Founding Secure Computation on Blockchains

Arka Rai Choudhuri[1][0000−0003−0452−3426], Vipul Goyal[2], and Abhishek Jain[1]

[1] Johns Hopkins University, Baltimore, USA
{achoud,abhishek}@cs.jhu.edu
[2] Carnegie Mellon University, Pittsburgh, USA
goyal@cs.cmu.edu

Abstract. We study the foundations of secure computation in the *blockchain-hybrid model*, where a blockchain – modeled as a *global* functionality – is available as an Oracle to all the participants of a cryptographic protocol. We demonstrate both destructive and constructive applications of blockchains:

- We show that classical rewinding-based simulation techniques used in many security proofs fail against *blockchain-active* adversaries that have read and post access to a global blockchain. In particular, we show that zero-knowledge (ZK) proofs with black-box simulation are impossible against blockchain-active adversaries.
- Nevertheless, we show that achieving security against blockchain-active adversaries is possible if the honest parties are also blockchain active. We construct an $\omega(1)$-round ZK protocol with black-box simulation. We show that this result is tight by proving the impossibility of constant-round ZK with black-box simulation.
- Finally, we demonstrate a novel application of blockchains to overcome the known impossibility results for concurrent secure computation in the plain model. We construct a concurrent self-composable secure computation protocol for general functionalities in the blockchain-hybrid model based on standard cryptographic assumptions.

We develop a suite of techniques for constructing secure protocols in the blockchain-hybrid model that we hope will find applications to future research in this area.

1 Introduction

Blockchain is an exciting new technology which is having a profound impact on the world of cryptography. Blockchains provide both: new applications of existing cryptographic primitives (such as hash function, or zero-knowledge proofs), as well as, novel foundations on which new cryptographic primitives can be realized (such as fair-secure computation [2, 10, 24], or, one-time programs [36]). In this work, we seek to examine the foundations of secure computation protocols in the context of blockchains. More concretely, we study what we call the *blockchain-hybrid model* and examine constructions of zero-knowledge and secure computation in this model.

The Blockchain-Hybrid Model. In order to facilitate the use of blockchains in secure computation, we study the blockchain-hybrid model, where the blockchain – modeled as a *global* ledger functionality – is available to all the participants of a cryptographic protocol. The parties can access the blockchain by posting and reading content, but no single party has any control over the blockchain. Our modeling follows previous elegant works on formalizing the blockchain functionality [47, 4, 3]. In particular, our model is based on the global blockchain ledger model from Badertscher et. al [4].

We study simulation-based security in the blockchain-hybrid model. In our model, *the simulator does not have any control over the blockchain*, and simply treats it as an oracle just like protocol participants. Thus, unlike traditional trusted setup models such as common reference string, the blockchain-hybrid model does not provide any new "power" to the simulator. In particular, the simulator is restricted to its plain model capabilities such as resetting the adversary or using knowledge of its code. Thus, in our model, the blockchain can be *global*, in that it can be used by multiple different protocols at the same time. This is reminiscent of simulation in the global UC framework [15, 18, 43]. A related model is the global Random Oracle model [18] where the simulator can only observe the queries made by the adversary to the random oracle, but cannot program the random oracle (since it is global and therefore shared across many protocols).

Secure Computation based on Blockchains. We study the foundations of secure computation in the presence of the global blockchain functionality. Interestingly, we demonstrate both destructive and constructive applications of blockchains to cryptography. Primitives which were earlier possible to realize now become impossible. At the same, working in this model allows us to overcome previously established deep impossibility results in cryptography. Interestingly, we also utilize mining delays – typically viewed as a negative feature of blockchains – for constructive purposes in this work. Our main results as discussed next.

1.1 Our Results

Simulation Failure in the Presence of Blockchains. We consider a new class of adversaries that we refer to as *blockchain-active adversaries*. These adversaries are similar to usual cryptographic adversaries, except that they have user access to a blockchain, i.e., they can post on the blockchain and read its state at any point.

We observe that such adversaries can foil many existing simulation techniques that are used for proving security of standard cryptographic schemes. To illustrate the main idea, let us consider rewinding-based black-box simulation techniques that are used, e.g., in zero-knowledge (ZK) proofs [35], secure multiparty computation [59, 34], and signature schemes in the random oracle model constructed via the Fiat-Shamir heuristic [29]. A crucial requirement for the success of rewinding-based simulation is that the adversary should be *oblivious* to the rewinding. Usually, this requirement can be easily met since the simulator

can simply "reset" the code of the adversary, which prevents it from keeping state across the rewindings.

A blockchain-active adversary, however, can periodically post on the blockchain and use it to maintain state across rewindings, and therefore detect that it is being rewound. In this case, the adversary can simply abort and therefore fail the simulation process.[3] It is not too difficult to turn the above idea into a formal impossibility result for ZK proofs against blockchain-active adversaries, when the simulation is required to be *black-box*.

Theorem 1 (Informal). *There does not exist an interactive argument in the plain model which is zero-knowledge w.r.t. black-box simulation against blockchain-active adversaries.*

The above impossibility result extends to secure multiparty computation and other natural cryptographic primitives whose security is proven via a rewinding simulator.

Constructing Zero-Knowledge Protocols. To overcome the above problems posed by blockchains, we look towards blockchains for a solution as well. Our idea is to make the *protocol* blockchain active as well. That is, in addition to the adversary, the honest parties would have access to the blockchain as well.

Our first positive result is an $\omega(1)$-round ZK proof system in the blockchain-hybrid model whose security is proven w.r.t. black-box simulation.

Theorem 2 (Informal). *Assuming collision-resistant hash functions, there exists an $\omega(1)$-round ZK proof system in the blockchain-hybrid model w.r.t. black-box simulation.*

Interestingly, in our construction, *the honest parties do not post any message on the blockchains*. Instead, they only keep a "tab" on the current state of the blockchain in order to decide whether or not to continue the protocol.

We also show that the above result is tight. Namely, we show that using black-box simulation, constant-round ZK is impossible in the blockchain-hybrid model.

Theorem 3 (Informal). *Assuming one-way functions, there does not exist an $O(1)$-round ZK argument system in the blockchain-hybrid model w.r.t. a (expected probabilistic polynomial time) black-box simulator.*

This is in sharp contrast to the plain model where there are a number of classical constant round zero-knowledge protocols that are proven secure w.r.t. a black-box simulator [33, 28, 9].

Concurrent Secure Computation using Blockchains. Classical secure computation protocols such as [59, 34] only achieve "stand-alone" security, and fail

[3] This is reminiscent to the problems that arise in the context of UC security, where the adversary cannot be rewound since it can communicate with an external environment, leading to broad impossibility results for zero-knowledge and secure computation [14, 16, 19].

in the setting of *concurrent self-composition*, where multiple copies of a protocol may be executed concurrently, under the control of an adversary. In fact, achieving concurrent secure computation in the plain model has been shown to be impossible [13, 50, 51, 52, 8, 37, 1, 32]. The above impossibility results are far reaching and rule out secure computation for a large class of functionalities in a variety of settings.

Interestingly, we show that concurrent self-composition is possible in the blockchain-hybrid model w.r.t. standard real/ideal model notion of security with a PPT simulator. Thus, our results (put together) show that designing cryptographic primitives in the blockchain-hybrid model is, in some sense, harder and easier at the same time.

Theorem 4 (Informal). *Assuming collision-resistant hash functions and oblivious transfer, there exists a concurrent self-composable secure computation protocol for all polynomial-time functionalities in the blockchain-hybrid model.*

In our protocol, each party is required to post an initial message (which corresponds to a commitment to its input and randomness) on the blockchain. However, an honest party can simply perform this posting in an "offline" phase prior to the start of the protocol. In particular, once the protocol starts, an honest party is not required to post any additional message on the blockchain.

A number of prior beautiful works have constructed concurrent (and universally composable) secure computation in various setup models such as the trusted common reference string model [21], the registered public-key model [6], the tamper-proof hardware model [45, 22, 39], and the physically uncloneable functions model [12, 25, 5]. We believe that the blockchain model provides an appealing *decentralized* alternative to these models since there are no physical assumptions or centralized trusted parties involved. Moreover, it allows for basing concurrent security on an already existing and widely used infrastructure. Further, it is possible to obtain strong guarantees of the following form: an adversary who can break our construction can also break the security of the underlying blockchain (potentially allowing it to gain large amounts of cryptocurrency), or the underlying cryptographic assumptions (oblivious transfer and collision-resistant hash functions in our case).

Impossibility of UC Security. While Theorem 4 establishes the feasibility of concurrent self-composition, we show that universal composition security [14] is impossible in the blockchain-hybrid model:

Theorem 5 (Informal). *Universally composable commitments are impossible in the blockchain-hybrid model.*

We prove the above result via a simple adaptation of the impossibility result of [16] to the blockchain-hybrid model. The main intuition behind this result is that a simulator in the blockchain-hybrid model has the same capabilities as in the plain model, namely, the ability to rewind the adversary or using knowledge of its code. Crucially, (unlike the non-programmable random oracle model [18]) the ability to see the queries made to the blockchain do not constitute a new capability for the simulator since *everyone* can see those queries.

1.2 Technical Overview

We start with the observation that if an adversary is blockchain-active, it can "detect" that it is being rewound by posting the transcript of the interaction so far on the blockchain. In more detail, upon getting an incoming message, the adversary concatenates the entire transcript with a session ID and submits it to the blockchain Oracle. Before giving a response, the adversary waits for the next block to be mined and checks the following: the transcript it posted on the blockchain has indeed appeared, and, no such transcript (for the same session and the same round) appeared on any of the prior blocks. If the check passes (which is guaranteed in the real execution), the adversary proceeds honestly with computing and sending the next protocol message. We show that it would be impossible for any polynomial-time simulator to rewind this adversary which forms the basis of our black-box impossibility result for zero-knowledge.

Constructing Black-Box Zero-Knowledge Protocols. To overcome the above problems posed by blockchains, we look towards blockchains for a solution as well. Our idea is to make the *protocol* blockchain active as well. Specifically, we let the honest prover keep track of the blockchain state, and, if the number of new blocks mined since the beginning of the protocol exceed a fixed number k, abort. Thus, the honest parties use the blockchain to implement a time-out mechanism. We emphasize, however, that we do not require the honest parties to have synchronized clocks. The only requirement placed is that the protocol must be finished in an a priori bounded amount of time, *as measured by the progress of the blockchain*. For example, while using Bitcoin, if k is set to 20, this gives the parties nearly 3.5 hours to finish the zero-knowledge protocol before a time-out occurs (since a block is mined roughly every 10 minutes in Bitcoin). For simplicity, we will treat the parameter k as a constant (even though our constructions can handle an arbitrary value of k by scaling the round complexity of the protocol appropriately).

We devise a construction for black-box zero-knowledge proofs where the number of "slots" (or rewinding opportunities) in the protocol is higher than k. While the adversary can send any information to the blockchain Oracle at any point of time, there can be at most k points in the protocol execution where the adversary actually *receives* from the Oracle a new (unforgeable) mined block. However by our construction, this would still leave several slots in the protocol where the simulator is free to rewind (without having to forge the blockchain state).

A potentially complication in the design of the simulator arises from the fact that, apart from the newly mined blocks, the adversary can also "listen in" on the network communication in real time. This could consist of various (honest party) transactions currently outstanding on the network and waiting to be included in the next block. This is formalized by *buffer reads* in the model of Badertscher et. al [4]. We handle this problem by having the simulator simply replay the honest-party outstanding transactions since they could not have changed from the main thread to the look-ahead thread. The adversarial outstanding transactions (which might change from thread to thread) in the current thread are

already known to the simulator since the simulator can read all outgoing messages from the adversary. The above ideas form the basis of our first positive result modulo the issue of simulation time which is discussed next.

The Issue of Simulation Time. Interestingly, the fact that blockchains can be used to implement a global unforgeable clock presents a novel challenge in proving security against blockchain-active adversaries, that to the best of our knowledge, does not arise elsewhere in cryptography. Typically in cryptography, the running time of the simulator is larger than the running time of the adversary. This means that the number of blocks mined during a simulated execution may be higher than the number of blocks mined during a real execution. Then, the number of mined blocks can be used as "side-channel" information to distinguish real and simulated executions, if the adversary and the distinguisher are blockchain-active! Such a difficulty does not arise in the plain model since the simulator is assumed to have complete control over the clock of the adversary (including the ability to freeze it).

To address this issue, we seek to construct a simulator whose running time is the same as the real protocol execution. Towards that end, we build upon techniques from the notion of precise zero-knowledge [53]. To start with, it would seem that we need to construct a simulator with precision exactly 1, something that is currently not known to be possible. To resolve this problem, our key observation is that there is a crucial difference between the *time that the simulator takes to finish* and *the number of computation steps it executes*. In particular, if the simulator can execute a number of computations *in parallel*, it could potentially perform more computations than the prover in the real execution, and yet, finish in the same amount of time. Our rewinding strategy would run several threads of execution in parallel (e.g., by making several copies of the adversary code) and ensure that by the time the *main*[4] thread finishes, all the rewound execution threads have finished as well. To ensure that the simulation succeeds, our simulator is necessarily required to have a super-constant number of rewinding opportunities (which can be pursued in parallel). Such a simulator would give a guarantee of the following form: any information learnt by an adversarial verifier in the protocol could also be produced from scratch by an algorithm which is capable of running sufficient (polynomial) number of computations in parallel. For example, a quad core processor is capable of running 4 parallel computations.

We believe that the issue of simulation time is one of independent interest. In particular, developing an understanding of the time required by the simulator (as opposed to the number of computation steps) could shed additional light on the knowledge complexity of cryptographic constructions as well as motivate the study of strong notions of security.

Lower Bound on Round Complexity of Black-Box Zero-Knowledge. We prove that constant round ZK arguments are impossible w.r.t black-box simulation in the blockchain-hybrid model. Our impossibility result holds even for expected polynomial-time simulators.

[4] The thread output by the simulator is referred to as the main thread.

Consider an adversarial verifier that waits for a fixed constant time c before responding to any message from the prover. Our proof works in two steps:

1. Recall that black-box simulators can only query the adversarial verifier as an Oracle. However, the simulator may choose to make these queries *in parallel* rather than sequentially by making several copies of the adversary state (and hence, increasing the number of available Oracles).
   In the first step, we assume that the simulator is *memory bounded*. This means that at any given time, the simulator may only have a bounded (strict polynomial) number of copies (say) $q(\cdot)$ of the adversary. Furthermore, since the verifier takes time c to answer each query, the total number of queries the simulator may make to the adversary in a given time t can be bounded by $\frac{q \cdot t}{c}$ (an a priori bounded strict polynomial). Now we observe the following:
   - The simulator must terminate within roughly t steps where t is the time an honest prover takes to complete the proof. To see this, let r be an upper bound on the number of blocks that can created in the time taken by the honest prover to complete the proof. We consider a blockchain active adversary that observes the state of the blockchain when the protocol starts, and posts a transcript on the completion of the proof. If it notices that more than r blocks have been created since the protocol started, it concludes that it is interacting with the simulator.
   - Thus, the overall number of queries (and hence) the running time of the simulator is a strict polynomial. Now, we can directly invoke the result of Barak and Lindell [7] that rules out constant-round ZK arguments with strict polynomial-time black-box simulation.
2. The above only rules out a simulator with "a priori bounded parallelism." However what if, e.g., the number of parallel queries the simulator may make to the verifier cannot be a priori bounded (and instead we only require that the simulator finish in a priori bounded number of computational steps)? In particular, the simulator may see the responses to the queries made so far, and, *adaptively* decide to increase the number of parallel queries (i.e., the number of copies of the adversary)? This case is more tricky and as such, the ideas from the work of [7] don't apply.
   To resolve this issue, we crucially rely upon the fact that by carefully choosing the delay parameter c and an aborting probability for the adversary, the number of such "adaptive steps" can be bounded by a constant. Thereafter, we argue that in each adaptive step, if the simulator increases the number of parallel copies by more than an a priori bounded polynomial factor, it runs the risks of blowing the number of computation steps to beyond expected polynomial. On the other hand if the number of parallel copies blow up by at most a fixed polynomial factor, since the number of adaptive steps is a constant, the simulator is still using "bounded parallelism" (a case already covered by our previous step). The full proof is delicate and can be found in the full version.

Concurrent Secure computation. We now proceed to describe the main ideas behind our positive result for concurrent self-composable secure computation.

We start by recalling the intuition behind the impossibility of concurrent secure computation w.r.t. black-box simulation in the plain model.

A primary task of a simulator for a secure computation protocol is to extract the adversary's input. A black-box simulator extracts the input of the adversary by rewinding. However, in the concurrent setting, extracting the input of the adversary in each session is a non-trivial task. In particular, given an adversarial scheduling of the messages of concurrent sessions, it may happen that in order to extract the input of the adversary in a given session s, the simulator rewinds past the beginning of another session s' that is interleaved inside the protocol messages of session s. When this happens, the adversary may change its input in session s'. Thus, the simulator would be forced to query the ideal functionality more than once for the session s'.

Indeed, as shown in [51], this intuition can be formalized to obtain a black-box impossibility result for concurrent self-composition w.r.t. the standard definition of secure computation, where only one query per session is allowed. While Lindell's impossibility result is only w.r.t. black-box simulation, subsequent works have shown impossibility of concurrent secure computation even w.r.t. non-black-box simulation [8, 37, 1, 32].

In order to overcome the impossibility results, our starting idea is the following: prior to the start of a protocol, each party must commit to its input and randomness on the blockchain. It must then wait for its commitment string to be posted on the blockchain before sending any further message in the protocol. Similar to our ZK protocols (with stand-alone security), we use a time-out mechanism to place an upper bound on the number of blocks that can be mined during a session. Then, by using sufficiently many rewinding "slots," we can ensure that there exist some slots in each session where the adversary is guaranteed to not see new block (and hence no new interleaved sessions), making them "safe" for rewinding. Note, however, that this mechanism does not bound the overall number of concurrent sessions since an adversary can start any polynomial number of sessions *in parallel.*

Once we have the above protocol template, the key technical challenge is to perform concurrent extraction of the adversary's inputs in all of the sessions. Since there are multiple "unsafe" rewinding slots in every session (wherever a new block is mined), we need to extract adversary's inputs in all of the sessions under the constraint that only the safe slots are rewound. Unfortunately, commonly known rewinding strategies in the concurrent setting [58, 49, 57] rewind all parts of the protocol transcripts (potentially multiple times). Therefore, they immediately fail in our setting.

In order to solve this problem, we develop a new concurrent rewinding strategy. The starting idea towards developing this rewinding strategy is the observation that our particular setting has some similarities to the work of Goyal et al. [41] who were interested in a seemingly unrelated problem: designing commitment schemes that are secure w.r.t. chosen commitment attacks [20]. Goyal et al. introduced what they call the "robust extraction lemma" that guarantees concurrent extraction even if a *constant* number of "breakpoints" – that cannot be

rewound – are interspersed throughout the overall transcript of the concurrent sessions. These breakpoints are analogous to the unsafe points in our setting.

While this serves as a useful starting point, robust extraction is not directly applicable to our setting since overall, the number of external blocks seen by the adversary (the equivalent of breakpoints in [41]) cannot be bounded. Indeed, if the number of sessions is T, the number of blocks can only be upper bounded by $T \cdot k$ (if e.g., all the sessions are sequential).

Our main observation is that the concurrent adversary can only choose one of the following: either too much concurrency, or too many newly mined blocks, but not both. This allows us to come up with a new variant and analysis of the robust extraction lemma which we believe could be of independent interest. In particular, our new variant uses twice as many slots as the one used by the robust extraction lemma. We refer the reader to the technical sections for more details.

1.3 Related Work

Blockchains and Cryptography. In a recent work, [36] used blockchains to construct non-interactive zero-knowledge (NIZK) arguments and selectively-secure one-time programs. Their model, however, is fundamentally different from ours in that they rely on a much stronger notion of simulation where the simulator controls all the honest miners in the blockchain. Intuitively, this is somewhat similar to the honest majority model used to design (universally composable) secure multiparty computation protocols. Due to the power given to the simulator, their model necessitates the blockchain to be "local" (i.e., private) to the protocol. In contrast, our model allows for the blockchain to be a "global" setup since the simulator has no extra power over the blockchain compared to the adversary. This is similar to the difference between universal composability framework [14] and global universal composability framework [15], where in the former model, a setup (such as a common reference string) cannot be reused by different protocols, whereas in the latter model, a common setup can be used across multiple protocols. Indeed, since the simulator has no additional power except the ability to reset the adversary or use knowledge of its code, NIZKs are impossible in our model, similar to the plain model. Unlike our work, [36] do not consider interactive ZK proofs or any notion of secure multiparty computation.

In another recent work, [24] study the problem of fair multiparty computation in a "bulletin-board" model that can be implemented with blockchains. Similar to [36], however, their model provides the simulator the ability to control the blockchain. Prior to their work, multiple works [2, 10] studied the problem of fairness with penalties using cryptocurrencies.

Several elegant works have conducted a formal study of various properties of blockchains [30, 56, 31, 46, 4]. Most relevant to our work is that of Badertscher et. al [4] whose modeling of the blockchain ideal functionality we closely follow.

Concurrent Security. The study of concurrent security for cryptographic protocols was initiated by Dwork et al. [27] who also introduced a timing model

for constructing concurrent ZK. In this model, the parties have synchronized clocks and are required to insert "delays" at appropriate points in the protocol. A refined version of their model was later considered in [44] for the problem of concurrent secure computation. We note that while our approach to concurrent secure computation in the blockchain-hybrid model appears to bear some similarity to the timing model, there are fundamental differences that separate these models. For example, the simulator can fully control the clock of the adversary in the timing model, while this is not possible in our setting since the blockchains provide an unforgeable clock to the adversary. More importantly, in the timing model, there are no "unsafe" points, and the simulator can rewind anywhere. For this reason, the timing model does not require developing new concurrent extraction techniques, and instead standard rewinding techniques for the stand-alone setting are applicable there. Finally, in the timing model, honest parties insert artificial delays in the protocol based on their clocks, while in our constructions, an honest party responds immediately to messages received from the other (possibly adversarial) party.

1.4 Organization

We start with our model of the blockchain in section 2, and all subsequent results are in this model. In section 4 we describe a $\omega(1)$ round black-box zero-knowledge protocol. We describe our concurrently extractable commitment scheme in section 5 and use our constructed commitment scheme to achieve a concurrently secure two-party computation protocol described in section 5.2.

1.5 Full Version

Due to space constraints, our impossibility results and the security proofs are omitted from this manuscript, and appear in the full version of this paper[23].

2 Blockchain Model

Blockchains. In a blockchain protocol, the goal of all parties is to maintain a global ordered set of records that are referred to as *blocks*. New blocks can only be added using a special mining procedure that simulates a puzzle-solving race between participants and can be run by any party (called miner) executing the blockchain protocol. Presently, two broad categories of puzzles are used: Proof-of-Work (PoW) and Proof-of-Stake (PoS).

Following the works of [47, 4, 3], we model the blockchain as a global ledger $\mathcal{G}_{\mathsf{ledger}}$ that internally keeps a state state which is the sequence of all the blocks in the ledger. Parties interact with the ledger by making one of many queries described by the functionality.

We reproduce here the ledger functionality described in [4] with a few minor modifications to be described subsequent to the description.

The ledger maintains a central and unique permanent state denoted by state. When data/transactions are sent to $\mathcal{G}_{\mathsf{ledger}}$, they are validated using a Validate predicate and added to a buffer buffer. The buffer is meant to indicate those transactions that are not sufficiently deep to become permanent. The Blockify function creates a block including some transactions from buffer and extends state. The decision of when the state is extended is left to the adversary. The adversary proposes a next block candidate NxtBC containing the transactions from the buffer it wants included in the state. An empty NxtBC is used to indicate that the adversary does not want the state to be updated at the current clock tick. To restrict the behavior of the adversary, there is a ledger algorithm ExtendPolicy that enforces a state-update policy restriction. Further discussion on the ExtendPolicy can be found in the full version.

Each registered party can see the state, but is guaranteed only a sufficiently long prefix of it. This is implemented by monotonically increasing pointers pt_i, defining the prefix $\mathsf{state}|_{\mathsf{pt}_i}$, for each party that the adversary can manipulate with some restrictions. This can be viewed as a sliding window over the state, wherein the adversary can only set pointers to be within this window starting from the head of state. The size of the sliding window is denoted by windowSize. It should be noted that the prefix view guarantees that the value at position k will appear in position i in every party's state.

A party is said to be desynchronized if the party recently registered or recently got de-registered from the clock. At this point, due to network delays, the adversary can make the parties believe in any value of the state up until the party gets messages from the network. This time period is denoted by the parameter Delay, wherein the desynchronized parties are practically under the control of the adversary. A timed honest input sequence $\overrightarrow{I}^T_H$, is a vector of the form $((x_1, P_1, \tau_1), \cdots, (x_m, P_m, \tau_m))$, used to denote the inputs received by the parties from the environment, where P_i is the player that received the input and τ_i was the time of the clock when the environment handed the input to P_i. The ledger uses the function predict-time to ensure that the ideal world execution advances with the same pace (relative to the clock) as the protocol does. $\overrightarrow{\tau}_{\mathsf{state}}$ denotes the block-insertion times vector, which lists the times each block was inserted into state.

Functionality $\mathcal{G}_{\mathsf{ledger}}$

$\mathcal{G}_{\mathsf{ledger}}$ is parameterized by found algorithms, Validate, ExtendPolicy, Blockify, and predict-time: windowSize, Delay$\in \mathbb{N}$. The functionality manages variables state, NxtBCbuffer, τ_L, and $\overrightarrow{\tau}_{\mathsf{state}}$ as described above. The variables are initialized as follows: state $:= \overrightarrow{\tau}_{\mathsf{state}} :=$ NxtBC $:= \varepsilon$, buffer $:= \emptyset$, $\tau_L = 0$.

The functionality maintains the set of registered parties $\mathcal{P}$, the subset of honest parties $\mathcal{H} \subseteq \mathcal{P}$ and the subset of de-synchronized honest parties $\mathcal{P}_{DS} \subset \mathcal{H}$. The sets $\mathcal{P}, \mathcal{H}, \mathcal{P}_{DS}$ are all initially set to $\emptyset$. When a new honest party is registered at the ledger, if it is registered with the clock already then it added to the party

sets $\mathcal{H}$ and $\mathcal{P}$ and the current time of registration is also recorded if the current time $\tau_L > 0$, it is also added to $\mathcal{P}_{DS}$. Similarly, when a party is deregistered, it is removed from both $\mathcal{P}$ (and therefore also from $\mathcal{P}_{DS}$ or $\mathcal{H}$). The ledger maintains the invariant that it is registered (as a functionality) to the clock whenever $\mathcal{H} \neq \emptyset$.

For each party $P_i \in \mathcal{P}$ the functionality maintains a pointer pt_i (initially set to 1) and a current-state view $\mathsf{state}_i := \varepsilon$ (initially set to empty). The functionality also keeps track of the timed honest-input sequence in a vector $\overrightarrow{\mathcal{I}}_H^T$ (initially $\overrightarrow{\mathcal{I}}_H^T := \varepsilon$)

Upon receiving any input I from any party or from the adversary, send $(\mathsf{CLOCK\text{-}READ}, \mathsf{sid}_C)$ to $\mathcal{G}_{\mathsf{clock}}$ and upon receiving the response $(\mathsf{CLOCK\text{-}READ}, \mathsf{sid}_C, \tau)$, set $\tau_L := \tau$ and do the following:

1. Let $\widehat{\mathcal{P}} \subseteq \mathcal{P}_{DS}$ denote the set of desynchronized honest parties that have been registered continuously since time $\tau' < \tau_L - \mathsf{Delay}$ (to both ledger and clock). Set $\mathcal{P}_{DS} := \mathcal{P}_{DS} \setminus \widehat{\mathcal{P}}$.
2. If I was received from an honest party $P_i \in \mathcal{P}$:
   (a) Set $\overrightarrow{\mathcal{I}}_H^T := \overrightarrow{\mathcal{I}}_H^T || (I, P_i, \tau_L)$;
   (b) Compute $\overrightarrow{N} = (\overrightarrow{N}_1, \cdots, \overrightarrow{N}_\ell) := \mathsf{ExtendPolicy}\left(\overrightarrow{\mathcal{I}}_H^T, \mathsf{state}, \mathsf{NxtBC}, \mathsf{buffer}, \overrightarrow{\tau}_{\mathsf{state}}\right)$ and if $\overrightarrow{N} \neq \varepsilon$ set $\mathsf{state} := \mathsf{state} || \mathsf{Blockify}(\overrightarrow{N}_1) || \cdots || \mathsf{Blockify}(\overrightarrow{N}_\ell)$ and $\overrightarrow{\tau}_{\mathsf{state}} := \overrightarrow{\tau}_{\mathsf{state}} || \tau_L^\ell$ where $\tau_L^\ell = \tau_L || \cdots || \tau_L$.
   (c) If there exists $P_j \in \mathcal{H} \setminus \mathcal{P}_{DS}$ such that $|\mathsf{state}| - \mathsf{pt}_j > \mathsf{windowSize}$ or $\mathsf{pt}_j < |\mathsf{state}_j|$, then set $\mathsf{pt}_k := |\mathsf{state}|$ for all $P_k \in \mathcal{H} \setminus \mathcal{P}_{DS}$.
   (d) If $\overrightarrow{N} \neq \varepsilon$, send (state) to $\mathcal{A}$; else send (I, P_i, τ_L) to $\mathcal{A}$
3. Depending on the above input I and its sender's ID, $\mathcal{G}_{\mathsf{ledger}}$ executes the corresponding code from the following list:
   – *Submitting data:*
   If $I = (\mathsf{SUBMIT}, \mathsf{sid}, x)$ and is received from a party $P_i \in \mathcal{P}$ or from $\mathcal{A}$ (on behalf of corrupted party P_i) do the following
   (a) Choose a unique identifier uid and set $y := (x, \mathsf{uid}, \tau_L, P_i)$
   (b) $\mathsf{buffer} := \mathsf{buffer} \cup \{y\}$.
   (c) Send (SUBMIT, y) to $\mathcal{A}$ if not received from $\mathcal{A}$.
   – *Reading the state:*
   If $I = (\mathsf{READ}, \mathsf{sid})$ is received from a party $P_i \in \mathcal{P}$ then set $\mathsf{state}_i := \mathsf{state}|_{\min\{\mathsf{pt}_i, |\mathsf{state}|\}}$ and return $(\mathsf{READ}, \mathsf{sid}, \mathsf{state}_i)$ to the requestor. If the the requestor is $\mathcal{A}$ then send $(\mathsf{state}, \mathsf{buffer})$.
   – *Maintain the ledger state:*
   If $I = (\mathsf{MAINTAIN\text{-}LEDGER}, \mathsf{sid})$ is received by an honest $P_i \in \mathcal{P}$ and $\mathsf{predict\text{-}time}(\overrightarrow{\mathcal{I}}_H^T) = \widetilde{\tau} > \tau_L$ then send $(\mathsf{CLOCK\text{-}UPDATE}, \mathsf{sid}_C)$ to $\mathcal{G}_{\mathsf{clock}}$. Else send I to $\mathcal{A}$.

– *The adversary proposing the next block:*
If $I = (\mathsf{NEXT\text{-}BLOCK}, \mathsf{hflag}, (\mathsf{uid}_1, \cdots, \mathsf{uid}_\ell))$ is sent from the adversary, update NxtBC as follows:
(a) Set $\mathsf{listOfUid} \leftarrow \varepsilon$
(b) For $i \in [\ell]$, if there exists $y \coloneqq (x, \mathsf{uid}, \tau_L, P_i) \in \mathsf{buffer}$ with ID $\mathsf{uid} = \mathsf{uid}_i$ then set $\mathsf{listOfUid} \coloneqq \mathsf{listOfUid}||\mathsf{uid}_i$.
(c) Finally, set $\mathsf{NxtBC} \coloneqq \mathsf{NxtBC}||(\mathsf{hflag}, \mathsf{listOfUid})$.

– *The adversary setting state-slackness:*
If $I = (\mathsf{SET\text{-}SLACK}, (P_{i_1}, \widehat{\mathsf{pt}_{i_1}}), \cdots, (P_{i_\ell}, \widehat{\mathsf{pt}_{i_\ell}}))$ with $\{P_{i_1}, \cdots, P_{i_\ell}\} \subseteq \mathcal{H} \setminus \mathcal{P}_{DS}$ is received from the adversary, do the following:
(a) If $\forall j \in [\ell] : |\mathsf{state}| - \widehat{\mathsf{pt}}_{i_j} \leq \mathsf{windowSize}$ and $\widehat{\mathsf{pt}}_{i_1} \geq |\mathsf{state}_{i_j}|$, set $\mathsf{pt}_{i_j} \coloneqq \widehat{\mathsf{pt}}_{i_j}$ for every $j \in [\ell]$.
(b) Otherwise set $\mathsf{pt}_{i_j} \coloneqq |\mathsf{state}|$ for all $j \in [\ell]$

– *The adversary setting the state for desynchronized parties:*
If $I = (\mathsf{DESYNC\text{-}STATE}, (P_{i_1}, \mathsf{state}'_{i_1}), \cdots, (P_{i_1}, \mathsf{state}'_{i_\ell}))$ with $\{P_{i_1}, \cdots, P_{i_\ell}\} \subseteq \mathcal{P}_{DS}$ is received from the adversary, set set $\mathsf{state}_{i_j} \coloneqq \mathsf{state}'_{i_j}$ for every $j \in [\ell]$.

The work of Badertscher et al [4] show that under appropriate assumptions, Bitcoin realizes the ledger functionality described enforcing the ExtendPolicy described in the full version of our paper. For convenience we've made a few syntactic changes to the $\mathcal{G}_{\mathsf{ledger}}$ functionality as described in [4]:

– Firstly, the Validate predicate is not relevant in our setting since parties will use ledger to post data, and these should be trivially validated. Hence, we've abstracted out the Validate predicate from the description of the model.
– We require that the adversary cannot invalidate data sent by other parties, thereby denying data from ever making it on to the ledger. For transactions, the adversary can invalidate honest transactions. This can be remedied using a strong variant of $\mathcal{G}_{\mathsf{ledger}}$ described in [4].
– Every time that the size of the state increases, the adversary is notified of the new state by $\mathcal{G}_{\mathsf{ledger}}$.

The changed functionality the same properties of the ideal $\mathcal{G}_{\mathsf{ledger}}$ functionality as described in [4].

Remarks. We point out a few properties of the $\mathcal{G}_{\mathsf{ledger}}$ functionality and its use case in our setting.

– As described in [4], we can achieve a strong liveness guarantee by slightly modifying the above ledger functionality which guarantees that posted information will make it on to the view of other parties within $\Delta \coloneqq 4 \cdot \mathsf{windowSize}$ number of blocks (relative to the view of the submitting party).

– There are occasions wherein we will run parallel executions of the adversary, and one thread will be assigned to be the main execution thread while the others will be denoted as 'look-ahead" threads. In an effort to make the adversary oblivious to rewinding, we cannot allow messages from these "look-ahead" threads to make its way to $\mathcal{G}_{\mathsf{ledger}}$. Drop messages sent by the adversary to $\mathcal{G}_{\mathsf{ledger}}$ and will have to abort the thread if $\mathcal{G}_{\mathsf{ledger}}$ sends a state with an increased size.
– We require that for a READ query, buffer is efficiently simulatable, while state is not. This is a reasonable assumption to make given that the state indicates the permanent component of the blockchain, and simulating this would requiring forging the state. On the other hand, the buffer consists of outstanding queries from both honest and adversarial parties. From the description of $\mathcal{G}_{\mathsf{ledger}}$, each time a SUBMIT query is made to $\mathcal{G}_{\mathsf{ledger}}$, the information is passed along to the adversary, and the adversary's own outstanding queries are known. Looking ahead, a READ query can be answered without making a query to $\mathcal{G}_{\mathsf{ledger}}$. The honest outstanding queries are replayed on each thread since they could not have changed across threads, while the adversarial queries local to that thread are known to the simulator.
– We wait for Delay time before the start of any protocol to ensure all parties are synchronized. Moving ahead, for simplicity of exposition, the notion of de-synchronised parties is ignored.
– While the works of [47, 4, 3] use $\mathcal{G}_{\mathsf{clock}}$ functionality, we do not require parties to have access to a clock and can consider this to be local to $\mathcal{G}_{\mathsf{ledger}}$. In fact our positive results do not rely on parties having access to a clock.
– Additionally, we require that a locally initialized $\mathcal{G}_{\mathsf{ledger}}$ is efficiently simulatable to any adversary that does not have additional access to the global $\mathcal{G}_{\mathsf{ledger}}$. These local $\mathcal{G}_{\mathsf{ledger}}$ will be useful in establishing certain properties of our protocol.

Blockchain active (BCA) adversaries. Consider an adversary that has access to $\mathcal{G}_{\mathsf{ledger}}$, and thus can post to and access the state (the entire blockchain) at any time. In fact its strategy in any protocol may be a function of the state. We refer to any such adversary that actively uses the $\mathcal{G}_{\mathsf{ledger}}$ as a *blockchain active adversary (*BCA*)*.

Simulation in the Blockchain-hybrid model. Moving ahead, we interchangeably use blockchain-hybrid and $\mathcal{G}_{\mathsf{ledger}}$-hybrid, while preferring the later for our formal descriptions. A simulator has the same power as other parties while accessing the global functionality $\mathcal{G}_{\mathsf{ledger}}$. In addition, it acts as an interface between the party and $\mathcal{G}_{\mathsf{ledger}}$, and thus can choose what messages between the party and the functionality it wants delivered. This is unlike the setting considered in [24, 36] where the simulator has control of the blockchain, and thus can "rewind" the blockchain by discarding and re-creating blocks. This is reminiscent of the difference between simulation in Universal Composability (UC) framework [14] and simulation in the global UC framework [15, 18, 43].

Our simulator can use arbitrary polynomial amount of parallelism. Although arbitrary, the polynomial is fixed in advance. We will use this modeling to run parallel invocations of the adversary by making copies.

At this point we would like to emphasize the need for considering this model for the simulator. We start off by mentioning that any party can use the state obtained from $\mathcal{G}_{\mathsf{ledger}}$ as the basis for its execution. Importantly, the adversary's view is now no longer determined solely by the message it receives from the simulator since the $\mathcal{G}_{\mathsf{ledger}}$ state gives it an additional auxiliary input. In the plain model, if we wanted to rewind the adversary back to a specific point in the execution, we could restart the adversary and send the same messages up to the specific point. And we were guaranteed that the adversary's responses would be identical. But now since the adversary has access to $\mathcal{G}_{\mathsf{ledger}}$, its responses could depend on the state of $\mathcal{G}_{\mathsf{ledger}}$.

Let us consider such an adversary. Now when the simulator tries to restart the adversary, suppose the state has expanded since. Even if the simulator provides the same messages as a previous execution, the adversary's behavior now may be drastically different and of potentially no use to the simulator. The simulator could ensure identical behavior by providing it the earlier truncated view of the state, but moving forward with this execution would be problematic since any message that the adversary wants to post will no longer appear on the state within the promised time period, and thus the adversary will notice that the $\mathcal{G}_{\mathsf{ledger}}$ no longer follows the model specified. Thus it is imperative that executions are run in parallel to ensure that views across multiple threads are identical if the same inputs are provided.

The above modeling is crucial for rewinding when we prove security of our protocols. We will work with this modeling unless otherwise specified. Looking ahead, our construction of the zero-knowledge proof in the non-black-box setting will use a modified variant of this model.

Security. Since the distinguisher attempting to distinguish between views of the adversary in the real and simulated setting has access to $\mathcal{G}_{\mathsf{ledger}}$, the simulator cannot create an isolated view of $\mathcal{G}_{\mathsf{ledger}}$ for the adversary. But as it turns out, the ability to initialize a local $\mathcal{G}_{\mathsf{ledger}}$ is a useful property useful in certain situations that we will leverage in our work.

Protocols in the plain model are a reference to any protocol that does not require its participants to interact with $\mathcal{G}_{\mathsf{ledger}}$ in any form. These protocols are proven secure without considering the presence of $\mathcal{G}_{\mathsf{ledger}}$. Given such a protocol, a blockchain active adversary may try to leverage access to this global functionality $\mathcal{G}_{\mathsf{ledger}}$ to gain undue advantage over the setting where it did not have such access. We are interested in such adversaries since we want to see how the security of known protocols or primitives fare when the adversary has access to the $\mathcal{G}_{\mathsf{ledger}}$.

3 Definitions and Preliminaries

Unless otherwise specified, we consider the adversaries that have access to the global functionality $\mathcal{G}_{\mathsf{ledger}}$, and thus the view includes messages received from

and sent to $\mathcal{G}_{\mathsf{ledger}}$. Thus, when we denote that two distributions representing the views of parties with access to $\mathcal{G}_{\mathsf{ledger}}$ are computationally indistinguishable in the $\mathcal{G}_{\mathsf{ledger}}$-hybrid model, we give distinguisher access to the global $\mathcal{G}_{\mathsf{ledger}}$ functionality. An immediate consequence of this is that, any view generated by a simulator using a privately initialized $\mathcal{G}_{\mathsf{ledger}}$ functionality will be trivially distinguished from the real execution by the distinguisher that views the state of the global $\mathcal{G}_{\mathsf{ledger}}$.

3.1 Zero Knowledge in the $\mathcal{G}_{\mathsf{ledger}}$-hybrid model

Definition 1. *An interactive protocol* (P, V) *for a language* L *is zero knowledge in the* $\mathcal{G}_{\mathsf{ledger}}$*-hybrid model if the following properties hold:*

- **Completeness.** *For every* $x \in L$*,*

$$\Pr\left[\mathsf{out}_{\mathsf{V}}\left[\mathsf{P}(x,w) \leftrightarrow \mathsf{V}(x)\right] = 1\right] = 1$$

- **Soundness.** *There exists a negligible function* $\mathsf{negl}(\cdot)$ *s.t.* $\forall x \notin L$ *and for all adversarial prover* P^**.*

$$\Pr\left[\mathsf{out}_{\mathsf{V}}\left[\mathsf{P}^*(x) \leftrightarrow \mathsf{V}(x)\right] = 1\right] \leq \mathsf{negl}(n)$$

- **Zero Knowledge.** *For every PPT adversary* V^**, there exists a PPT simulator* Sim *such that the probability ensembles*
  - $\left\{\mathsf{view}_{\mathsf{V}}\left[\mathsf{P}(x,w) \leftrightarrow \mathsf{V}(x,z)\right]\right\}_{x\in L, w\in R_L(x), z\in\{0,1\}^*}$
  - $\left\{\mathsf{Sim}(x,z)\right\}_{x\in L, w\in R_L(x), z\in\{0,1\}^*}$

  are computationally indistinguishable *in the* $\mathcal{G}_{\mathsf{ledger}}$*-model.*

3.2 Concurrently Secure Computation in the $\mathcal{G}_{\mathsf{ledger}}$-hybrid model

In this work, we consider a malicious, static adversary that chooses whom to corrupt before the execution of the protocol. The adversary controls the scheduling of the concurrent executions. We only consider *computational* security and therefore restrict our attention to adversaries running in probabilistic polynomial time. We denote computational indistinguishability by $\approx_c$, and the security parameter by n. We do not require fairness and hence in the ideal model, we allow a corrupt party to receive its output in a session and then optionally block the output from being delivered to the honest party, in that session. Further, we only consider "security with abort". To formalize the above requirement and define security, we follow the standard paradigm for defining secure computation (see also [52]). We define an ideal model of computation and a real model of computation, and require that any adversary in the real model can be *emulated* by an adversary in the ideal model. More details follow.

IDEAL MODEL. We first define the ideal world experiment, where there is a trusted party for computing the desired two-party functionality $\mathcal{F} : \{0,1\}^{r_1} \times$

$\{0,1\}^{r_2} \to \{0,1\}^{s_1} \times \{0,1\}^{s_2}$. Let P_1 and P_2 denote the two parties in a single execution. In total. let there be k parties $Q_1, Q_2, \cdots, Q_k$, where each party may be involved in multiple sessions with possibly interchangeable roles, i.e. Q_i may play the role of P_1 in one session and P_2 in some other session. Let the total number of executions be $m = m(n)$. For each $\ell \in [m]$, we will denote by P_1^ℓ, the party playing the role of P_1 in session ℓ. P_2^ℓ is defined analogously. The adversary may corrupt any subset of the parties in $Q_1, \ldots, Q_k$. The ideal world execution proceeds as follows:

I **Inputs:** There is a PPT *usage scenario* which gives inputs to all the parties. For each session $\ell \in [m]$, it gives inputs $x_\ell \in X \subseteq \{0,1\}^{r_1}$ to P_1^ℓ and $y_\ell \in Y \subseteq \{0,1\}^{r_2}$ to P_2^ℓ. The adversary is given auxiliary input $z \in \{0,1\}^*$, and chooses the subset of the parties to corrupt, say M. The adversary receives the inputs of the corrupted parties.

II **Session initiation:** When the adversary wishes to initiate the session number ℓ, it sends a $(\mathsf{start\text{-}session}, \ell)$ message to the trusted party. On receiving a message of the form $(\mathsf{start\text{-}session}, \ell)$, the trusted party sends $(\mathsf{start\text{-}session}, \ell)$ to both P_1^ℓ and P_2^ℓ.

III **Honest parties send inputs to the trusted party:** Upon receiving $(\mathsf{start\text{-}session}, \ell)$ from the trusted party, an honest party P_i^ℓ sends its real input along with the session identifier. More specifically, if P_1^ℓ is honest, it sends (ℓ, x_ℓ) to the trusted party. Similarly, an honest P_2^ℓ sends (ℓ, y_ℓ) to the trusted party.

IV **Corrupted parties send inputs to the trusted party:** At any point during execution, a corrupted part P_1^ℓ may send a message (ℓ, x'_ℓ) to the trusted party, for any string x'_ℓ (of appropriate length) of its choice. Similarly, a corrupted party P_2^ℓ sends (ℓ, y'_ℓ) to the trusted party, for any string y'_ℓ (of appropriate length) of its choice.

V **Trusted party sends results to the adversary:** For a session ℓ, when the trusted party has received messages from both P_1^ℓ and P_2^ℓ, it computes the output for that session. Let x'_ℓ and y'_ℓ be the inputs received from P_1^ℓ and P_2^ℓ, respectively. It computes the output $\mathcal{F}(x'_\ell, y'_\ell)$. If either P_1^ℓ or P_2^ℓ is corrupted, it sends $(\ell, \mathcal{F}(x'_\ell, y'_\ell))$ to the adversary. If neither of the parties is corrupted, then the trusted party sends the output message $(\ell, \mathcal{F}(x'_\ell, y'_\ell))$ to both P_1^ℓ and P_2^ℓ.

VI **Adversary instructs the trusted party to answer honest players:** For a session ℓ, where exactly one of the party is corrupted, the adversary, depending on its view up to this point, may send the message (output, ℓ) to the trusted party. Then, the trusted party sends the output $(\ell, \mathcal{F}(x'_\ell, y'_\ell))$, computed in the previous step, to the honest party in session ℓ.

VII **Outputs:** An honest party always outputs the value that it received from the trusted party. The adversary outputs an arbitrary (PPT computable) function of its entire view (including the view of all corrupted parties) throughout the execution of the protocol including messages exchanged with the $\mathcal{G}_{\mathsf{ledger}}$ functionality.

The ideal execution of a function $\mathcal{F}$ with security parameter n, input vectors $\overrightarrow{x}, \overrightarrow{y}$, auxiliary input z to Sim and the set of corrupted parties M, denoted by $\mathsf{IDEAL}^{\mathcal{F}}_{M,\mathsf{Sim}}(n, \overrightarrow{x}, \overrightarrow{y}, z)$, is defined as the output pair of the honest parties and the ideal world adversary Sim from the above ideal execution.

REAL MODEL. We now consider the real model in which a real two-party protocol is executed (and there exists no trusted third party). Let $\mathcal{F}, \overrightarrow{x}, \overrightarrow{y}, z$ be as above and let Π be a two-party protocol for computing $\mathcal{F}$. Let $\mathcal{A}$ denote a non-uniform probabilistic polynomial time adversary that controls any subset M of parties $Q_1, \ldots, Q_k$. The parties run concurrent executions of the protocol Π, where the honest parties follow the instructions of Π in all executions. The honest party initiates a new session ℓ, using the input provided whenever it receives a start-session message from $\mathcal{A}$. The scheduling of all messages throughout the execution is controlled by the adversary. That is, the execution proceeds as follows: the adversary sends a message of the form (ℓ, msg) to the honest party. The honest party then adds msg to its view of session ℓ and replies according to the instructions of Π and this view in that session. At the conclusion of the protocol, an honest party computes its output as prescribed by the protocol. Without loss of generality, we assume the adversary outputs exactly its entire view in the execution of the protocol, which includes messages exchanged with the $\mathcal{G}_{\mathsf{ledger}}$ functionality.

The real concurrent execution of Π with security parameter n, input vectors $\overrightarrow{x}, \overrightarrow{y}$, auxiliary input z to $\mathcal{A}$ and the set of corrupted parties M, denoted by $\mathsf{REAL}^{\mathcal{F}}_{M,\mathcal{A}}(n, \overrightarrow{x}, \overrightarrow{y}, z)$, is defined as the output pair of the honest parties and the real world adversary $\mathcal{A}$ from the above real world process.

Definition 2. *Let $\mathcal{F}$ and Π be as above. Then protocol Π for computing $\mathcal{F}$ is a concurrently secure computation protocol in the $\mathcal{G}_{ledger}$-hybrid model if for every probabilistic polynomial time adversary $\mathcal{A}$ in the real model, there exists a probabilistic polynomial time adversary* Sim *in the ideal model such that for every polynomial $m = m(n)$, every input vectors $\overrightarrow{x} \in X^m$, $\overrightarrow{y} \in Y^m$, every $z \in \{0,1\}^*$, and every subset of corrupt parties M, the following*

$$\left\{\mathsf{IDEAL}^{\mathcal{F}}_{M,\mathsf{Sim}}(n, \overrightarrow{x}, \overrightarrow{y}, z)\right\}_{n\in\mathbb{N}} \approx_c \left\{\mathsf{REAL}^{\mathcal{F}}_{M,\mathcal{A}}(n, \overrightarrow{x}, \overrightarrow{y}, z)\right\}_{n\in\mathbb{N}}$$

holds in the $\mathcal{G}_{ledger}$-hybrid model.

4 Black-box Zero Knowledge

In this section we will describe a $\omega(1)$ round zero-knowledge protocol that can be proven secure using a black-box simulator. We build our protocol atop the protocol for graph hamiltonicity proof.

4.1 Our Protocol

The high level idea for our protocol is that the verifier commits to its challenge via a multi-round extractable commitment, and reveals the challenge in place of

the second round of the Hamiltonicity proof system. Since we are constructing a proof system where the prover has unbounded computational power, we require the commitment by the verifier to be statistically hiding so that an unbounded adversarial prover is not able to guess the challenge. We refer to the multi-round extractable commitment as the preamble.

In the preamble, the challenge committed to by the verifier is retrieved by rewinding the verifier in each of the slots. As long as the rewinding is successful in one of the slots, the committed challenge can be extracted. But in the presence of the blockchain (abstracted by the $\mathcal{G}_{\mathsf{ledger}}$ functionality) this becomes difficult. Consider a verifier that sends the challenge received by the prover in a given slot to $\mathcal{G}_{\mathsf{ledger}}$, and waits for the state to expand to include the challenge before responding to the challenge. It then checks in the state if there is another challenge from the prover for the same slot. If this is the case, it knows that it has been rewound, and will abort the protocol. Thus, in the simulated setting, the verifier will abort with a disproportionate probability in comparison to the real execution.

The trivial solution of not relaying messages from the verifier to $\mathcal{G}_{\mathsf{ledger}}$ on the look ahead threads does not work because the verifier can refuse to respond unless the state expands.

Thus, to overcome this issue, we design a protocol in the blockchain-hybrid model, where the protocol requires all parties to access $\mathcal{G}_{\mathsf{ledger}}$ in order to participate in the protocol. In our protocol, we just require that during the preamble, the local state of each party increases by at most k. But since parties may have different views of thus state, we must be careful when we claim the state size increase for other parties. But since $\mathcal{G}_{\mathsf{ledger}}$ guarantees that $|\mathsf{state}_{\mathsf{P}} - \mathsf{state}_{\mathsf{V}}| \leq \mathsf{windowSize}$, we are guaranteed that if the size of the state of one party increases by k, the size of the state of any other party can increase by at most $\mathsf{windowSize} + k$ (with maximum when both parties point to the head of the state initially).

If we set the number of rounds of the preamble to be $m > k + \mathsf{windowSize}$, we are guaranteed to have at least $m - (k + \mathsf{windowSize})$ slots where the state does not expand during the slot. For simplicity we assume k to be a constant, but our protocol can handle arbitrary k by scaling the number of rounds accordingly. The high level idea then is to just rewind in the slots where the state has not expanded, and thus the verifier does not expect the state to expand before it responds, and thus messages to or from $\mathcal{G}_{\mathsf{ledger}}$ can be kept from the verifier on the look ahead threads. Of course the exact number of rounds would depend on the exact simulator strategy. In our protocol, the number of rounds in the preamble is set to be $m = \omega(1)$. We should point out that $k > \mathsf{windowSize}$ to avoid trivial aborts in an honest execution of the protocol since otherwise the parties may start off with states that may then be k behind the head of the state, and in one computation step catches up to the head, thereby increasing local state size by k, and thus causing an abort. The complete protocol is presented in Figure 1.

Protocol BCA-ZK

Common Input: An instance x of a language L with witness relation R_L, the security parameter n, the time out parameter k and the round parameter $m := m(n)$.

Auxiliary Input for Prover: a witness w, such that $(x, w) \in R_L$, size of local state from the ledger $i_{\mathsf{P}} := |\mathsf{state}_{\mathsf{P}}|$.

Auxiliary Input for Verifier: size of local state from the ledger $i_{\mathsf{V}} := |\mathsf{state}_{\mathsf{V}}|$.

Phase I: Prior to each message sent in this phase, the respective party checks if the size of the state is such that $|\mathsf{state}_{\mathsf{P}}| < i_{\mathsf{P}} + k$ (correspondingly $|\mathsf{state}_{\mathsf{V}}| < i_{\mathsf{V}} + k$ for the verifier). If not, the party aborts.

1. Prover uniformly select a first message for a two round *statistically hiding commitment scheme* and send it to the verifier.
2. Verifier uniformly selects $\sigma \in \{0,1\}^n$, and mn pairs of n-bit strings $(\sigma^0_{\ell,p}, \sigma^1_{\ell,p})$ for $\ell \in [n], p \in [m]$ such that $\forall \ell, p: \ \sigma^0_{\ell,p} \oplus \sigma^1_{\ell,p} = \sigma$. It commits to all $2mn+1$ selected strings using the statistically hiding commitment scheme. The commitments are denoted by $\alpha, \{\alpha^b_{\ell,p}\}_{b\in\{0,1\},\ell\in[n],p\in[m]}$.
3. For $p = 1$ to m:
   (a) Prover sends an n-bit challenge string $r_p = r_{1,p}, \ldots, r_{n,p}$ to the verifier.
   (b) Verifier decommits $\alpha^{r_{1,p}}_{1,p}, \ldots, \alpha^{r_{n,p}}_{n,p}$ to $\sigma^{r_{1,p}}_{1,p}, \ldots, \sigma^{r_{n,p}}_{n,p}$.
4. The prover proceeds with the execution if and only if all the decommitments send by the verifier are valid.

Phase II: The prover and verifier engage in n parallel executions of the Hamiltonicity protocol as described below:

1. The prover sends the first message of the Hamiltonicity proof system.
2. The verifier decommits α to σ. And also reveals all mn commitments not decommitted to in the earlier phase.
3. The prover checks if decommitted values $\sigma, \{\sigma^b_{\ell,p}\}_{b\in\{0,1\},\ell\in[n],p\in[m]}$ are valid decommitments. Additionally, check if $\forall \ell, p: \ \sigma^0_{\ell,p} \oplus \sigma^1_{\ell,p} = \sigma$. If any of the checks fail, abort. Else, send the third message of the Hamiltonicity proof system.
4. Verifier checks if all conditions of the Hamiltonicity proof system are met. It accepts if and only if this is the case.

Fig. 1. Protocol for zero-knowledge proof in the blockchain aware setting.

Theorem 6. *The protocol* BCA-ZK *is a Zero-Knowledge Proof with black-box simulation in the* $\mathcal{G}_{\mathsf{ledger}}$-hybrid model.

5 Concurrent Self Composable Secure Computation

In this section, we will construct a two-party protocol that is secure under concurrent self composition. We follow the line of works [17, 38, 40] that rely on realizing an extractable commitment scheme that remains extractable even when

there are multiple concurrent copies of this scheme in execution. Thus we construct our protocol in a two-step process. First, we describe a modified version of the multi-round extractable commitment preamble in the blockchain-hybrid model and show that we can extract from each session when multiple sessions are executed concurrently. Next, we plug our constructed concurrently extractable commitments into the compilers constructed in [17, 38, 40] to achieve a concurrently secure two-party computation protocol.

5.1 Concurrently Extractable Commitment

In this section we present our construction of the concurrently extractable commitment scheme in the blockchain-hybrid model. We will refer to this as the modified PRS preamble. The idea for the modified PRS preamble is quite simple. Prior to starting the preamble, the party needs to post the first message to $\mathcal{G}_{\mathsf{ledger}}$. It is guaranteed that it will appear in the view of every party within the next $\Delta := 4 \cdot \mathsf{windowSize}$ blocks. Once the local state increase by Δ blocks, it sends the same message to the receiver. Posting to $\mathcal{G}_{\mathsf{ledger}}$ gives the party an "expiry period" of k-blocks after the Δ wait i.e., all slots of the preamble must be completed before the size of the state increases by a total of $\Delta+k$. As in the case of zero-knowledge, if the size of the state of a party increases by $\Delta + k$, for any other party the size of the state can have increased by at most $\Delta+k+\mathsf{windowSize}$, which is a constant when k is a constant. This needs to be taken into account when choosing the parameters ℓ and k. The formal description of the protocol is given below.

Protocol $\langle C, R\rangle_{\mathsf{BCA}}$

Common Input: The security parameter n, the time-out parameter k, and the round parameter $2 \cdot \ell := \ell(n)$.

Input to the Committer: the value σ to be committed, size of local state from the ledger $i_C := |\mathsf{state}_C|$.

Input to the Receiver: size of local state from the ledger $i_R := |\mathsf{state}_R|$.

Commitment:

1. Committer uniformly selects $\sigma \in \{0,1\}^n$, and $2 \cdot \ell \cdot n$ pairs of n-bit strings $(\sigma^0_{\ell,p}, \sigma^1_{\ell,p})$ for $\ell \in [n], p \in [2 \cdot \ell]$ such that $\forall \ell, p : \ \sigma^0_{\ell,p} \oplus \sigma^1_{\ell,p} = \sigma$. It generate commitments to all $2(2 \cdot \ell) \cdot n + 1$ selected strings using the statistically binding commitment scheme. The commitments are denoted by $\alpha, \{\alpha^b_{\ell,p}\}_{b\in\{0,1\},\ell\in[n],p\in[2\cdot\ell]}$. Send a SUBMIT query of these commitments to $\mathcal{G}_{\mathsf{ledger}}$. By our assumption, these will be guaranteed to appear in every party's state (at the same position) when $|\mathsf{state}_C| = i_C + \Delta$. Let it appear in index i of the state.

2. The committer sends to receiver the commitments along with the index i of the state that it appears in. The receiver verifies if the commitments were indeed in the designated index of the state.
3. Prior to each message subsequently sent, the respective party checks if the size of the state is such that $|\mathsf{state}_R| < i_R + k + \Delta$ (correspondingly $|\mathsf{state}_C| < i_C + k + \Delta$ for the committer). If not, the party aborts.
   For $p = 1$ to m:
   (a) Receiver sends an n-bit challenge string $r_p = r_{1,p}, \ldots, r_{1^n,p}$ to the committer.
   (b) Committer decommits $\alpha_{1,p}^{r_{1,p}}, \ldots, \alpha_{n,p}^{r_{n,p}}$ to $\sigma_{1,p}^{r_{1,p}}, \ldots, \sigma_{n,p}^{r_{n,p}}$.

In the full version of our paper, we describe the simulation-extraction strategy to extract values committed by an adversary in every session of multiple concurrent executions of the modified preamble described above.

5.2 Protocol for Concurrent Self Composable Secure Computation

We now describe our concurrent secure computation protocol Π in the $\mathcal{G}_{\mathsf{ledger}}$-hybrid model for a general functionality $\mathcal{F}$. Our protocol is, in fact, the same as the one presented in [40, 38, 17], except that we use the concurrently extractable commitment from Section 5.1. Indeed, the core ingredient of the compiler in [40] (which is also used in [38, 17]) is a concurrently extractable commitment, and in particular, it follows from these works that if there exists a concurrent simulator for the extractable commitment, then the resultant compiled protocol securely evaluates the function $\mathcal{F}$.

For completeness, we recall the protocol here. The proof of security for our case follows in essentially an identical fashion to [40] with the main difference being that our simulator only performs a single ideal world query per session (while the simulator performs multiple ideal world queries per session in their work). We discuss other minor differences below.

Building Blocks

Statistical Binding String Commitments. We will use a (2-round) statistically binding string commitment scheme, e.g., a parallel version of Naor's bit commitment scheme [54] based on one-way functions. For simplicity of exposition, however, in the presentation of our results, we will use a non-interactive perfectly binding string commitment. Let $\mathsf{com}(\cdot)$ denote the commitment function of the string commitment scheme.

Statistical Witness Indistinguishable Arguments. We shall use a statistically witness indistinguishable (SWI) argument $\langle P_{\mathsf{swi}}, V_{\mathsf{swi}} \rangle$ for proving membership in any **NP** language with perfect completeness and negligible soundness error. Such a scheme can be constructed by using $\omega(\log k)$ copies of Blum's Hamiltonicity protocol [11] in parallel, with the modification that the prover's

commitments in the Hamiltonicity protocol are made using a statistically hiding commitment scheme [55, 42] .

Semi-Honest Two Party Computation. We will also use a semi-honest two party computation protocol $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}} \rangle$ that emulates the ideal functionality $\mathcal{F}$ in the stand-alone setting. The existence of such a protocol $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}} \rangle$ follows from [59, 34, 48].

Concurrent Non-Malleable Zero Knowledge Argument. Concurrent non-malleable zero knowledge (CNMZK) considers the setting where a man-in-the-middle adversary is interacting with several honest provers and honest verifiers in a concurrent fashion: in the "left" interactions, the adversary acts as verifier while interacting with honest provers; in the "right" interactions, the adversary tries to prove some statements to honest verifiers. The goal is to ensure that such an adversary cannot take "help" from the left interactions in order to succeed in the right interactions. This intuition can be formalized by requiring the existence of a machine called the simulator-extractor that generates the view of the man-in-the-middle adversary and additionally also outputs a witness from the adversary for each "valid" proof given to the verifiers in the right sessions.

Barak, Prabhakaran and Sahai [8] gave the first construction of a concurrent non-malleable zero knowledge (CNMZK) argument for every language in **NP** with perfect completeness and negligible soundness error. In our construction, we will use a specific CNMZK protocol, denoted $\langle P, V \rangle$, based on the CNMZK protocol of Barak et al. [8] to guarantee non-malleability. Specifically, we will make the following two changes to Barak et al's protocol: (a) Instead of using an $\omega(\log n)$-round extractable commitment scheme [57], we will use the N-round extractable commitment scheme $\langle C, R \rangle$ (described in the full version). (b) Further, we require that the non-malleable commitment scheme being used in the protocol be public-coin w.r.t. receiver[5]. We now describe the protocol $\langle P, V \rangle$.

Let P and V denote the prover and the verifier respectively. Let L be an NP language with a witness relation R. The common input to P and V is a statement $x \in L$. P additionally has a private input w (witness for x). Protocol $\langle P, V \rangle$ consists of two main phases: (a) the *preamble phase*, where the verifier commits to a random secret (say) σ via an execution of $\langle C, R \rangle$ with the prover, and (b) the *post-preamble phase*, where the prover proves an NP statement. In more detail, protocol $\langle P, V \rangle$ proceeds as follows.

Preamble Phase.

1. P and V engage in execution of $\langle C, R \rangle$ where V commits to a random string σ.

[5] The original NMZK construction only required a public-coin extraction phase inside the non-malleable commitment scheme. We, however, require that the entire commitment protocol be public-coin. We note that the non-malleable commitment protocol of [26] only consists of standard perfectly binding commitments and zero knowledge proof of knowledge. Therefore, we can easily instantiate the DDN construction with public-coin versions of these primitives such that the resultant protocol is public-coin.

Post-preamble Phase.

2. P commits to 0 using a statistically-hiding commitment scheme. Let c be the commitment string. Additionally, P proves the knowledge of a valid decommitment to c using a statistical zero-knowledge argument of knowledge (SZKAOK).
3. V now reveals σ and sends the decommitment information relevant to $\langle C, R\rangle$ that was executed in step 1.
4. P commits to the witness w using a public-coin non-malleable commitment scheme.
5. P now proves the following statement to V using SZKAOK:
   (a) *either* the value committed to in step 4 is a valid witness to x (i.e., $R(x, w) = 1$, where w is the committed value), *or*
   (b) the value committed to in step 2 is the trapdoor secret σ.
   P uses the witness corresponding to the first part of the statement.

Modified Extractable Commitment Scheme $\langle C', R'\rangle$ Due to technical reasons, in our secure computation protocol, we will also use a minor variant, denoted $\langle C', R'\rangle_{\mathsf{BCA}}$, of the extractable commitment scheme presented in 5.1. Protocol $\langle C', R'\rangle_{\mathsf{BCA}}$ is the same as $\langle C, R\rangle_{\mathsf{BCA}}$, except that for a given receiver challenge string, the committer does not "open" the commitments, but instead simply reveals the appropriate committed values (without revealing the randomness used to create the corresponding commitments). More specifically, in protocol $\langle C', R'\rangle_{\mathsf{BCA}}$, on receiving a challenge string $v_j = v_{1,j}, \ldots, v_{\ell,j}$ from the receiver, the committer uses the following strategy: for every $i \in [\ell]$, if $v_{i,j} = 0$, C' sends $\alpha^0_{i,j}$, otherwise it sends $\alpha^1_{i,j}$ to R'. Note that C' does not reveal the decommitment values associated with the revealed shares.

When we use $\langle C', R'\rangle_{\mathsf{BCA}}$ in our main construction, we will require the committer C' to prove the "correctness" of the values (i.e., the secret shares) it reveals in the last step of the commitment protocol. In fact, due to technical reasons, we will also require the the committer to prove that the commitments that it sent in the first step are "well-formed".

We remark that the extraction proof for the simulation-extraction procedure also holds for the $\langle C', R'\rangle_{\mathsf{BCA}}$ commitment scheme.

Protocol Description Notation. Let $\mathsf{com}(\cdot)$ denote the commitment function of a non-interactive perfectly binding commitment scheme. Let $\langle C, R\rangle_{\mathsf{BCA}}$ denote the N-round extractable commitment scheme and $\langle C', R'\rangle_{\mathsf{BCA}}$ be its modified version as described above. For the description, we drop the subscript and refer to them as $\langle C, R\rangle$ and $\langle C', R'\rangle$ respectively. Let $\langle P, V\rangle$ denote the modified version of the CNMZK argument of Barak et al. [8]. Further, let $\langle P_{\mathsf{swi}}, V_{\mathsf{swi}}\rangle$ denote a SWI argument and let $\langle P^{\mathsf{sh}}_1, P^{\mathsf{sh}}_2\rangle$ denote a semi-honest two party computation protocol $\langle P^{\mathsf{sh}}_1, P^{\mathsf{sh}}_2\rangle$ that securely computes $\mathcal{F}$ in the stand-alone setting as per the standard definition of secure computation.

Let P_1 and P_2 be two parties with inputs x_1 and x_2. Let n be the security parameter. The protocol proceeds as follows.

Protocol BCA-CONC

I. Trapdoor Creation Phase.

1. $P_1 \Rightarrow P_2 : P_1$ creates a commitment $\mathsf{Com}_1 = \mathsf{com}(0)$ to bit 0 and sends Com_1 to P_2. P_1 and P_2 now engage in the execution of $\langle P, V \rangle$ where P_1 proves that Com_1 is a commitment to 0.
2. $P_2 \Rightarrow P_1 : P_2$ now acts symmetrically. That is, it creates a commitment $\mathsf{Com}_2 = \mathsf{com}(0)$ to bit 0 and sends Com_2 to P_1. P_2 and P_1 now engage in the execution of $\langle P, V \rangle$ where P_2 proves that Com_2 is a commitment to 0.

Informally speaking, the purpose of this phase is to aid the simulator in obtaining a "trapdoor" to be used during the simulation of the protocol.

II. Input Commitment Phase. In this phase, the parties commit to their inputs and random coins (to be used in the next phase) via the commitment protocol $\langle C', R' \rangle$.

1. $P_1 \Rightarrow P_2 : P_1$ first samples a random string r_1 (of appropriate length, to be used as P_1's randomness in the execution of $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}} \rangle$ in Phase III) and engages in an execution of $\langle C', R' \rangle$ (denoted as $\langle C', R' \rangle_{1 \to 2}$) with P_2, where P_1 commits to $x_1 \| r_1$. Next, P_1 and P_2 engage in an execution of $\langle P_{\mathsf{swi}}, V_{\mathsf{swi}} \rangle$ where P_1 proves the following statement to P_2: (a) *either* there exist values $\hat{x}_1$, $\hat{r}_1$ such that the commitment protocol $\langle C', R' \rangle_{1 \to 2}$ is *valid* with respect to the value $\hat{x}_1 \| \hat{r}_1$, *or* (b) Com_1 is a commitment to bit 1.
2. $P_2 \Rightarrow P_1 : P_2$ now acts symmetrically. Let r_2 (analogous to r_1 chosen by P_1) be the random string chosen by P_2 (to be used in the next phase).

Informally speaking, the purpose of this phase is aid the simulator in extracting the adversary's input and randomness.

III. Secure Computation Phase. In this phase, P_1 and P_2 engage in an execution of $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}} \rangle$ where P_1 plays the role of P_1^{sh}, while P_2 plays the role of P_2^{sh}. Since $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}} \rangle$ is secure only against semi-honest adversaries, we first enforce that the coins of each party are truly random, and then execute $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}} \rangle$, where with every protocol message, a party gives a proof using $\langle P_{\mathsf{swi}}, V_{\mathsf{swi}} \rangle$ of its honest behavior "so far" in the protocol. We now describe the steps in this phase.

1. $P_1 \leftrightarrow P_2 : P_1$ samples a random string r_2' (of appropriate length) and sends it to P_2. Similarly, P_2 samples a random string r_1' and sends it to P_1. Let $r_1'' = r_1 \oplus r_1'$ and $r_2'' = r_2 \oplus r_2'$. Now, r_1'' and r_2'' are the random coins that P_1 and P_2 will use during the execution of $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}} \rangle$.
2. Let t be the number of rounds in $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}} \rangle$, where one round consists of a message from P_1^{sh} followed by a reply from P_2^{sh}. Let transcript $T_{1,j}$ (resp., $T_{2,j}$) be defined to contain all the messages exchanged between

P_1^{sh} and P_2^{sh} before the point P_1^{sh} (resp., P_2^{sh}) is supposed to send a message in round j. For $j = 1, \ldots, t$:

(a) $P_1 \Rightarrow P_2$: Compute $\beta_{1,j} = P_1^{\mathsf{sh}}(T_{1,j}, x_1, r_1'')$ and send it to P_2. P_1 and P_2 now engage in an execution of $\langle P_{\mathsf{swi}}, V_{\mathsf{swi}} \rangle$, where P_1 proves the following statement:

i. *either* there exist values $\hat{x}_1$, $\hat{r}_1$ such that (a) the commitment protocol $\langle C', R' \rangle_{1\to 2}$ is *valid* with respect to the value $\hat{x}_1 \| \hat{r}_1$, and (b) $\beta_{1,j} = P_1^{\mathsf{sh}}(T_{1,j}, \hat{x}_1, \hat{r}_1 \oplus r_1')$

ii. *or*, Com_1 is a commitment to bit 1.

(b) $P_2 \Rightarrow P_1$: P_2 now acts symmetrically.

Proof of Security. Our proof of security follows in almost an identical fashion to [40, 38, 17]. The main difference is that due to the property of our concurrent extractor (discussed in the full version), our simulator only needs to make one ideal world query per session (as opposed to multiple ideal world queries). Indeed, this is why we achieve standard concurrent security, while [40, 38, 17] achieve security in the so-called multiple-ideal-query model.

Our indistinguishability hybrids also follow in the same manner as in [40, 38, 17]. There is one minor difference that we highlight. The hybrids of [40, 38, 17] maintain a "soundness invariant", where roughly speaking, it is guaranteed that whenever an honest party changes its input in any sub-protocol used within the secure computation protocol, the value committed by the adversary in the non-malleable commitment (inside the CNMZK) does not change, except with negligible probability. In some hybrids, this property is argued via extraction from the non-malleable commitment.

In our setting, we have to be careful with such an extraction since a blockchain-active adversary may try to keep state using $\mathcal{G}_{\mathsf{ledger}}$. However, the key point is that for such a soundness argument, the reduction can use a locally initialized $\mathcal{G}_{\mathsf{ledger}}$ that it controls (and can therefore modify arbitrarily). This follows from the fact that we do not care about the view of an adversary in such a reduction to be indistinguishable to a distinguisher that has access to $\mathcal{G}_{\mathsf{ledger}}$. In fact, it will trivially be distinguishable. But since a locally initialized $\mathcal{G}_{\mathsf{ledger}}$ is indistinguishable to the adversary that is simply allowed to interact using the given interface (i.e. efficiently simulatable), the adversary's behavior does not change. Using this idea, we can perform extraction as in the plain model.

Acknowledgments. The second author's research was supported in part by a grant from Northrop Grumman, a gift from DOS Networks, and, a Cylab seed funding award. The first and third authors' research was supported in part by a DARPA/ARL Safeware Grant W911NF-15-C-0213, and a subaward from NSF CNS-1414023.

References

1. Agrawal, S., Goyal, V., Jain, A., Prabhakaran, M., Sahai, A.: New impossibility results for concurrent composition and a non-interactive completeness theorem for secure computation. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 443–460. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 19–23, 2012)
2. Andrychowicz, M., Dziembowski, S., Malinowski, D., Mazurek, L.: Secure multiparty computations on bitcoin. In: 2014 IEEE Symposium on Security and Privacy, SP 2014, Berkeley, CA, USA, May 18-21, 2014. pp. 443–458 (2014)
3. Badertscher, C., Gaži, P., Kiayias, A., Russell, A., Zikas, V.: Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability. Cryptology ePrint Archive, Report 2018/378 (2018), https://eprint.iacr.org/2018/378
4. Badertscher, C., Maurer, U., Tschudi, D., Zikas, V.: Bitcoin as a transaction ledger: A composable treatment. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017, Part I. LNCS, vol. 10401, pp. 324–356. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 20–24, 2017)
5. Badrinarayanan, S., Khurana, D., Ostrovsky, R., Visconti, I.: Unconditional UC-secure computation with (stronger-malicious) PUFs. In: Coron, J., Nielsen, J.B. (eds.) EUROCRYPT 2017, Part I. LNCS, vol. 10210, pp. 382–411. Springer, Heidelberg, Germany, Paris, France (Apr 30 – May 4, 2017)
6. Barak, B., Canetti, R., Nielsen, J.B., Pass, R.: Universally composable protocols with relaxed set-up assumptions. In: 45th FOCS. pp. 186–195. IEEE Computer Society Press, Rome, Italy (Oct 17–19, 2004)
7. Barak, B., Lindell, Y.: Strict polynomial-time in simulation and extraction. In: 34th ACM STOC. pp. 484–493. ACM Press, Montréal, Québec, Canada (May 19–21, 2002)
8. Barak, B., Prabhakaran, M., Sahai, A.: Concurrent non-malleable zero knowledge. In: 47th FOCS. pp. 345–354. IEEE Computer Society Press, Berkeley, CA, USA (Oct 21–24, 2006)
9. Bellare, M., Jakobsson, M., Yung, M.: Round-optimal zero-knowledge arguments based on any one-way function. In: Fumy, W. (ed.) EUROCRYPT'97. LNCS, vol. 1233, pp. 280–305. Springer, Heidelberg, Germany, Konstanz, Germany (May 11–15, 1997)
10. Bentov, I., Kumaresan, R.: How to use bitcoin to design fair protocols. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part II. LNCS, vol. 8617, pp. 421–439. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 17–21, 2014)
11. Blum, M.: How to prove a theorem so no one else can claim it. In: International Congress of Mathematicians. pp. 1444–1451 (1987)
12. Brzuska, C., Fischlin, M., Schröder, H., Katzenbeisser, S.: Physically uncloneable functions in the universal composition framework. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 51–70. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 14–18, 2011)
13. Canetti, R., Kushilevitz, E., Lindell, Y.: On the limitations of universally composable two-party computation without set-up assumptions. J. Cryptology 19(2), 135–167 (2006)
14. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: 42nd FOCS. pp. 136–145. IEEE Computer Society Press, Las Vegas, NV, USA (Oct 14–17, 2001)

15. Canetti, R., Dodis, Y., Pass, R., Walfish, S.: Universally composable security with global setup. In: Theory of Cryptography, 4th Theory of Cryptography Conference, TCC 2007, Amsterdam, The Netherlands, February 21-24, 2007, Proceedings. pp. 61–85 (2007)
16. Canetti, R., Fischlin, M.: Universally composable commitments. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 19–40. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 19–23, 2001)
17. Canetti, R., Goyal, V., Jain, A.: Concurrent secure computation with optimal query complexity. In: Gennaro, R., Robshaw, M.J.B. (eds.) CRYPTO 2015, Part II. LNCS, vol. 9216, pp. 43–62. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 16–20, 2015)
18. Canetti, R., Jain, A., Scafuro, A.: Practical UC security with a global random oracle. In: Ahn, G.J., Yung, M., Li, N. (eds.) ACM CCS 14. pp. 597–608. ACM Press, Scottsdale, AZ, USA (Nov 3–7, 2014)
19. Canetti, R., Kushilevitz, E., Lindell, Y.: On the limitations of universally composable two-party computation without set-up assumptions. In: EUROCRYPT. pp. 68–86 (2003)
20. Canetti, R., Lin, H., Pass, R.: Adaptive hardness and composable security in the plain model from standard assumptions. In: 51st FOCS. pp. 541–550. IEEE Computer Society Press, Las Vegas, NV, USA (Oct 23–26, 2010)
21. Canetti, R., Lindell, Y., Ostrovsky, R., Sahai, A.: Universally composable two-party and multi-party secure computation. In: 34th ACM STOC. pp. 494–503. ACM Press, Montréal, Québec, Canada (May 19–21, 2002)
22. Chandran, N., Goyal, V., Sahai, A.: New constructions for UC secure computation using tamper-proof hardware. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 545–562. Springer, Heidelberg, Germany, Istanbul, Turkey (Apr 13–17, 2008)
23. Choudhuri, A.R., Goyal, V., Jain, A.: Founding secure computation on blockchains. Cryptology ePrint Archive, Report 2019/253 (2019), https://eprint.iacr.org/2019/253
24. Choudhuri, A.R., Green, M., Jain, A., Kaptchuk, G., Miers, I.: Fairness in an unfair world: Fair multiparty computation from public bulletin boards. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 17. pp. 719–728. ACM Press, Dallas, TX, USA (Oct 31 – Nov 2, 2017)
25. Dachman-Soled, D., Fleischhacker, N., Katz, J., Lysyanskaya, A., Schröder, D.: Feasibility and infeasibility of secure computation with malicious PUFs. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part II. LNCS, vol. 8617, pp. 405–420. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 17–21, 2014)
26. Dolev, D., Dwork, C., Naor, M.: Non-malleable cryptography (extended abstract). In: 23rd ACM STOC. pp. 542–552. ACM Press, New Orleans, LA, USA (May 6–8, 1991)
27. Dwork, C., Naor, M., Sahai, A.: Concurrent zero-knowledge. In: 30th ACM STOC. pp. 409–418. ACM Press, Dallas, TX, USA (May 23–26, 1998)
28. Feige, U., Shamir, A.: Witness indistinguishable and witness hiding protocols. In: 22nd ACM STOC. pp. 416–426. ACM Press, Baltimore, MD, USA (May 14–16, 1990)
29. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings. pp. 186–194 (1986)

30. Garay, J.A., Kiayias, A., Leonardos, N.: The bitcoin backbone protocol: Analysis and applications. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015, Part II. LNCS, vol. 9057, pp. 281–310. Springer, Heidelberg, Germany, Sofia, Bulgaria (Apr 26–30, 2015)
31. Garay, J.A., Kiayias, A., Leonardos, N.: The bitcoin backbone protocol with chains of variable difficulty. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017, Part I. LNCS, vol. 10401, pp. 291–323. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 20–24, 2017)
32. Garg, S., Kumarasubramanian, A., Ostrovsky, R., Visconti, I.: Impossibility results for static input secure computation. In: CRYPTO. pp. 424–442 (2012)
33. Goldreich, O., Krawczyk, H.: On the composition of zero-knowledge proof systems. SIAM J. Comput. 25(1), 169–192 (1996)
34. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or A completeness theorem for protocols with honest majority. In: Aho, A. (ed.) 19th ACM STOC. pp. 218–229. ACM Press, New York City, NY, USA (May 25–27, 1987)
35. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof-systems (extended abstract). In: Proceedings of the 17th Annual ACM Symposium on Theory of Computing, May 6-8, 1985, Providence, Rhode Island, USA. pp. 291–304 (1985)
36. Goyal, R., Goyal, V.: Overcoming cryptographic impossibility results using blockchains. In: Kalai, Y., Reyzin, L. (eds.) TCC 2017, Part I. LNCS, vol. 10677, pp. 529–561. Springer, Heidelberg, Germany, Baltimore, MD, USA (Nov 12–15, 2017)
37. Goyal, V.: Positive results for concurrently secure computation in the plain model. In: 53rd FOCS. pp. 41–50. IEEE Computer Society Press, New Brunswick, NJ, USA (Oct 20–23, 2012)
38. Goyal, V., Gupta, D., Jain, A.: What information is leaked under concurrent composition? In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part II. LNCS, vol. 8043, pp. 220–238. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 18–22, 2013)
39. Goyal, V., Ishai, Y., Sahai, A., Venkatesan, R., Wadia, A.: Founding cryptography on tamper-proof hardware tokens. In: Micciancio, D. (ed.) TCC 2010. LNCS, vol. 5978, pp. 308–326. Springer, Heidelberg, Germany, Zurich, Switzerland (Feb 9–11, 2010)
40. Goyal, V., Jain, A., Ostrovsky, R.: Password-authenticated session-key generation on the internet in the plain model. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 277–294. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 15–19, 2010)
41. Goyal, V., Lin, H., Pandey, O., Pass, R., Sahai, A.: Round-efficient concurrently composable secure computation via a robust extraction lemma. In: Dodis, Y., Nielsen, J.B. (eds.) TCC 2015, Part I. LNCS, vol. 9014, pp. 260–289. Springer, Heidelberg, Germany, Warsaw, Poland (Mar 23–25, 2015)
42. Haitner, I., Horvitz, O., Katz, J., Koo, C.Y., Morselli, R., Shaltiel, R.: Reducing complexity assumptions for statistically-hiding commitment. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 58–77. Springer, Heidelberg, Germany, Aarhus, Denmark (May 22–26, 2005)
43. Hazay, C., Polychroniadou, A., Venkitasubramaniam, M.: Composable security in the tamper-proof hardware model under minimal complexity. In: Theory of Cryptography - 14th International Conference, TCC 2016-B, Beijing, China, October 31 - November 3, 2016, Proceedings, Part I. pp. 367–399 (2016)

44. Kalai, Y.T., Lindell, Y., Prabhakaran, M.: Concurrent general composition of secure protocols in the timing model. In: Gabow, H.N., Fagin, R. (eds.) 37th ACM STOC. pp. 644–653. ACM Press, Baltimore, MA, USA (May 22–24, 2005)
45. Katz, J.: Universally composable multi-party computation using tamper-proof hardware. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 115–128. Springer, Heidelberg, Germany, Barcelona, Spain (May 20–24, 2007)
46. Kiayias, A., Russell, A., David, B., Oliynykov, R.: Ouroboros: A provably secure proof-of-stake blockchain protocol. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017, Part I. LNCS, vol. 10401, pp. 357–388. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 20–24, 2017)
47. Kiayias, A., Zhou, H.S., Zikas, V.: Fair and robust multi-party computation using a global transaction ledger. In: Fischlin, M., Coron, J.S. (eds.) EUROCRYPT 2016, Part II. LNCS, vol. 9666, pp. 705–734. Springer, Heidelberg, Germany, Vienna, Austria (May 8–12, 2016)
48. Kilian, J.: Founding cryptography on oblivious transfer. In: 20th ACM STOC. pp. 20–31. ACM Press, Chicago, IL, USA (May 2–4, 1988)
49. Kilian, J., Petrank, E.: Concurrent and resettable zero-knowledge in poly-loalgorithm rounds. In: STOC. pp. 560–569 (2001)
50. Lindell, Y.: General composition and universal composability in secure multi-party computation. In: FOCS. pp. 394–403 (2003)
51. Lindell, Y.: Lower bounds for concurrent self composition. In: Naor, M. (ed.) TCC 2004. LNCS, vol. 2951, pp. 203–222. Springer, Heidelberg, Germany, Cambridge, MA, USA (Feb 19–21, 2004)
52. Lindell, Y.: Lower bounds and impossibility results for concurrent self composition. Journal of Cryptology 21(2), 200–249 (Apr 2008)
53. Micali, S., Pass, R.: Local zero knowledge. In: Proceedings of the 38th Annual ACM Symposium on Theory of Computing, Seattle, WA, USA, May 21-23, 2006. pp. 306–315 (2006), http://doi.acm.org/10.1145/1132516.1132561
54. Naor, M.: Bit commitment using pseudorandomness. Journal of Cryptology 4(2), 151–158 (Jan 1991)
55. Naor, M., Ostrovsky, R., Venkatesan, R., Yung, M.: Perfect zero-knowledge arguments for NP using any one-way permutation. Journal of Cryptology 11(2), 87–108 (Mar 1998)
56. Pass, R., Seeman, L., shelat, a.: Analysis of the blockchain protocol in asynchronous networks. In: Coron, J., Nielsen, J.B. (eds.) EUROCRYPT 2017, Part II. LNCS, vol. 10211, pp. 643–673. Springer, Heidelberg, Germany, Paris, France (Apr 30 – May 4, 2017)
57. Prabhakaran, M., Rosen, A., Sahai, A.: Concurrent zero knowledge with logarithmic round-complexity. In: 43rd FOCS. pp. 366–375. IEEE Computer Society Press, Vancouver, British Columbia, Canada (Nov 16–19, 2002)
58. Richardson, R., Kilian, J.: On the concurrent composition of zero-knowledge proofs. In: EUROCRYPT. pp. 415–431 (1999)
59. Yao, A.C.C.: How to generate and exchange secrets (extended abstract). In: 27th FOCS. pp. 162–167. IEEE Computer Society Press, Toronto, Ontario, Canada (Oct 27–29, 1986)