satisfied by providing all parties with VSS shares that reconstruct the value of the node for internal nodes and VSS shares for values for leaves that is the convincing response for the PCPP verifier. Furthermore, the t-correctness of the MPC-in-the-head protocol and statistically-binding property of the commitments using $\mathsf{Com}^{\mathsf{x}}$ ensures that P^* can only commit to correct VSS shares corresponding to these nodes in P_4.

This means that for independently drawn PCPP-queries in V_3, either the proofs are the same or different, i.e. P^* replies consistently or not. If it does not reply consistently with non-negligible probability over the PCPP-queries of the verifier, then we know that we can find a collision to the underlying signature scheme since we have already established that the VSS shares are consistent with the label of the node and the root commitments in P_3 are the same for independent PCPP queries.

If the PCPP responses are consistent, we look at two executions for independently drawn r and $\tilde{r}$ by the verifier in V_2. Now, it must be the case that with overwhelming property the underlying proofs Π and $\tilde{\Pi}$, circuit descriptions M and $\tilde{M}$ must be different. Since we use error-correcting codes ECC with constant min-distance, if $\Pi \neq \tilde{\Pi}$, then for a randomly chosen i, the i^{th} bit in the description of M and $\tilde{M}$ are different with constant probability. Again, since the VSS shares corresponding to every node in the Signature Tree corresponding to M and $\tilde{M}$ are consistent with the label of the node and the root commitment in P_1 are the same, it must be the case that P^* can find a collision in the signature scheme. We remark that the views of the MPC-in-the-head players committed in P_5 contain as input all the information regarding the nodes, in particular, the signatures and the VSS views corresponding to the current node. Since this commitment is an extractable commitment scheme (and the prover cannot equivocate since $x \notin L$), we can extract all values from leaf to the root.

Since the underlying signature scheme is unforgeable, we can construct CF that with a signing oracle internally emulates P^* on input x feeding the signatures requested by P^* by forwarding it to the signing oracle. Since CF is guaranteed to find a collision with non-negligible probability either by forking in message V_3 with independent PCPP queries or forking in message V_2 with independent random strings, CF breaks the unforgeability of the signature scheme. This concludes the proof of soundness.

Argument of Knowledge. Proving that this protocol is an argument of knowledge essentially relies on [GOSV14, CPS13, BG02]. The idea here is that weak argument of knowledge property (introduced by [BG08]) of universal arguments says that we can extract each bit of the PCP proof as long as the prover cannot find collisions. We implicitly have a Merkle Tree that serves as the universal argument. Suppose we have a non-uniform PPT prover P^* that for infinitely many n, convince a verifier on statement y sampled by the prover with probability $\frac{1}{p}$ for some polynomial p, then we can extract every bit of the witness used with non-negligible probability. We provide a proof sketch next.

Suppose P^* equivocates any commitment for random continuations from messages P_2, P_3, P_4 or P_5 in a random execution then we can immediately obtain the witness using the trapdoor property of our commitments. Second we can assume that since P^* is efficient it cannot violate the unforgeability of the underlying signature scheme. Implicit in Lemma 3.5 [BG02] is an extractor algorithm that can extract the proof bit by bit from the Merkle hash tree. Following the same approach we can extract the PCPP proof if P^* never equivocates or violates unforgeability and this will be a contradiction just as in [BG02]. We remark that, the subtle issue here is that we require that the extractor be able to open all the VSS shares in the leaf nodes (as opposed to just t of them) to learn the value of the leaves. This is possible since the views of the MPC-in-the-head players are committed to using an extractable commitments scheme in P_5 and the views together contain all the VSS shares required.

Zero-Knowledge. The zero-knowledge simulator starts by honestly emulating the signature slot for the malicious verifier V^*. V^* provides accepting signatures, the simulator sets up a procedure to obtain signatures for commitments of arbitrary messages. In [CPS13], they achieve this by appropriately rewinding the malicious verifier during the signature slot. The verifier may not always answer, but by "rewinding" an appropriate number of times and sending fresh commitments each time [CPS13], a Goldreich-Kahan type argument guarantees that the simulator will succeed with very high probability. More precisely, they show that if V^* did provide a valid signature during the first pass, then in expectation, by the hiding property of the commitment scheme, a polynomial number of rewindings will be enough. However this approach will not work here due to the second signature slot. Looking ahed, that the simulation strategy will be to first obtain the required signatures from the first slot and then repeat the signature queries corresponding to the PCPP queries in the second slot. To prevent the verifier from correlating the aborting probability across the two slots, we need to ensure that the distribution of the messages in the second slot are "uniformly" generated. Towards this we consider the first signature slot that provides not just one signature but polynomially many signatures. Then using a simple amplification via parallel-repetition argument we can show that for most random tapes τ we can obtain a signature for $c = \mathsf{Com}^{\mathsf{x}}(m, \tau)$ a commiment to message m. Next, we show how to modify the distribution of the messages in the second signature slot to achieve our result.

More precisely, Sim internally emulates the code of V^* and proceeds as follows:

Verifier move V_0 **:** Sim receives vk from V^*.

Prover move P_1**:** Sim generates and sends to V^* honest commitments to 0^v with uniformly chosen randomness just as the honest prover.

Verifier move V_1**:** V^* either aborts or produces signatures for all T commitments.

- If V^* aborts or any of the signatures are invalid, 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 $\mathsf{GetSign}$ (analogous to the sig-com oracle in [CPS13]) that will return a signature for any commitment c to message m.

  Procedure $\mathsf{GetSign}(c, m)$**:** c is a commitment to message $m \in \{0,1\}^v$

  - For $i = 1$ to $2ntT^2$

    Select a random i and $T-1$ uniformly random commitments to m, say $c_1, \ldots, c_{i-1}$, $c_{i+1}, \ldots c_T$, set $c_i = c$ and query V^* at the first Signing Slot with $c_1, \ldots, c_T$. If V^* replies a valid signature σ for c_i, then O outputs σ.
  - If O never output anything, then return $\bot$.

FORLOOP: For $\mathsf{trials} = 1$ **to** $n \log^2 n$

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 $\mathsf{no\text{-}op}$'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 Signature Tree for $\mathsf{ECC}(V^{**})$. To create the Signature Tree, the simulator needs to obtain signatures of commitments to actual messages. For any message m, Sim obtains the signature of m as follows:

- Sim makes T calls to $\mathsf{GetSign}(c, m)$ each time with a random commitment c to m. If any of the T trials succeed then it has obtained a valid signature of a commitment to m. Otherwise Sim aborts outputting fail_2.

Then Sim commits to the VSS shares of the left and right child of the root of this Signature Tree and the shares obtained by running VSS on input $\mathsf{tc}_{dM} = log|V^*| + i$

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

Prover move P_3: Create a Signature Tree for the PCPP proof that V^{**} predicts r. To build this tree, the simulator needs to obtain signatures of commitments to string and uses the same idea as in the Prover-move P_2. Then Sim commits to the VSS shares of the left and right child of the root of this Signature Tree. Additionally, Sim commits to the depth $\log|V^*| + j$ using VSS.

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 $i \in [k]$.

Prover move P_4: V expects to see a sequence of nodes, for paths defined by queries q_i^j, p_i^j. As each node is a sequence of VSS share, let N be the total number of shares that Sim must prepares for V.

We know that there exists a $j^* \in [\ell_d]$ 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. For the queries $(q_i^{j^*}, p_i^{j^*})_{i \in [k]}$, Sim needs to provide a sequence of nodes which are consistent with the tree committed in steps P_2, P_3. Let us denote by VS the set of indexes of the commitments that must be taken from the real tree in order to consistently answer to the queries $(q_i^{j^*}, p_i^{j^*})_{i \in [k]}$. Let us denote by c a commitment that is part of the real tree.

For the remaining queries $(q_i^{\kappa}, p_i^{\kappa})_{i \in [k]}$, with $\kappa \neq j^*$, Sim freshly generates nodes by computing VSS of 0. Let us denote by c' a commitment that is freshly generated.

Now, let us consider $c'_1, \ldots, c'_N$ as a sequence of all *freshly* generated commitments.

Sim proceeds as follows.

- **CHECK GOOD:** For every $i \in \mathsf{VS}$ run $\mathsf{GetSign}(c'_i, s'_i)$ and see if it returns a valid signature. Even if there exists one index on which $\mathsf{GetSign}$ did not return a signature, then we say this trial fails and return to **FORLOOP** just before P_2.

Otherwise, replace c'_i with c_i and feed $c'_1, \ldots, c'_n$ to V^*. Exit **FORLOOP**.

Verifier move V_4: V^* either aborts or produces signatures for all T commitments. If V^* aborts or supplies an invalid share, Sim halts outputting the transcript upto that point.

Prover move P_5: 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. 4. 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 $\mathcal{F}_{\mathsf{VerPCPP}}$.

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 $1 - p$, 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 $H_0, H_1, \ldots, H_8$; starting from H_0, the view of the verifier V^* in the real interaction with the honest prover to H_8, 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 Sim. The slight modification here is that instead of using $\mathsf{Com}^{\mathsf{x}}$ the hybrid simulator will use

$\mathsf{EQCom}^{\mathsf{x}}$ 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 $\mathsf{EQCom}^{\mathsf{x}}$. 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. In essence, this follows using a standard Yao-type parallel- repetition theorem, where we can show that if an adversary can answer T repetitions with probability $1/p$ for some polynomial p then we can answer one repetition with probability $1 - n/T$. 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 tree generated uses $\mathsf{EQCom}^{\mathsf{x}}$ instead of $\mathsf{Com}^{\mathsf{x}}$. It follows exactly as before that the view output in H_3 and H_2 are statistically-close.

Hybrid H_4: This experiment proceeds exactly as H_3 with the exception that P_4 is generated according to Sim and as before all commitments for this message are generated using $\mathsf{EQCom}^{\mathsf{x}}$ instead of $\mathsf{Com}^{\mathsf{x}}$. More precisely, in step P_4, the hybrid simulator will sample random commitments of 0^v using $\mathsf{EQCom}^{\mathsf{x}}$ and then check if the commitments corresponding to relevant paths are "good", i.e. passes the CHECK. If it does, then the hybrid simulator replaces these commitments with the actual commitments used to generate the trees in step P_2 and P_3. To argue that the view output by H_4 is statistically close to H_3 we will rely on the fact that the condition for using a commitment to generate the Merkle Hash tree is the same condition for replacing the commitments in message P_4. More precisely, in Claim 4, we show that conditioned on not outputting fail_3 the distributions output by the hybrid simulator in H_4 and H_3 are identically distributed. Then in Claim 5 we show that probability that the simulator outputs fail_3 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 will follow that the view output by this hybrid is distributed identically to the previous hybrid.

We remark that in this hybrid for random continuations from P_2 as well as P_4, the values revealed for any commitment made using $\mathsf{EQCom}^{\mathsf{x}}$ is unique. This is because we use the Sim's strategy to assign values to commitments and Sim never equivocates.

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 $4t(n)/p$ steps where p 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 $\frac{p}{2}$.Now we use the truncated experiments as they run in strict polynomial time. This essentially follows the same argument used in [DSMRV13] and relies on the fact the knowing the witness we can generate a commitment to 0 and 1 from a distribution of commitments to 0 and commitments to 1.

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

Proof. First, the computational hiding property of $\mathsf{Com}^{\mathrm{x}}$ implies that there exists some negligible $\nu(\cdot)$ such that $|p(m) - p| \leq \nu(n)$ for every $m \in \{0,1\}^l$. Now we consider two cases. If $p \leq 2\nu$, then the indistinguishability trivially holds since the interaction aborts at the end of the Signature Slot (in this case, the view is perfectly simulated) with all but negligible probability. On the other hand, if $p \geq 2\nu$, we show that Sim obtains a signature for any message m with high probability and the distribution of the prover message P_4 is indistinguishable.

Recall that the simulator makes T calls to $\mathsf{GetSign}$ to obtain a signature to m. To see that the hybrid simulator obtains a signature with high probability, 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 $t \geq n/2p(m)$. 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 $\tau \in G_n$, the probability that $\mathsf{GetSign}$ fails to return a signature is e^{-n} since $\mathsf{GetSign}$ tries $2ntT^2$ times. It only remains to argue that G_n contains sufficiently many tapes τ and this follows using a standard Yao-amplification type proof. More precisely, if we can show that the probability that a random τ is in G_n with probability at least $1 - \frac{n}{T}$, then since Sim makes T attempts, the probability it misses a τ from G_n is at most e^{-n}. Overall the probability that Sim fails to obtain a signature is at most $2^{-O(n)}$ and this concludes the proof of the claim.

The only thing remaining to argue is 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 $\frac{2}{t} + e^{-n} < \frac{1}{t}$ which is a contradiction. Therefore G_n must contain at least $1 - \frac{n}{T}$ fraction of the tapes □

Claim 4. *The distribution of message P_4 in hybrids H_4 and H_3 are identicaly distributed conditioned on the hybrid simulator not outputting fail_3.*

Proof. Here we argue that the message P_4 is identically distributed in H_3 and H_4 conditioned on the hybrid simulator not We first note that all commitments except the ones in the relevant paths are independently and identically distributed. For the relevant paths, we replace them with the commitments obtained from before, only if the current commitment for the relevant paths passes the CHECK. On a high-level this does not affect the distribution because the condition to select a commitment for generating the Merkle Hash Tree is the same as the condition to choose the commitments to replace in P_4. More precisely, this condition is that, a commitment c for message m using a particular random tape τ is selected only if $\mathsf{GetSign}(c, m)$ procedure returns a signature for c. We now prove this formally.

Among the N commitments sent in P_4 let $R \subset [N]$ contain the indices for relevant commitments and $IR \subset [N]$ contain the indices for irrelevant commitments. Let $\vec{c}_R$ and $\vec{c}_{IR}$ be length $|R|$ and $|IR|$ vectors of commitments respectively. We want to prove that conditioned on not outputting fail_3, the probability that the hybrid simulators in H_3 and H_4 send $\vec{c} = (\vec{c}_R, \vec{c}_{IR})$ as P_4 are identical.

For a commitment c, let X_c denote the random variable representing if $\mathsf{GetSign}$ produced a signature to c in a single call to the procedure. Let C_i denote r.v. representing the commitment in index i in H_3.

The probability that the simulator in H_3 sends $\vec{c}$ as P_4 is

$$\Pr[\vec{C} = \vec{c}\ |\neg\mathsf{fail}_3] = \Pr[\vec{C} = \vec{c} \mid X_{C_i}\ \forall i \in R] = \Pr[(\vec{C}_R, \vec{C}_{IR}) = (\vec{c}_R, \vec{c}_{IR}) \mid X_{C_i}\ \forall i \in R]$$

Let C'_i be the r.v. representing the commitments first selected and C^*_i be the relevant commitments obtained using Signature Slot 1. The probability that the simulator in H_4 sends $\vec{c}$ is

$$\Pr[(\vec{C}^*_R, \vec{C'}_{IR}) = (\vec{c}_R, \vec{c}_{IR}) \mid X_{C'_i}\ \forall i \in R\ \wedge\ X_{C^*_i}\ \forall i \in R] = \Pr[(\vec{C}^*_R, \vec{C'}_{IR}) = (\vec{c}_R, \vec{c}_{IR}) \mid X_{C^*_i}\ \forall i \in R]$$

since the commitments are sampled independently. We now conclude that the probabilities are identical because the distribution of C^*_i and C_i are identical and independent of C'_i.

□

Claim 5. *The probability that the simulator outputs* fail_3 *in* H_4 *is negligible.*

Proof. Recall that we rewind $n \log^2 n$ times from P_2. If we show that the probability a random continuation from P_2 passes the CHECK is at least $\frac{1}{\log^2 n}$ then it follows that fail_3 will be output only with probability at most e^{-n}. Recall that the distributions of messages from P_2 to P_4 do not reveal the choices i and j made in hybrid H_3. From Claim 4 above, we can conclude that the same holds in H_4. We will show that for any random continuation from P_2 with high probability there exists a $i^*, j^* \in [\log n]$ such that if $i = i^*$ and $j = j^*$ then the CHECK passes. Since the messages do not reveal the choices, this happens with probability at least $\frac{1}{\log^2 n}$.

To show the existence of i^*, j^* we first show that for any particular i (analogously any j) the probability that $\mathsf{GetSign}$ fails to return the signature for some commitment corresponding to the path in the Merkle Hash Tree for length $|V^*| + i$ (analogously proof tree for running time $t_{V^*} + j$) is at most $\frac{npolylog}{T} \leq \frac{1}{n}$. Therefore, the probability that all pairs i, j fail to pass CHECK is negligible. □

It suffices to argue that in one of the n^3 trials CHK returns true. This is because the probability that $\mathsf{GetSign}$ fails to return a signature for a random commitment to a message is at most $\frac{n}{T}$ (since this happens only if the chosen random tape τ lies outside G_n). Using an union bound, the probability that CHK fails is at most $\frac{n|S|}{T} < \frac{1}{n}$ (by setting T appropriately). This means that among the n^2 choices, the set of commitments corresponding to one of the depths must pass the CHK with high probability and the probability that Sim picked that depth when simulating message P_2 is at least $\frac{1}{\log^2 n}$. Since Sim rewinds at least $n \log^2 n$ times, the probability that the Sim outputs fail_3 is at most e^{-n}.

Obtaining Resettable-soundness We can obtain resettable-soundness by applying the BGGL transformation to the protocol presented above. Just as in [CPS13], our new protocol $(\tilde{P}, \tilde{V})$ will proceed exactly like (P, V) described in the previous section, with the exception that the verifier will initially pick a seed for a pseudo-random function family (PRF), then apply the PRF on the partial transcript to generate each verifier message except the messages in the signature slot. Arguing that the protocol is a resettably-sound zero-knowledge argument of knowledge essentially follows from [BGGL01]. Therefore, we have 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.*