Founding Secure Computation on Blockchains

Arka Rai Choudhuri
Johns Hopkins University

Vipul Goyal
Carnergie Mellon University

Abhishek Jain
Johns Hopkins University

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 [ADMM14, BK14, CGJ$^+$17], or, one-time programs [GG17]). 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 [KZZ16, BMTZ17, BGK$^+$18]. In particular, our model is based on the global blockchain ledger model from Badertscher et. al [BMTZ17].

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 [CDPW07, CJS14a, HPV16]. A related model is the global Random Oracle model [CJS14a] 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 [GMR85], secure multiparty computation [Yao82, GMW87], and signature schemes in the random oracle model constructed via the Fiat-Shamir heuristic [FS86]. 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.[1] 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.

[1]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 [Can01a, CF01a, CKL03].

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 [GK96, FS90, BJY97].

Concurrent Secure Computation using Blockchains. Classical secure computation protocols such as [Yao82, GMW87] only achieve "stand-alone" security, and fail 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 [CKL06, Lin03, Lin04, Lin08, BPS06, Goy12, AGJ$^+$12, GKOV12]. 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 [CLOS02], the registered public-key model [BCNP04], the tamper-proof hardware model [Kat07, CGS08, GIS$^+$10], and the physically uncloneable functions model [BFSK11, DFK$^+$14, BKOV17]. 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 [Can01a] 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 [CF01a] 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 [CJS14a]) 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 [BMTZ17]. 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 [MP06]. 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*[2] 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.

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:

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

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 [BL02] 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 [BL02] 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 Appendix 6.

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 [Lin04], 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 [BPS06, Goy12, AGJ$^+$12, GKOV12].

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 [RK99, KP01, PRS02] 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. [GLP$^+$15] who were interested in a seemingly unrelated problem: designing commitment schemes that are secure w.r.t. chosen commitment attacks [CLP10]. 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 [GLP$^+$15]) 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, [GG17] 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 [Can01a] and global universal composability framework [CDPW07], 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, [GG17] do not consider interactive ZK proofs or any notion of secure multiparty computation.

In another recent work, [CGJ$^+$17] study the problem of fair multiparty computation in a "bulletin-board" model that can be implemented with blockchains. Similar to [GG17], however, their model provides the simulator the ability to control the blockchain. Prior to their work, multiple works [ADMM14, BK14] studied the problem of fairness with penalties using cryptocurrencies.

Several elegant works have conducted a formal study of various properties of blockchains [GKL15, PSs17, GKL17, KRDO17, BMTZ17]. Most relevant to our work is that of Badertscher et. al [BMTZ17] 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. [DNS98] 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 [KLP05] 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 5.3. We move on to our impossibility results starting with a lower bound on the round complexity of black-box zero-knowledge in section 6. In section 7 we show that allowing only the adversary access to the blockchain rules out zero knowledge. Finally, we show that UC commitments are impossible in section 8.

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 [KZZ16, BMTZ17, BGK$^+$18], 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 [BMTZ17] 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 $\mathsf{Validate}$ predicate and added to a buffer buffer. The buffer is meant to indicate those transactions that are not sufficiently deep to become permanent. The $\mathsf{Blockify}$ 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 $\mathsf{ExtendPolicy}$ that enforces a state-update policy restriction. See appendix A for further discussion on the $\mathsf{ExtendPolicy}$.

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 $\mathsf{windowSize}$. 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 $\vec{\mathcal{I}}_H^T$, 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 $\mathsf{predict\text{-}time}$ to ensure that the ideal world execution advances with the same pace (relative to the clock) as the protocol does. $\vec{\tau}_{\mathsf{state}}$ 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, $\mathsf{Validate}$, $\mathsf{ExtendPolicy}$, $\mathsf{Blockify}$, and $\mathsf{predict\text{-}time}$: $\mathsf{windowSize}$, $\mathsf{Delay} \in \mathbb{N}$. The functionality manages variables $\mathsf{state}, \mathsf{NxtBCbuffer}, \tau_L$, and $\vec{\tau}_{\mathsf{state}}$ as described above. The variables are initialized as follows: $\mathsf{state} := \vec{\tau}_{\mathsf{state}} := \mathsf{NxtBC} := \varepsilon$, $\mathsf{buffer} := \emptyset$, $\tau_L = 0$.

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
   $$\vec{N} = (\vec{N}_1, \cdots, \vec{N}_\ell) := \mathsf{ExtendPolicy}\left(\overrightarrow{\mathcal{I}}_H^T, \mathsf{state}, \mathsf{NxtBC}, \mathsf{buffer}, \vec{\tau}_{\mathsf{state}}\right)$$
   and if $\vec{N} \neq \varepsilon$ set $\mathsf{state} := \mathsf{state}||\mathsf{Blockify}(\vec{N}_1)|| \cdots ||\mathsf{Blockify}(\vec{N}_\ell)$ and $\vec{\tau}_{\mathsf{state}} := \vec{\tau}_{\mathsf{state}}||\tau_L^\ell$ 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 $\vec{N} \neq \varepsilon$, 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 $y := (x, \mathsf{uid}, \tau_L, P_i) \in \mathsf{buffer}$ with ID $\mathsf{uid} = \mathsf{uid}_i$ then set $\mathsf{listOfUid} := \mathsf{listOfUid}||\mathsf{uid}_i$.
   (c) Finally, set $\mathsf{NxtBC} := \mathsf{NxtBC}||(\mathsf{hflag}, \mathsf{listOfUid})$.

   – *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 $\mathsf{pt}_{i_j} := \widehat{\mathsf{pt}}_{i_j}$ for every $j \in [\ell]$.

(b) Otherwise set $\mathsf{pt}_{i_j} := |\mathsf{state}|$ 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 $\mathsf{state}_{i_j} := \mathsf{state}'_{i_j}$ for every $j \in [\ell]$.

The work of Badertscher et al [BMTZ17] show that under appropriate assumptions, Bitcoin realizes the ledger functionality described enforcing the ExtendPolicy described in appendix A. For convenience we've made a few syntactic changes to the $\mathcal{G}_{\mathsf{ledger}}$ functionality as described in [BMTZ17]:

– 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 [BMTZ17].

– 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 [BMTZ17].

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

– As described in [BMTZ17], 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 $\Delta := 4 \cdot \mathsf{windowSize}$ 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 [KZZ16, BMTZ17, BGK+18] 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 [CGJ+17, GG17] 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 [Can01a] and simulation in the global UC framework [CDPW07, CJS14a, HPV16].

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* $x \in L$*,*

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

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

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

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

  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 [Lin08]). 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 $\mathsf{ideal\ execution}$ 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 $\mathsf{start\text{-}session}$ 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 $\mathcal{G}_{\mathsf{ledger}}$-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 $\mathcal{G}_{\mathsf{ledger}}$-hybrid model.

3.3 Extractable Commitment Protocol $\langle C, R\rangle$

Let $\mathsf{com}(\cdot)$ denote the commitment function of a non-interactive perfectly binding string commitment scheme. Let n denote the security parameter. The commitment scheme $\langle C, R\rangle$ between the committer C and the receiver R is described as follows.

Commit Phase: This consists of two stages, namely, the Init stage and the Challenge-Response stage, described below:

<u>Init:</u> To commit to a n-bit string σ, C chooses $(\ell \cdot N)$ independent random pairs of n-bit strings $\{\alpha^0_{i,j}, \alpha^1_{i,j}\}^{\ell,N}_{i,j=1}$ such that $\alpha^0_{i,j} \oplus \alpha^1_{i,j} = \sigma$ for all $i \in [\ell], j \in [N]$. C commits to all these strings using com, with fresh randomness each time. Let $B \leftarrow \mathsf{com}(\sigma)$, and $A^0_{i,j} \leftarrow \mathsf{com}(\alpha^0_{i,j})$, $A^1_{i,j} \leftarrow \mathsf{com}(\alpha^1_{i,j})$ for every $i \in [\ell], j \in [N]$.

<u>Challenge-Response:</u> For every $j \in [N]$, do the following:

– $\mathsf{Challenge}$: R sends a random ℓ-bit challenge string $v_j = v_{1,j}, \ldots, v_{\ell,j}$.

– $\mathsf{Response}$: $\forall i \in [\ell]$, if $v_{i,j} = 0$, C opens $A^0_{i,j}$, else it opens $A^1_{i,j}$ by sending the decommitment information.

Open Phase: C opens all the commitments by sending the decommitment information for each one of them. R verifies the consistency of the revealed values. This completes the description of $\langle C, R\rangle$.

Notation. We introduce some terminology that will be used in the remainder of this paper. We refer to the committed value σ as the *preamble secret*. A slot_i of the commitment scheme consists of the i'th $\mathsf{Challenge}$ message from R and the corresponding $\mathsf{Response}$ message from C. Thus, in the above protocol, there are N slots.

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 start with a description of the graph hamiltonicity proof protocol, and then build our protocol atop this protocol.

4.1 Graph Hamiltonicity Zero-knowledge Proof

As a starting point, we describe the the Hamiltonicity proof system. In the simplest setting, the prover proves the existence of a Hamiltonian cycle in graph. The description of the protocol can be found in figure 1.

Properties. The protocol is zero-knowledge when a single instance is run, and thus witness indistinguishable. Witness indistinguishability holds even when the protocol is run in parallel. In addition, the above protocol satisfies the notion of *special simulation*, where the simulator can trivially simulate the proof if it is aware of the verifier's challenge. This, similar to witness indistinguishability, holds even when the protocol is run in parallel.

Roughly, the idea to simulate the protocol when we know the challenge prior to the first message sent by the prover, is the following:

– If the challenge bit is 0, then commit to adjacency matrix of the permuted graph $\pi(G)$.

– On the other hand, if the challenge bit is 1 commit to the complete graph K_n.

It is easy to see that the conditions for the corresponding challenge is met by the verifier. It also follows that the simulation holds when the protocol runs in parallel.

4.2 Our Protocol

The high level idea for our protocol is that the verifier commits to its challenge via the multi-round extractable commitment described in section 3.3, 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

Hamiltonicity proof system

Common Input: A directed graph $G = (V, E)$ with $n \overset{\text{def}}{=} |V|$.

Auxiliary Input for Prover: a directed Hamiltonian, $C \subset E$, in G.

1. Select a random permutation π, of the vertices V and using a statistically binding commitment scheme commit to the entries of the adjacency matrix of the permuted graph.
2. The verifier uniformly selects a bit σ and sends it to the prover.
3. The prover sends a message based on the value of σ it receives:
   - if $\sigma = 0$, the prover sends π along with the decommitment to all the values it had committed to earlier.
   - if $\sigma = 1$, the prover decommits only to entries in the permuted adjacency matrix $(\pi(u), \pi(v))$ with $(u, v) \in C$.
4. The verifier first checks if all the values decommitted to by the prover are valid (with respect to their corresponding commitment).
   - if $\sigma = 0$, the verifier checks if the revealed graph is the original graph permuted by π.
   - if $\sigma = 1$, the verifier checks if all the revealed values on the matrix is 1 and form a cycle.

   The verifier accepts if and only if both the initial checks, and the checks corresponding to the challenge verify .

Figure 1: Hamiltonicity proof system

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 2.

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

Completeness. The parameter k for protocol needs to be selected appropriately such that honest provers do not "time-out" prior to completion of the first phase (PRS preamble). Once this is ensured, the completeness of the protocol follows immediately from the completeness of the underlying Hamiltonicity proof system. We set k to be a constant satisfying this property.

A honest prover sends random challenges in the first phase and performs appropriate checks. In the second phase, it uses its witness to the statement to answer the verifier's challenge.

Soundness. On a high level, soundness follows from the statistical hiding property of the commitment scheme and the soundness of the underlying Hamiltonicity proof system. Because of statistical hiding, other than with negligible probability, even an unbounded prover cannot break the hiding of the commitment scheme. The values revealed by the verifier are randomly chosen strings, independent of the the challenge string. Thus, other than with negligible probability, no information about the challenge string is revealed by the end of Phase I. The rest of the protocol then relies on the soundness of the Hamiltonicity proof system, which simply requires that the challenge from the verifier is random and unknown to the prover.

For the reduction, we use prover $\mathcal{P}$, breaking the soundness of our protocol, to construct a prover $\mathcal{P}_{\mathsf{HC}}$ that breaks the soundness of the underlying Hamiltonicity proof system.

$\mathcal{P}_{\mathsf{HC}}$ behaves as follows:

- Commit to random challenge in the initial commitment for σ and likewise commits to $2mn$ random strings and send to $\mathcal{P}$. From the statistical hiding property, $\mathcal{P}$'s view is indistinguishable when $\mathcal{P}_{\mathsf{HC}}$ commits to random strings.

- Respond to $\mathcal{P}$'s challenges honestly.

- Forward $\mathcal{P}$'s first message in Phase II to the external challenge verifier.

- Let the challenge verifier respond with be $\widetilde{\sigma}$. We break the binding property of the the commitment scheme to find decommitments that are consistent with $\widetilde{\sigma}$. This applies to the initial commitment and the unopened commitments in Phase I.

- Send these decommitments to the prover.

- When the prover sends the third message in Phase II, relay it to the external challenge verifier. If $\mathcal{P}$ breaks soundness, so does $\mathcal{P}_{\mathsf{HC}}$.

Thus with only negligible probability difference, $\mathcal{P}_{\mathsf{HC}}$ breaks the soundness of the Hamiltonincity proof system.

Protocol $\mathsf{BCA\text{-}ZK}$

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 α, $\{\alpha^b_{\ell,p}\}_{b\in\{0,1\},\ell\in[n],p\in[m]}$.
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.

Figure 2: Protocol for zero-knowledge proof in the blockchain aware setting.

We note that we need statistical hiding in Phase I because an all powerful prover should not be able to guess the challenge bits. And we require statistical binding in the Hamiltonicity proof in Phase II because we don't want the prover to be able to break binding property to change the adjacency matrix that it committed to.

Zero-knowledge. We need to construct a simulator Sim such that the following ensembles are computationally indistinguishable in the $\mathcal{G}_{\mathsf{ledger}}$-hybrid model

$$\left\{\mathsf{view}_{\mathsf{V}}\left[\mathsf{P}(x,w) \leftrightarrow \mathsf{V}(x,z)\right]\right\}_{x\in L, w\in R_L(x), z\in\{0,1\}^*}, \left\{\mathsf{Sim}(x,z)\right\}_{x\in L, w\in R_L(x), z\in\{0,1\}^*}.$$

We now describe the simulator in Figure 3.

Simulator Sim

1. Send the first message of the two round statistically hiding commitment scheme.
2. At any point in the simulation, if on the main thread the size of the state of *either* the prover or verifier increases by k, then quit and output the view of the verifier. Unless otherwise specified, the simulator relays messages between the $\mathcal{G}_{\mathsf{ledger}}$ and the verifier.
3. For each $p \in [m]$,
   - Select a random challenge string for the main thread, and $n-1$ challenge strings for the look-ahead threads. All the threads are run in parallel.
   - On the look ahead thread, no messages to or from the $\mathcal{G}_{\mathsf{ledger}}$ are relayed.
   - A slot on the look ahead thread terminates if the slot completes successfully, or if the size of the local state increased after the threads were created (or if the adversary aborts).

   Note that there might be look-ahead threads from earlier slots running in parallel while the main thread may have progressed further.
4. If in each of the look-ahead threads, the slot is terminated prior to completion (all $m\cdot(n-1)$ of them), output $\bot_{\mathsf{rewind}}$ and exit.
5. Since the abort condition does not hold, other than with negligible probability, the challenge string has been obtained. Use the challenge string to simulate Phase II of the protocol.

Figure 3: Simulator for zero-knowledge proof in $\mathcal{G}_{\mathsf{ledger}}$-hybrid model.

In order to prove zero-knowledge, we consider an intermediate simulator Sim_1 that receives a witness w to the statement x. Sim_1 on input x, w, z proceeds identically to the honest execution except that in Phase I, before the challenge message from the prover P in each slot, we sample $n-1$ other challenge messages and fork look ahead threads to be run in parallel. Thus, there are a total of $m\cdot(n-1)$ look ahead threads.

The main and look ahead threads are run in parallel. They are executed almost identically but for the following difference: In the look ahead threads, none of the queries made to the $\mathcal{G}_{\mathsf{ledger}}$ are forwarded to the $\mathcal{G}_{\mathsf{ledger}}$. As described in section 2, the READ queries to $\mathcal{G}_{\mathsf{ledger}}$ are simulated within each thread without having to query $\mathcal{G}_{\mathsf{ledger}}$. In addition, the look ahead threads are run until either a block is created, or the slot completes, whichever happens first (the adversary may also abort).

At the completion of Phase I, if none of the slots on the look ahead thread completed before being terminated, output $\bot_{\mathsf{rewind}}$. (i.e. for none of the slots look ahead thread completed successfully with a response from the verifier.)

We note that to optimize the simulator, we can stop creation of look ahead threads if at any point a look ahead thread successfully completes its slot. But for simplicity of exposition, we do not do so here.

Now, we claim the following:

Claim 1. *The following are statistically indistinguishable in the $\mathcal{G}_{\mathsf{ledger}}$-hybrid model:*

$$\left\{ \mathsf{view_V} \left[\mathsf{P}(x,w) \leftrightarrow \mathsf{V}(x,z) \right] \right\}_{x \in L, w \in R_L(x), z \in \{0,1\}^*}, \left\{ \mathsf{Sim}_1(x,w,z) \right\}_{x \in L, w \in R_L(x), z \in \{0,1\}^*}$$

Proof. Since the main thread remains unchanged and the only difference is the creation of look ahead threads, all we need to argue is that $\perp_{\mathsf{rewind}}$ is output with negligible probability. To this end, we make the following additional claim.

Claim 2. *The probability with which* Sim_1 *outputs* $\perp_{\mathsf{rewind}}$ *is negligible.*

Proof. We prove this by contradiction. Assume that the probability is noticeable. We note that Sim_1 outputs $\perp_{\mathsf{rewind}}$ only if Phase I completes, and in all the look ahead threads, the simulator is forced to terminate it before it completes (i.e. if the size of the state increased before the look ahead threads were completed). It should be noted that since the adversary (with restrictions) controls when the state expands, it could potentially attempt to cause the state to increase at different rates in the look ahead thread. This makes no difference to the analysis since this can just be thought of as the adversary waiting for the state to expand before completion of the slot. For ease of notation, we shall call a slot good if the state did not expand after the slot started, and before its completion. This can refer to either a slot in the main thread, or in the look ahead thread. Thus, by this notation, Sim_1 outputs $\perp_{\mathsf{rewind}}$ only if the preamble completes and none of the slots on the look ahead threads are good.

To prove this, we shall consider yet another intermediate simulator Sim_1'. It is identical to Sim_1, but instead of picking the main thread challenge, and then the look-ahead challenges, pick n random strings and assign one of them to be the main thread challenge, and the remaining strings will be challenges on the look ahead thread. (The only difference is that in this case is that the main challenge is decided randomly from a set of challenges, as opposed to fixing a main thread and then choosing the look ahead threads.) Hence the probability that Sim_1' outputs $\perp_{\mathsf{rewind}}$ is identical to the probability that Sim_1 outputs $\perp_{\mathsf{rewind}}$.

To recall, Sim_1' outputs $\perp_{\mathsf{rewind}}$ if both the following conditions hold:

– $k' = (k + \mathsf{windowSize})$ blocks are not mined prior to completion of Phase I on the main thread. This ensures that at least $m - k'$ slots on the main thread are good.

– For each $p \in [m]$, every look ahead slot is not good.

Now let us look at the probability with which this happens. If the Phase I does not abort, then we know that there are at least $m - k$ slots on the main thread T that are good. And for the failure condition to be met, all the look-ahead slots at the same position terminate before completion (as the state expanded before it completed). Thus given the view prior to the slot, the corresponding good slot was chosen with probability $1/n$. We can repeat this analysis for each such good slot on the main thread. Given that we choose the random string to be the main thread challenge independently for each $p \in [m]$, for any choice of m strings of length n each we get:

$$\mathsf{Pr}[\mathsf{Sim}_1' \text{ outputs } \perp_{\mathsf{rewind}}] \leq \frac{1}{n^{m-k}} \leq n^k \cdot \mathsf{negl}(n) \tag{1}$$

when $m = \omega(1)$.

Thus, from our assumption that k is a constant, we get the probability that Sim_1' outputs $\perp_{\mathsf{rewind}}$ to be negligible. This is also gives us the requirement that the number of rounds in the protocol must be $\omega(1)$. □

This gives us a contradiction to our assumption that the probability with which Sim_1 outputs $\bot_{\mathsf{rewind}}$ is non-negligible. □

Now in the last step, the only change from Sim_1 is that we use the special simulation property of Hamiltonicity proof system. Now this becomes identical to our described simulator. View indistinguishability follows trivially as a consequence. Thus we get the following claim.

Claim 3. *The following are computationally indistinguishable in the $\mathcal{G}_{\mathsf{ledger}}$-hybrid model:*

$$\left\{\mathsf{Sim}_1(x, w, z)\right\}_{x \in L, w \in R_L(x), z \in \{0,1\}^*} \text{ and } \left\{\mathsf{Sim}(x, w, z)\right\}_{x \in L, w \in R_L(x), z \in \{0,1\}^*}$$

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 [CGJ15, GGJ13, GJO10] 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 [CGJ15, GGJ13, GJO10] 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 $\alpha^{r_{1,p}}_{1,p}, \ldots, \alpha^{r_{n,p}}_{n,p}$ to $\sigma^{r_{1,p}}_{1,p}, \ldots, \sigma^{r_{n,p}}_{n,p}$.

5.2 Simulation-Extraction Strategy

In this section, we will describe a simulation-extraction strategy. The goal of this strategy is to extract the value committed by an adversary in every session of multiple concurrent executions of the modified preamble described above. We present a new concurrent strategy where our starting point is the simulator described in [GLP+15]. The relevant description has been reproduced here.

The scheduling of messages is controlled by the adversary $\mathcal{A}^*$, and when $\mathcal{A}^*$ sends the p-th message of a session s, it immediately receives the next message of s. The only exception to this are the special messages relevant to the $\mathcal{G}_{\mathsf{ledger}}$. In this case, $\mathcal{A}^*$ sends a message to $\mathcal{G}_{\mathsf{ledger}}$, and can proceed with the other parts of the execution. Once $\mathcal{G}_{\mathsf{ledger}}$ sends a message to $\mathcal{A}^*$, $\mathcal{A}^*$ immediately receives it. This is not universal across all queries to $\mathcal{G}_{\mathsf{ledger}}$ since a READquery is immediately responded to by $\mathcal{G}_{\mathsf{ledger}}$, and needs to be handled accordingly.

The state of $\mathcal{A}^*$ at any given point consists of its view up to that point. The starting state of $\mathcal{A}^*$ is denoted by sta_0, which is its state before it receives its first message. In addition, $\mathsf{LIVE}(\mathsf{sta})$ denotes the set of live sessions when the execution is at the state sta. When the preamble starts, $\mathcal{A}^*$ sends a START along with its commitment, and on successful completion of the preamble expects a message of the form (END, α) from the simulator. As described in [GLP+15], we will require this to be the value $\mathcal{A}^*$ committed in preamble.

The simulator Sim receives as auxiliary input a string $z \in \{0,1\}^*$, and the security parameter n. It incorporates $\mathcal{A}^*$ as a black-box, and let $T = T(n)$ be the maximum number of sessions (of the modified preambles) that are started by $\mathcal{A}^*$. Unlike the setting described in [GLP+15], the external messages refer to those from $\mathcal{G}_{\mathsf{ledger}}$. We let the execution on look ahead threads proceed when the adversary has sent a message to $\mathcal{G}_{\mathsf{ledger}}$ without forwarding the query, and only abort the thread when the size of the state increases. Recall, from the description of $\mathcal{G}_{\mathsf{ledger}}$, the adversary also receives SUBMIT messages every time an honest party submits something to $\mathcal{G}_{\mathsf{ledger}}$. These are passed on to $\mathcal{A}^*$ on both the main and look ahead threads. This stems from the fact that these can be replayed in each thread of execution. On the look ahead thread, when $\mathcal{A}^*$ makes read queries, we follow the strategy outlined in the model in section 2. These READ queries are answered local to a thread without having to pass on queries to $\mathcal{G}_{\mathsf{ledger}}$ as follows: The adversary, and thus the simulator, is aware of the transactions/data sent by honest parties since $\mathcal{G}_{\mathsf{ledger}}$ sends them to the adversary as and when they arrive. These messages can be replayed within threads. For the transactions/data sent by the adversary, the simulator maintains a local buffer for each thread, collecting but not forwarding these queries. When the look ahead threads make a READ query, combine the state from the start of the thread (since we haven't aborted the thread, we're guaranteed that the state hasn't expanded), the honest SUBMIT queries along with the SUBMIT queries sent by the adversary **only local to that thread**. It is crucial to note that we do not make any changes to state.

Sim starts by setting $(1^n, z)$ on $\mathcal{A}^*$'s input tape, and a sufficiently long uniform string on its random tape. Sim then starts the recurse procedure:

$$(\mathsf{sta}, \mathcal{T}) \leftarrow \mathsf{extract}(2 \cdot \ell \cdot T, \mathsf{sta}_0, \emptyset, 1, \emptyset, 0)$$

The terminology used in the recurse procedure are:

- t is the block length (of a block of recursion), and the base case will occur when $t = 1$.
- sta refers to the starting state.
- $\mathcal{T}$ refers to the table containing solutions.
- f is used to denote if the execution lies on the main thread. $f = 1$ if and only if the block lies on the main thread of execution.
- aux refers to auxiliary tables that are used in special cases (see [GLP+15] for details).
- id refers to the identity of a block of execution used to uniquely identify it.

Throughout its execution, messages of recurse are forwarded back and forth between $\mathcal{G}_{\mathsf{ledger}}$ and $\mathcal{A}^*$. The final output of Sim is the first output of recurse, namely sta which is also known as the main thread of execution.

Procedure $\mathsf{recurse}(t, \mathsf{sta}, \mathcal{T}, \mathsf{f}, \mathsf{aux}, \mathsf{id})$

---

1. If $t = 1$, repeat the following keeping in mind that if at any point, a session "expires" (state size has increased by $k + \Delta$ since session information posted to $\mathcal{G}_{\mathsf{ledger}}$), abort the session:
   (a) If the next message is START, check if relevant information is in the state and start a new session s.
      - send $r \leftarrow_\$ \{0,1\}^n$ as the challenge of the first slot of s.
      - add entry $(s : 1, r, \_\_)$ to $\mathcal{T}$.
   (b) If the next message is the slot-i challenge of an existing session s.
      - send $r \leftarrow_\$ \{0,1\}^n$ as the slot-i challenge of s.
      - add entry $(s : i, r, \_\_)$ to $\mathcal{T}$.
   (c) If the next message is the slot-i response, say γ, of an existing session s.
      - If γ is a valid message.

```
            – update entry $(s : i, r, \_\_)$ to $(s : i, r, \gamma)$.
            – if $i = 2 \cdot \ell$, i.e., it is the last slot, send $(\mathsf{END}, \mathsf{extract}(s, \mathsf{id}, \mathcal{T}, \mathsf{aux}))$.
         – Otherwise, if $\gamma = \perp$, abort session $s$ and add $(s : \perp, \perp, \perp)$ to $\mathcal{T}$.
         – Update $\mathsf{sta}$ to be the current state of $\mathcal{A}^*$
         – return $(\mathsf{sta}, \mathcal{T})$.
   (d) If the next message is a message from $\mathcal{A}^*$ to the $\mathcal{G}_{\mathsf{ledger}}$
         – If $\mathsf{READ}$ message, drop the message, simulate the $\mathsf{READ}$ response (as described)
           and continue.
         – If $\mathsf{f} = 0$, i.e., it is a look ahead block, then drop the message (stop it from
           reaching $\mathcal{G}_{\mathsf{ledger}}$) and continue.
         – If $\mathsf{f} = 1$, i.e., it is the main thread, then forward the message to $\mathcal{G}_{\mathsf{ledger}}$, and
           continue.
   (e) If the next message is an expanded state from the $\mathcal{G}_{\mathsf{ledger}}$ to $\mathcal{A}^*$
         – If $\mathsf{f} = 0$, i.e., it is a look ahead block, then return $(\mathsf{sta}, \mathcal{T})$.
         – If $\mathsf{f} = 1$, i.e., it is the main thread, do the following:
            – Update $\mathsf{sta}$ to be the current state of $\mathcal{A}^*$
            – For every live session $s \in \mathsf{LIVE}(\mathsf{sta})$, do the following:
               – $\times_{s,\mathsf{id}} = \mathsf{true}$
               – for every block $\mathsf{id}'$ that contain the block $\mathsf{id}$, set $\times_{s,\mathsf{id}'} = \mathsf{true}$.
   (f) If other messages from $\mathcal{G}_{\mathsf{ledger}}$, pass to $\mathcal{A}^*$ and continue.

2. If $t > 1$,

      # Rewind the first half twice
   (a) $(\mathsf{sta}_1, \mathcal{T}_1) \leftarrow \mathsf{recurse}(t/2, \mathsf{sta}, \mathcal{T}, 0, \mathsf{aux}, \mathsf{id} \circ 1)$            [look-ahead block $C'$]
   (b) Let $\mathsf{aux}_2 := (\mathsf{aux}, \mathcal{T}_1 \setminus \mathcal{T})$,
       $(\mathsf{sta}_2, \mathcal{T}_2) \leftarrow \mathsf{recurse}(t/2, \mathsf{sta}, \mathcal{T}, \mathsf{f}, \mathsf{aux}_2, \mathsf{id} \circ 2)$           [main block $C$]

      # Rewind the second half twice
   (c) Let $\mathcal{T}^* := \mathcal{T}_1 \cup \mathcal{T}_2$,
       $(\mathsf{sta}_3, \mathcal{T}_3) \leftarrow \mathsf{recurse}(t/2, \mathsf{sta}_2, \mathcal{T}^*, 0, \mathsf{aux}, \mathsf{id} \circ 3)$            [look-ahead block $D'$]
   (d) Let $\mathsf{aux}_2 := (\mathsf{aux}, \mathcal{T}_1 \setminus \mathcal{T})$,
       $(\mathsf{sta}_4, \mathcal{T}_4) \leftarrow \mathsf{recurse}(t/2, \mathsf{sta}_2, \mathcal{T}^*, \mathsf{f}, \mathsf{aux}_4, \mathsf{id} \circ 4)$           [main block $D$]

   (e) return $(\mathsf{sta}_4, \mathcal{T}_3 \cup \mathcal{T}_4)$.
```

```
Procedure $\mathsf{extract}(s, \mathsf{id}, \mathcal{T}, \mathsf{aux})$

1. Attempt to extract a value for $s$ from $\mathcal{T}$.

2. If extraction fails, consider every block $\mathsf{id}_1$ for which $\times_{s,\mathsf{id}_1} = \mathsf{true}$.
```

- Let id'_1 be the sibling of id_1, with input/output tables $\mathcal{T}_{\mathsf{in}}, \mathcal{T}_{\mathsf{out}}$ respectively.
- Attempt to extract from $\mathsf{aux}_{\mathsf{id}'_1} := \mathcal{T}_{\mathsf{out}} \setminus \mathcal{T}_{\mathsf{in}}$;   (included in aux).

3. If all attempts fail, abort the simulation and return $\mathsf{ExtractFail}$.
   Otherwise return the extracted value.

Although we describe the simulator in terms of a recursive rewinding strategy, to ensure that the adversary does not gain any side channel leakage in the form of increase of state size for the state in $\mathcal{G}_{\mathsf{ledger}}$, all threads are run in parallel. The total number of threads is given by the recursion $h(2 \cdot \ell \cdot T(n)) \leq 4 \cdot h(\ell \cdot T(n))$ which gives us $h(2 \cdot \ell \cdot T(n)) \leq (\ell \cdot T(n))^2 \mathsf{poly}(n)$ threads. Thus if the total number of sessions and slots are polynomial, we have only polynomial many threads. And by our assumption of the simulator having access to arbitrary polynomial parallelism, we can run this threads in parallel.

Claim 4. Sim *succeeds other than with negligible probability.*

To prove this claim, we shall rely on the following robust extraction lemma. Informally, the lemma states that there exists a simulator that can extract the commitment made by the adversary without having to rewind an external protocol Π. The lemma states this by describing an online extractor $\mathcal{E}$ which can run in super-polynomial time to extract the committed value. We refer the reader to [GLP+15] for further details.

Lemma 1 (Robust Extraction Lemma [GLP+15])**.** *There exists an interactive Turing Machine* Sim *("robust simulator") such that for every $\mathcal{A}^*$, for every $\Pi = \langle B, A \rangle$, there exists a party $\mathcal{E}$ ("online extractor"), such that for every $n \in \mathbb{N}$, for every $x \in \mathsf{dom}_B(n)$, and every $z \in \{0,1\}^*$, the following conditions hold:*

1. **Validity constraint.** *For every output ν of $\mathsf{REAL}^{\mathcal{A}^*}_{\mathcal{E},\Pi}(n,x,z)$, for every* PRS *preamble s (appearing in ν) with transcript τ_s, if there exists a unique value $v \in \{0,1\}^n$ and randomness ρ such that $\mathsf{open}_{\mathsf{PRS}}(\tau_s, v, \rho) = 1$, then*

$$\alpha_s = v,$$

   where α_s is the value $\mathcal{E}$ sends at the completion of preamble s.

2. **Statistical simulation.** *If $k = k(n)$ and $\ell = \ell(n)$ denote the round complexities of Π and the* PRS *preamble respectively, then the statistical distance between distributions $\mathsf{REAL}^{\mathcal{A}^*}_{\mathcal{E},\Pi}(n,x,z)$ and $\mathsf{out}_s\left[B(1^n, x) \leftrightarrow \mathsf{Sim}^{\mathcal{A}^*}(1^n, z)\right]$ is given by:*

$$\Delta(n) \leq \frac{1}{2^{\Omega(\ell - k \cdot \log T(n))}},$$

   where $T(n)$ is the maximum number of total PRS *preambles between $\mathcal{A}^*$ and $\mathcal{E}$. Further the running time of* Sim *is $\mathsf{poly}(n) \cdot T(n)^2$.*

The reason the lemma doesn't directly apply to our setting is that there needs to be a "gap" between the number of slots of the preamble and the number of external messages. For instance, if the number of external messages are constant and the number of slots super-constant, we get a simulator that fails with negligible probability. Unfortunately in our setting, we can loosely upper bound the number of external state expansion messages by $T \cdot k$, which is no longer a constant.

To see why this our previous point about simulating the buffer is important important, each READ query and its corresponding response from $\mathcal{G}_{\mathsf{ledger}}$ will count as an external message by our definition. Since there is no prior bound on the number of such queries the adversary can make, we cannot hope to use the approaches listed above directly.

The proof of the above claim follows an argument of contradiction. Assume there is an adversary $\mathcal{A}$ for which the above simulator Sim fails, we shall construct using $\mathcal{A}$ and Sim, a new adversary $\widetilde{\mathcal{A}}$ such that there are only a constant number of external messages, but the simulator $\widetilde{\mathsf{Sim}}$ described in [GLP+15] fails with noticeable probability, thus violating the robust extraction lemma.

As a matter of technicality, and for ease of proof, the adversaries $\mathcal{A}$ and $\widetilde{\mathcal{A}}$ participate in slightly different preambles. $\mathcal{A}$ participates in preambles that have $2 \cdot \ell$ slots, while $\widetilde{\mathcal{A}}$ participates in preambles that have only ℓ slots. Thus, care must be taken when we $\widetilde{\mathcal{A}}$ forwards messages from $\mathcal{A}$.

Before we describe the constructed adversary, $\widetilde{\mathcal{A}}$, we introduce some notation. Let sta_i be the state of the adversary $\mathcal{A}$ on the main thread, when the START_i message is sent on this thread. We partitions the set $\mathsf{LIVE}(\mathsf{sta}_i)$ into two sets $\mathsf{HALF}(\mathsf{sta}_i)$ and $\mathsf{HALF}^c(\mathsf{sta}_i) = \mathsf{LIVE}(\mathsf{sta}_i) \setminus \mathsf{HALF}(\mathsf{sta}_i)$, where $\mathsf{HALF}(\mathsf{sta}_i)$ is the set of all preambles that have completed at least half (ℓ) of their slots (but not all of them since they're in $\mathsf{LIVE}(\mathsf{sta}_i)$). Intuitively, these preambles for these sessions already contain enough information on the tables generated by Sim, and thus there is no need to forward them to $\widetilde{\mathsf{Sim}}$.

Recall, in the preamble, the first message that is sent along with START, is the commitment to v and $2 \cdot \ell \cdot n$ pairs $\left(v^0_{i_1,i_2}, v^1_{i_1,i_2}\right)$ for , $i_1 \in [2 \cdot \ell], i_2 \in [n]$ such that $\forall i_1 \in [2 \cdot \ell], i_2 \in [n]$ $v^0_{i_1,i_2} \oplus v^1_{i_1,i_2} = v$. We denote this message for a session u to be ExtCom_u. Additionally, we denote by $\mathsf{ExtCom}_u[p : p+i_3]$ for $p + i_3 \leq 2 \cdot \ell$, the (truncated) commitment consisting of the commitment to v as before and of $(i_3 + 1) \cdot n$ pairs $\left(v^0_{i_1,i_2}, v^1_{i_1,i_2}\right)$ for $i_1 \in [p, p + i_3], i_2 \in [n]$. Thus $\mathsf{ExtCom}_u = \mathsf{ExtCom}_u[1 : 2 \cdot \ell]$.

Consider session $j \in \mathsf{HALF}^c(\mathsf{sta}_i)$. Let p^i_j be the slot of session j for which $\mathcal{A}$ received a challenge, but did not send a response. i.e. $p^i_j - 1$ slots were completed in session j. Given that $j \in \mathsf{HALF}^c(\mathsf{sta}_i)$, we have $1 \leq p^i_j \leq \ell$. We will use $\mathsf{ExtCom}_u[p^i_j + 1 : p^i_j + \ell]$ as the preamble commitment for $\widetilde{\mathcal{A}}$. This leaves slots $p^i_j + \ell + 1$ to $2 \cdot \ell$ when $p^i_j < \ell$ that need to be dealt with appropriately.

Given the notation and the idea described, we construct the adversary $\widetilde{\mathcal{A}}$ as follows:

adversary $\widetilde{\mathcal{A}}(1^n, z)$

1. Guess the slot $\widehat{i}$ for which the simulator Sim fails to extract from $\mathcal{A}$.
2. Initialize Sim with random coins and auxiliary input z. Sim will in turn initialize $\mathcal{A}$ to use in a black-box manner.
3. Prior to $\widetilde{\mathcal{A}}$ sending out any messages, it executes the interaction between $\mathcal{A}$ and Sim up to the point that $\mathcal{A}$ sends $\mathsf{START}_{\widehat{i}}$ on the main thread. It does so by relaying messages between Sim and $\mathcal{A}$ in each parallel execution till $\mathcal{A}$ outputs $\mathsf{START}_{\widehat{i}}$. During the execution of the interaction between Sim and $\mathcal{A}$, if Sim send messages to the $\mathcal{G}_{\mathsf{ledger}}$, $\widetilde{\mathcal{A}}$ sends this message too. When $\widetilde{\mathcal{A}}$ receives a state expansion message from the $\mathcal{G}_{\mathsf{ledger}}$, this is forwarded to Sim.

```
4. On receiving $\mathsf{START}_{\widehat{i}}$ and the corresponding $\mathsf{ExtCom}_{\widehat{i}}$,
      output $\mathsf{START}_{\widehat{i}}$ and $\mathsf{ExtCom}_{\widehat{i}}[1:\ell]$
      forward response received to $\mathcal{A}$.

5. For all other messages, the behavior is defined as follows:
   – If the next message from $\mathcal{A}$ is a $\mathsf{START}_j$ message for session $j$,
        output $\mathsf{START}_j$ and $\mathsf{ExtCom}_j[1:\ell]$
        forward response received to $\mathcal{A}$.
   – If the next message from $\mathcal{A}$ is the slot-$p$ response, say $\gamma$, of session $j$.
      – If $j \in \mathsf{HALF}(\mathsf{sta}_{\widehat{i}})$    #sessions with at least half the slots completed
         – if $p < 2\cdot\ell$
              respond internally with the challenge for slot $p+1$.
         – if $p = 2\cdot\ell$    #last slot of the main preamble
              use $\mathsf{Sim}$'s $\mathsf{extract}$ procedure to extract the value $v_j$ committed in session $j$.
              respond with $(\mathsf{END}_j, v_j)$
      – If $j \in \mathsf{HALF}^c(\mathsf{sta}_{\widehat{i}})$    #sessions with at least half the slots remaining
         – if $p = p_j^{\widehat{i}}$,
              output $\mathsf{START}_j$ and $\mathsf{ExtCom}_j[p+1:p+\ell]$.
              forward response received to $\mathcal{A}$.
         – if $p_j^{\widehat{i}} < p < p_j^{\widehat{i}} + \ell$,
              output $\gamma$.
              forward response received to $\mathcal{A}$.
         – if $p = p_j^{\widehat{i}} + \ell$,    #last slot of modified preamble
              output $\gamma$.
              on receiving $(\mathsf{END}_j, \widetilde{v_j})$, store $\widetilde{v_j}$ and respond internally with the challenge for slot $p+1$.
         – if $p_j^{\widehat{i}} + \ell < p < 2\cdot\ell$,
              respond internally with the challenge for slot $p$.
         – if $p = 2\cdot\ell$,    #last slot on main preamble
              use the stored value $\widetilde{v_j}$ and respond with $(\mathsf{END}_j, \widetilde{v_j})$.
      – If $j \notin \mathsf{LIVE}(\mathsf{sta}_{\widehat{i}})$    #sessions started after (and including) session $\widehat{i}$
         – if $p < \ell$,
              output $\gamma$.
              forward response received to $\mathcal{A}$.
         – if $p = \ell$,    #last slot of modified preamble
              output $\gamma$.
              on receiving $(\mathsf{END}_j, \widetilde{v_j})$, store $\widetilde{v_j}$ and respond internally with the challenge for slot $p+1$.
         – if $\ell < p < 2\cdot\ell$,
              respond internally with the challenge for slot $i+1$.
```

    - if $p = 2 \cdot \ell$, *#last slot on main preamble*
      use the stored value $\widetilde{v_j}$ and respond with $(\mathsf{END}_j, \widetilde{v_j})$.
  - If the next message from $\mathcal{A}$ is a message to $\mathcal{G}_{\mathsf{ledger}}$, output this query.
  - On receiving state expansion message from the $\mathcal{G}_{\mathsf{ledger}}$, forward to $\mathcal{A}$.

6. Quit on receiving response to the last slot in session $\widehat{i}$.

We would like to make a minor technical note at this point. While we stated that we wanted an adversary $\widetilde{\mathcal{A}}$ that receives only a constant number of external messages (state expansion messages from $\mathcal{G}_{\mathsf{ledger}}$), prior to $\widetilde{\mathcal{A}}$ sending any messages for the session, it waits for some external messages. Unfortunately, these may not be a constant. But, this makes little difference as the "core" of the transcript still contain only a constant number of external messages, and rewinding at these points do not affect the external messages sent early on.

5.3 The Protocol

In this section, we 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 [GJO10, GGJ13, CGJ15], except that we use the concurrently extractable commitment from Section 5.1. Indeed, the core ingredient of the compiler in [GJO10] (which is also used in [GGJ13, CGJ15]) 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 [GJO10] 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 in Section 5.3.2.

5.3.1 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 [Nao91] 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 [Blu87] in parallel, with the modification that the prover's commitments in the Hamiltonicity protocol are made using a statistically hiding commitment scheme [NOVY98, HHK+05] .

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 [Yao86, GMW87, Kil88].

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 [BPS06] 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. [BPS06] 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 [PRS02], we will use the N-round extractable commitment scheme $\langle C, R\rangle$ (described in Section 3.3). (b) Further, we require that the non-malleable commitment scheme being used in the protocol be public-coin w.r.t. receiver[3]. 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 $\langle C, R\rangle$ (Section 3.3) where V commits to a random string σ.

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.

[3]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 [DDN91] 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.

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.

5.3.2 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. [BPS06]. 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 $\mathsf{BCA\text{-}CONC}$

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 $\langle P^{\mathsf{sh}}_1, P^{\mathsf{sh}}_2\rangle$ 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 [GJO10, GGJ13, CGJ15]. The main difference is that due to the property of our concurrent extractor (Section 5.2), 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 [GJO10, GGJ13, CGJ15] achieve security in the so-called multiple-ideal-query model.

Our indistinguishability hybrids also follow in the same manner as in [GJO10, GGJ13, CGJ15]. There is one minor difference that we highlight. The hybrids of [GJO10, GGJ13, CGJ15] 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.

6 Impossibility of Constant Round Black-Box Zero Knowledge

In this section, we prove the impossibility of constant round zero-knowledge protocols w.r.t. black-box simulation in the blockchain-hybrid model. The starting point for our result is the beautiful work by Barak and Lindell[BL02] who showed that it is impossible to construct (non-trivial) constant round zero knowledge arguments or proofs with respect to black-box simulation if the simulator runs in strict polynomial time. At a high level, their impossibility result constructs a verifier with appropriate probability of abort in a given step such that with noticeable probability an honest execution will complete, but the simulator "runs out of time" when it attempts to gain any advantage over an honest prover. While they prove their impossibility result w.r.t. strict polynomial time black-box simulators, we extend their impossibility in our blockchain-hybrid model to stronger classes of simulators. In addition to simulators which run in strict polynomial time, we consider two additional classes of simulators: (1) simulators that can run in expected polynomial time, but have an a priori bounded memory, and hence at any time, can make a fixed polynomial number of queries to the adversary by running them in parallel; and (2) simulators that can run in expected polynomial time, but can make an unbounded number of queries to the adversary by running them in parallel. It is easy to see that the second class is stronger than the first. These results complement our positive results and demonstrate that our constructed protocols are tight.

For our setting, a *universally constant* round protocol will be such that the number of rounds are constant, and there is a constant upper bound on the size that the state can increase by during an execution of the protocol.

For a constant round protocol, let us first consider the simplest setting where the simulator can only run for some strict polynomial time. In addition, it can make a fixed number of polynomial queries in parallel to the adversary by making copies. While the polynomial can be arbitrary, it is fixed in advance. Thus, the effective number of computation steps for the simulator is a strict polynomial. We can thus apply directly the result from [BL02] to construct a verifier that prevents the simulator from gaining any advantage over an honest prover.

Bounded memory simulator. Now we let us consider an expected polynomial time simulator with bounded memory. This means that at any given time, the simulator may only have a strict polynomially bounded number of parallel executions. To invoke the result from Barak-Lindell [BL02], we need to describe a verifier strategy that forces the simulator to always run in strict polynomial time. Intuitively this means that for the given verifier strategy, any simulator running in super-polynomial time would leak side-channel information, i.e. the verifier would realize it was being run in super-polynomial time. We assume that a k-round protocol gives some upper bound r on how much the size of the state can expand during the execution of the protocol. This is enforced by the $\mathsf{ExtendPolicy}$ function. See appendix A.

We describe below an adversarial verifier strategy that forces the simulator to run in strict polynomial time:

– At the start of the protocol, the verifier obtains the state $\mathsf{state}_{\mathsf{V}}$ from $\mathcal{G}_{\mathsf{ledger}}$. Let the size of the state be $i_{\mathsf{V}} := |\mathsf{state}_{\mathsf{V}}|$.

– Behave according to underlying honest strategy.

– When the prover sends its last message p_ℓ, the verifier sends the signed transcript to $\mathcal{G}_{\mathsf{ledger}}$, and waits for $\mathsf{state}_{\mathsf{V}}$ to include the transcript. Let i^* be the index of the state that the transcript appear in. It then checks if $i^* - i_{\mathsf{V}}$ is larger than r, if so it outputs a special abort symbol $\perp$.

Note that unlike plain model, due to the presence of $\mathcal{G}_{\mathsf{ledger}}$, a verifier's view is not completely determined by the messages it receives from the prover.

Given that the interval between state being expanded is some polynomial (since the adversary controls this in a restricted manner), a super-polynomial running time would ensure that the increase in state size is not a constant. Thus, to avoid a trivial distinguisher that looks for the special abort, the simulator must run in strict polynomial time.

Unbounded memory. We now proceed to extend the impossibility to simulators that have no a priori bound on the number of parallel queries they make to the verifier, but still run in expected polynomial time. Here, we need to be wary of the simulator adaptively choosing to increase number of parallel queries.

While the result [BL02] does not apply to this setting, our results will build on their verifier strategy. Our adversarial verifier waits for constant time c_1 (here time is in terms of computation steps) on getting the query, answers correctly with prob ϵ and aborts with prob $1-\epsilon$. We will choose the probability such that $\epsilon > 1/q(n)$ for some polynomial $q(\cdot)$. The probability (over P's random coins) that an honest P causes the honest verifier to accept is $p = \epsilon^c$, where c is the number of rounds in the protocol. Since c is a constant, an honest prover will convince the verifier with noticeable probability as desired.

Let $\epsilon = \epsilon(n)$ be some value to be determined later. Let $\mathcal{H} = \{H_n\}_{n\in\mathbb{N}}$ be a family of $f(n)$-wise independent hash function, such that for every $h \in H_n, h : \{0,1\}^{\leq c\cdot m} \to \{0,1\}^n$, where $\{0,1\}^{\leq c\cdot m}$ denotes all strings of length at most $c \cdot m$. $f(n)$ will be determined later. c denotes the prover messages in the proof, and $m = m(n)$ denotes the longest prover message of the protocol. We present the detailed strategy below:

Verifier V

Random tape: (h, r) - h defines a function in H_n, and r is of the length of the random tape required by the honest verifier strategy **Input:** Series of prover messages $q = (\alpha_1, \ldots, \alpha_i)$

1. Step 1 - decide whether or not to abort:
   (a) Compute $h(q')$ for every prefix q' of q. That is, for every $j(1 \leq j \leq i)$, compute $h(\alpha_1, \ldots, \alpha_j)$.
   (b) Wait c_1 time and abort (by outputting the special symbol $\perp$), unless for every j, the first $\log\left(\frac{1}{\epsilon}\right)$ bits of $h(\alpha_1, \ldots, \alpha_j)$ are equal to 0.

   (Since the definition of V is by its next message function, we have to ensure that it replies to q only if it would not have aborted on messages sent prior to q in an interactive setting. This is carried out by checking that it would not have aborted on all prefixes of q.)
2. Step 2 - if not aborting,
   (a) Run the honest verifier on input $\alpha_1, \ldots, \alpha_i$ and with random tape r, and obtain its response β.

(b) Wait c_1 time and output β.

For simplicity of exposition, we assume that the verifier is always convinced. Our analysis naturally extends to the more general setting, where this may not hold.

Now consider the behavior of the same verifier in a simulation by Sim. Sim can issue any number of queries (in parallel) to the adversary at any point of time (by making copies). Consider epochs each of length c_1. In each epoch, the simulator may make a query at any point (and will get the answer only in the next epoch after time c_1 has elapsed). Represent the total number of queries the simulator makes in epoch i by q_i. Note that the Sim can only run for fixed constant c_2 epochs (else our earlier proof would apply). Let $\widetilde{p}$ be the probability that Sim output a transcript $(\alpha_1, \ldots, \alpha_c)$ such that the honest verifier accepts, and Sim received a non-aborting response for every prefix query. By the zero-knowledge property, $\widetilde{p}$ is at least $p - \mathsf{negl}(n)$. Specifically, because ϵ is chosen such that it is an inverse of a polynomial, we have $\widetilde{p} \geq p/2$. Since the simulator Sim is black-box, it does not know ϵ and can only observe the output of the queries.

We show that in each adaptive step, if Sim increases the number of parallel copies by more than an a priori bounded polynomial, it no longer runs in expected polynomial time.

– Consider epoch 1. During this epoch, by construction of the verifier, the simulator has not received any response from the verifier. Irrespective, it makes q_1 queries to the verifier. We claim that other than with negligible probability (over the coins of Sim), there exists a polynomial $f_1(n)$ such that $q_1 \leq f_1(n)$.

  If it is not the case, with noticeable probability Sim runs in super-polynomial time. Thus, Sim's expected running time would become super-polynomial as well, violating our restriction.

– We define E to be the even that the simulator has seen less than $c + 1$ queries answered up until that point.

– Now consider epoch 2. During this epoch, it receives responses to the queries made in epoch 1. Let us represent this by a "configuration" vector v_1 of dimension q_1 where each entry is either $\perp$ if it aborted, or $\top$ otherwise. We split our analysis of the number of queries q_2, in epoch 2, into two cases: **Case I.** If the number of non-aborts in v_1 are at most c, i.e. event E holds, then there exists a polynomial $f_2(n)$ such that for every configuration of v_1, $q_2 \leq f_2(n)$.

  Case II. There is no such f_2, i.e. there is a configuration v_1^* such that the number of queries q_2 is super-polynomial. If event E holds, then at most c "$\top$" symbols in every characteristic vector v_1. Thus, the total number of such characteristic vectors are

$$\binom{q_1}{0} + \binom{q_1}{1} + \cdots + \binom{q_1}{c}.$$

  Since c is a constant, the total number is a polynomial (other than with negligible probability). Thus each possible configuration v_1^* occurs with noticeable probability given event E holds. With noticeable probability Sim now runs in super-polynomial time. Thus, Sim's expected running time would become super-polynomial as well, violating our restriction.

  We notice that if the event E holds, other than with negligible probability the simulator makes q_2 number of queries, that are bounded by some polynomial.

– For $i \in \{2, \ldots, c_2\}$, we follow identically the analysis for q_2. But here, the characteristic vector v_i would include a symbol ($\perp$ or $\top$) for all queries made before epoch i started. Thus, its dimension would be $\sum_{i=1}^{i-1} q_i$.

– Thus, conditioned on E, the queries of Sim are bounded by $f_1(n), f_2(n), \ldots, f_{c_2}(n)$ in their respective epochs. We denote by $f := f(n) = \sum_{i=1}^{c_2} f_i(n)$ the bound on the number of queries made by the simulator given event E holds.

From the formula that for n Bernoulli trials where each trial succeeds with probability p, the probability of having k or more successes is at most $\binom{n}{k}p^k$. In our setting, a trial is a query message from the simulator, and the trial succeeds with probability ϵ. Thus the probability of having $c+1$ or more successes in f trials is at most $\binom{f}{c+1}\epsilon^{c+1}$.

Consider the event good where Sim outputs a verifying transcript, but receives exactly c non-aborting responses from the verifier. From the earlier discussion, Sim must output an accepting transcript with probability at least $\epsilon^c/2$. And since the probability of having $c+1$ or more successes in f trials is at most $\binom{f}{c+1}\epsilon^{c+1}$, the event good occurs with probability at least $\epsilon^c/2 - \binom{f}{c+1}\epsilon^{c+1}$. If we can set the value of ϵ to be such that this probability is noticeable, we can rely on the proof in [BL02]. Roughly, this is because we can then use Sim to convince an honest verifier with the same probability. We refer the reader to [BL02] for details.

To this end, we set $\epsilon = \frac{1}{4 \cdot \binom{f}{c+1}}$. Note that since f is a polynomial, and c is a constant, there exists a polynomial such that $\epsilon(n) = 1/p(n)$. Thus the difference between $\epsilon^c/2$ and $\binom{f}{c+1}\epsilon^{c+1}$ is $\epsilon^c/4$, which is an inverse polynomial as desired.

7 Black-Box Impossibility of Zero Knowledge in the Plain Model

In this section we shall show that in the plain model, it is impossible to construct a zero knowledge proof system against blockchain active adversaries (BCA).

Recall that the main advantage of a black-box simulator over an adversary is its ability to rewind the adversary. We now sketch a simple verifier strategy that makes it impossible for the simulator to successfully rewind the verifier.

For any polynomial ℓ, let $(p_1, v_1, p_2, v_2, \cdots, p_\ell, v_\ell, p_{\ell+1})$ denote the sequence of messages in a purported ZK protocol. Let $\mathsf{state}_{\mathsf{V}}$ be the $\mathcal{G}_{\mathsf{ledger}}$ state of the verifier. The adversarial verifier strategy is as follows:

– On receiving a prover message p_i, send the transcript $(p_1, v_1, p_2, v_2, \cdots, p_i)$ (along with the corresponding session id) to $\mathcal{G}_{\mathsf{ledger}}$, and wait for the size of the state to increase by $4 \cdot \mathsf{windowSize}$.

– Now, check for the if the following two conditions hold:

  – the sent message is on the state; and

  – there is no other transcript of the form $(\widetilde{p}_1, \widetilde{v}_1, \widetilde{p}_2, \widetilde{v}_2, \cdots, \widetilde{p}_i)$ for the same session id anywhere in the state.

  Only if both the checks pass, proceed by sending the honest response v_i. It is not explicitly stated, but the verifier maintains the state of the blockchain, and updates its state as it receives valid blocks.

It is clear that in the interaction with the honest prover, the verifier behaves honestly and completes the protocol.

Now, consider any simulator. For the simulator to gain any advantage over a prover, the simulator must receive responses from the verifier for at least two different i-th messages p_i and $\widetilde{p}_i$ for some i. From the verifier's strategy, we know that in both cases the verifier sends the transcript to $\mathcal{G}_{\mathsf{ledger}}$ and waits for the state to expand sufficiently before responding. If the posted transcript does not appear on the state after the specified wait period, the verifier aborts. This ensures that the simulator cannot rewind the verifier without detection, and thus gains no advantage over an honest prover.

8 UC Impossibility

In this section, we will show that it is impossible to achieve universal composition security [Can01b] in the blockchain model. This might seem surprising given the positive results of [CJS14b] in the non-programmable global random oracle (RO) model, which seems to have a similar flavor.

A crucial difference in these settings is that in the case of the non-programmable global RO model, the posts by the honest parties to the oracle, and their corresponding responses, are private. This is not true for the blockchain model as the queries are public to everyone. Importantly, this in turn implies that the environment sees the actual blockchain, and not a view that the simulator presents to the environment. This prevents the simulator from changing any query made by the adversary to the blockchain. If the simulator does change the query, the environment will always be able to tell the difference between the real and ideal execution. Intuitively, seeing the queries to blockchain is not unique to the adversary, and thus does not constitute a new capability.

For this impossibility, we work in a slightly different model where we allow parties to make an *outstanding query request* to the blockchain, asking for outstanding queries that have not been included in the block. This closely models what happens in practice since parties broadcast the information they want to post to the blockchain to anyone connected to them. This explicitly models the fact that parties can see queries made to the blockchain. We show the impossibility of UC secure commitments [CF01b] in the blockchain model. We define the ideal functionality of the commitment here, and refer to the reader to [Can01b, CF01b] for details and definitions.

Functionality $\mathbf{F}_{\mathsf{COM}}$

$\mathbf{F}_{\mathsf{COM}}$ proceeds as follows with parties $P_1, \ldots, P_n$ and an adversary Sim.

1. Upon receiving a value $(\texttt{Commit}, sid, P_i, P_j, b)$ from P_i, where $b \in \{0,1\}$, record the value b and send the message $(\texttt{Receipt}, sid, P_i, P_j)$ to P_j and Sim.
2. Upon receiving a value $(\texttt{Open}, sid, P_i, P_j)$ from P_i, proceed as follows: If some value b was previously recorded, then send the message $(\texttt{Open}, sid, P_i, P_j, b)$ to P_j and Sim and halt. Otherwise halt.

Figure 4: The ideal commitment functionality

Let us assume to the contrary, that there exists a simulator Sim for an adversarial committer. Then, we shall describe an adversarial receiver that uses Sim to extract the value committed by the committer. The adversarial committer works as follows:

1. Initialize Sim.

2. Behave as a "fake committer" by passing any message sent by the committer to Sim, and vice-versa.

3. As and when blocks are received from the blockchain, pass them on to Sim.

4. When the committer makes a query q to the blockchain, this query is sent to Sim on behalf of the "fake committer".

5. Since Sim cannot change the query made by the adversary, when it makes the same query q to the blockchain, we block this query. But as before, when a subsequent block is mined, we pass it on to Sim.

6. When Sim makes a query to the UC commitment ideal functionality, stop and output this as the committed value.

9 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

[ADMM14] Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and Lukasz Mazurek. Secure multiparty computations on bitcoin. In *2014 IEEE Symposium on Security and Privacy, SP 2014, Berkeley, CA, USA, May 18-21, 2014*, pages 443–458, 2014.

[AGJ+12] Shweta Agrawal, Vipul Goyal, Abhishek Jain, Manoj Prabhakaran, and Amit Sahai. New impossibility results for concurrent composition and a non-interactive completeness theorem for secure computation. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 443–460, Santa Barbara, CA, USA, August 19–23, 2012. Springer, Heidelberg, Germany.

[BCNP04] Boaz Barak, Ran Canetti, Jesper Buus Nielsen, and Rafael Pass. Universally composable protocols with relaxed set-up assumptions. In *45th FOCS*, pages 186–195, Rome, Italy, October 17–19, 2004. IEEE Computer Society Press.

[BFSK11] Christina Brzuska, Marc Fischlin, Heike Schröder, and Stefan Katzenbeisser. Physically uncloneable functions in the universal composition framework. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 51–70, Santa Barbara, CA, USA, August 14–18, 2011. Springer, Heidelberg, Germany.

[BGK+18] Christian Badertscher, Peter Gaži, Aggelos Kiayias, Alexander Russell, and Vassilis Zikas. Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability. Cryptology ePrint Archive, Report 2018/378, 2018. https://eprint.iacr.org/2018/378.

[BJY97] Mihir Bellare, Markus Jakobsson, and Moti Yung. Round-optimal zero-knowledge arguments based on any one-way function. In Walter Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 280–305, Konstanz, Germany, May 11–15, 1997. Springer, Heidelberg, Germany.

[BK14] Iddo Bentov and Ranjit Kumaresan. How to use bitcoin to design fair protocols. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 421–439, Santa Barbara, CA, USA, August 17–21, 2014. Springer, Heidelberg, Germany.

[BKOV17] Saikrishna Badrinarayanan, Dakshita Khurana, Rafail Ostrovsky, and Ivan Visconti. Unconditional UC-secure computation with (stronger-malicious) PUFs. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part I*, volume 10210 of *LNCS*, pages 382–411, Paris, France, April 30 – May 4, 2017. Springer, Heidelberg, Germany.

[BL02] Boaz Barak and Yehuda Lindell. Strict polynomial-time in simulation and extraction. In *34th ACM STOC*, pages 484–493, Montréal, Québec, Canada, May 19–21, 2002. ACM Press.

[Blu87] Manual Blum. How to prove a theorem so no one else can claim it. In *International Congress of Mathematicians*, pages 1444–1451, 1987.

[BMTZ17] Christian Badertscher, Ueli Maurer, Daniel Tschudi, and Vassilis Zikas. Bitcoin as a transaction ledger: A composable treatment. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 324–356, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Heidelberg, Germany.

[BPS06] Boaz Barak, Manoj Prabhakaran, and Amit Sahai. Concurrent non-malleable zero knowledge. In *47th FOCS*, pages 345–354, Berkeley, CA, USA, October 21–24, 2006. IEEE Computer Society Press.

[Can01a] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, pages 136–145, 2001.

[Can01b] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145, Las Vegas, NV, USA, October 14–17, 2001. IEEE Computer Society Press.

[CDPW07] Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. 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*, pages 61–85, 2007.

[CF01a] Ran Canetti and Marc Fischlin. Universally composable commitments. In *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*, pages 19–40, 2001.

[CF01b] Ran Canetti and Marc Fischlin. Universally composable commitments. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 19–40, Santa Barbara, CA, USA, August 19–23, 2001. Springer, Heidelberg, Germany.

[CGJ15] Ran Canetti, Vipul Goyal, and Abhishek Jain. Concurrent secure computation with optimal query complexity. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 43–62, Santa Barbara, CA, USA, August 16–20, 2015. Springer, Heidelberg, Germany.

[CGJ+17] Arka Rai Choudhuri, Matthew Green, Abhishek Jain, Gabriel Kaptchuk, and Ian Miers. Fairness in an unfair world: Fair multiparty computation from public bulletin boards. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 17*, pages 719–728, Dallas, TX, USA, October 31 – November 2, 2017. ACM Press.

[CGS08] Nishanth Chandran, Vipul Goyal, and Amit Sahai. New constructions for UC secure computation using tamper-proof hardware. In Nigel P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 545–562, Istanbul, Turkey, April 13–17, 2008. Springer, Heidelberg, Germany.

[CJS14a] Ran Canetti, Abhishek Jain, and Alessandra Scafuro. Practical UC security with a global random oracle. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*, pages 597–608, 2014.

[CJS14b] Ran Canetti, Abhishek Jain, and Alessandra Scafuro. Practical UC security with a global random oracle. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM CCS 14*, pages 597–608, Scottsdale, AZ, USA, November 3–7, 2014. ACM Press.

[CKL03] Ran Canetti, Eyal Kushilevitz, and Yehuda Lindell. On the limitations of universally composable two-party computation without set-up assumptions. In *EUROCRYPT*, pages 68–86, 2003.

[CKL06] R. Canetti, E. Kushilevitz, and Y. Lindell. On the limitations of universally composable two-party computation without set-up assumptions. *J. Cryptology*, 19(2):135–167, 2006.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *34th ACM STOC*, pages 494–503, Montréal, Québec, Canada, May 19–21, 2002. ACM Press.

[CLP10] Ran Canetti, Huijia Lin, and Rafael Pass. Adaptive hardness and composable security in the plain model from standard assumptions. In *51st FOCS*, pages 541–550, Las Vegas, NV, USA, October 23–26, 2010. IEEE Computer Society Press.

[DDN91] Danny Dolev, Cynthia Dwork, and Moni Naor. Non-malleable cryptography (extended abstract). In *23rd ACM STOC*, pages 542–552, New Orleans, LA, USA, May 6–8, 1991. ACM Press.

[DFK+14] Dana Dachman-Soled, Nils Fleischhacker, Jonathan Katz, Anna Lysyanskaya, and Dominique Schröder. Feasibility and infeasibility of secure computation with malicious PUFs. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 405–420, Santa Barbara, CA, USA, August 17–21, 2014. Springer, Heidelberg, Germany.

[DNS98] Cynthia Dwork, Moni Naor, and Amit Sahai. Concurrent zero-knowledge. In *30th ACM STOC*, pages 409–418, Dallas, TX, USA, May 23–26, 1998. ACM Press.

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings*, pages 186–194, 1986.

[FS90] Uriel Feige and Adi Shamir. Witness indistinguishable and witness hiding protocols. In *22nd ACM STOC*, pages 416–426, Baltimore, MD, USA, May 14–16, 1990. ACM Press.

[GG17] Rishab Goyal and Vipul Goyal. Overcoming cryptographic impossibility results using blockchains. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part I*, volume 10677 of *LNCS*, pages 529–561, Baltimore, MD, USA, November 12–15, 2017. Springer, Heidelberg, Germany.

[GGJ13] Vipul Goyal, Divya Gupta, and Abhishek Jain. What information is leaked under concurrent composition? In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 220–238, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Heidelberg, Germany.

[GIS$^+$10] Vipul Goyal, Yuval Ishai, Amit Sahai, Ramarathnam Venkatesan, and Akshay Wadia. Founding cryptography on tamper-proof hardware tokens. In Daniele Micciancio, editor, *TCC 2010*, volume 5978 of *LNCS*, pages 308–326, Zurich, Switzerland, February 9–11, 2010. Springer, Heidelberg, Germany.

[GJO10] Vipul Goyal, Abhishek Jain, and Rafail Ostrovsky. Password-authenticated session-key generation on the internet in the plain model. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 277–294, Santa Barbara, CA, USA, August 15–19, 2010. Springer, Heidelberg, Germany.

[GK96] Oded Goldreich and Hugo Krawczyk. On the composition of zero-knowledge proof systems. *SIAM J. Comput.*, 25(1):169–192, 1996.

[GKL15] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 281–310, Sofia, Bulgaria, April 26–30, 2015. Springer, Heidelberg, Germany.

[GKL17] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol with chains of variable difficulty. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 291–323, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Heidelberg, Germany.

[GKOV12] Sanjam Garg, Abishek Kumarasubramanian, Rafail Ostrovsky, and Ivan Visconti. Impossibility results for static input secure computation. In *CRYPTO*, pages 424–442, 2012.

[GLP$^+$15] Vipul Goyal, Huijia Lin, Omkant Pandey, Rafael Pass, and Amit Sahai. Round-efficient concurrently composable secure computation via a robust extraction lemma. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part I*, volume 9014 of *LNCS*, pages 260–289, Warsaw, Poland, March 23–25, 2015. Springer, Heidelberg, Germany.

[GMR85] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. 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*, pages 291–304, 1985.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th ACM STOC*, pages 218–229, New York City, NY, USA, May 25–27, 1987. ACM Press.

[Goy12] Vipul Goyal. Positive results for concurrently secure computation in the plain model. In *53rd FOCS*, pages 41–50, New Brunswick, NJ, USA, October 20–23, 2012. IEEE Computer Society Press.

[HHK$^+$05] Iftach Haitner, Omer Horvitz, Jonathan Katz, Chiu-Yuen Koo, Ruggero Morselli, and Ronen Shaltiel. Reducing complexity assumptions for statistically-hiding commitment. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 58–77, Aarhus, Denmark, May 22–26, 2005. Springer, Heidelberg, Germany.

[HPV16] Carmit Hazay, Antigoni Polychroniadou, and Muthuramakrishnan Venkitasubramaniam. 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*, pages 367–399, 2016.

[Kat07] Jonathan Katz. Universally composable multi-party computation using tamper-proof hardware. In Moni Naor, editor, *EUROCRYPT 2007*, volume 4515 of *LNCS*, pages 115–128, Barcelona, Spain, May 20–24, 2007. Springer, Heidelberg, Germany.

[Kil88] Joe Kilian. Founding cryptography on oblivious transfer. In *20th ACM STOC*, pages 20–31, Chicago, IL, USA, May 2–4, 1988. ACM Press.

[KLP05] Yael Tauman Kalai, Yehuda Lindell, and Manoj Prabhakaran. Concurrent general composition of secure protocols in the timing model. In Harold N. Gabow and Ronald Fagin, editors, *37th ACM STOC*, pages 644–653, Baltimore, MA, USA, May 22–24, 2005. ACM Press.

[KP01] Joe Kilian and Erez Petrank. Concurrent and resettable zero-knowledge in poly-loalgorithm rounds. In *STOC*, pages 560–569, 2001.

[KRDO17] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 357–388, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Heidelberg, Germany.

[KZZ16] Aggelos Kiayias, Hong-Sheng Zhou, and Vassilis Zikas. Fair and robust multi-party computation using a global transaction ledger. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 705–734, Vienna, Austria, May 8–12, 2016. Springer, Heidelberg, Germany.

[Lin03] Yehuda Lindell. General composition and universal composability in secure multi-party computation. In *FOCS*, pages 394–403, 2003.

[Lin04] Yehuda Lindell. Lower bounds for concurrent self composition. In Moni Naor, editor, *TCC 2004*, volume 2951 of *LNCS*, pages 203–222, Cambridge, MA, USA, February 19–21, 2004. Springer, Heidelberg, Germany.

[Lin08] Yehuda Lindell. Lower bounds and impossibility results for concurrent self composition. *Journal of Cryptology*, 21(2):200–249, April 2008.

[MP06] Silvio Micali and Rafael Pass. Local zero knowledge. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing, Seattle, WA, USA, May 21-23, 2006*, pages 306–315, 2006.

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

[NOVY98] Moni Naor, Rafail Ostrovsky, Ramarathnam Venkatesan, and Moti Yung. Perfect zero-knowledge arguments for NP using any one-way permutation. *Journal of Cryptology*, 11(2):87–108, March 1998.

[PRS02] Manoj Prabhakaran, Alon Rosen, and Amit Sahai. Concurrent zero knowledge with logarithmic round-complexity. In *43rd FOCS*, pages 366–375, Vancouver, British Columbia, Canada, November 16–19, 2002. IEEE Computer Society Press.

[PSs17] Rafael Pass, Lior Seeman, and abhi shelat. Analysis of the blockchain protocol in asynchronous networks. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part II*, volume 10211 of *LNCS*, pages 643–673, Paris, France, April 30 – May 4, 2017. Springer, Heidelberg, Germany.

[RK99] Ransom Richardson and Joe Kilian. On the concurrent composition of zero-knowledge proofs. In *EUROCRYPT*, pages 415–431, 1999.

[Yao82] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *FOCS*, pages 160–164, 1982.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167, Toronto, Ontario, Canada, October 27–29, 1986. IEEE Computer Society Press.

A Extend Policy for Bitcoin

We copy verbatim from [BMTZ17] the properties ensured by ExtendPolicy:

1. The speed of the ledger is not too slow. This is implemented by defining an upper bound $\mathsf{maxTime}_{\mathsf{window}}$ on the time interval within which at least windowSize state blocks have to be added. This is known as minimal chain-growth.

2. The speed of the ledger is not too fast. This is implemented by defining a lower bound $\mathsf{minTime}_{\mathsf{window}}$ on the time interval such that the adversary is not allowed to propose new blocks if windowSize or more blocks have already been added during that interval.

3. The adversary cannot create too many blocks with arbitrary (but valid) contents. This is formally enforced by defining an upper bound η on the number of these so-called adversarial blocks within a sequence of state blocks. This is knows as chain quality. Formally, this is

enforced by requiring that a certain fraction of blocks need to satisfy higher quality standards (to model blocks that were honestly generated).

4. Last, but not the least, ExtendPolicyguarantess that if a transaction is “old enough”, and still valid with respect to the actual state, then it is included into the state. This is a week form of guaranteeing that a transaction will make it into the state unless it is in conflict.

The formal description can be found in [BMTZ17].