Time-Traveling Simulators Using Blockchains and Their Applications[⋆]

Vipul Goyal[1,2], Justin Raizes[1], and Pratik Soni[1]

[1] Carnegie Mellon University
vipul@cmu.edu, jraizes@andrew.cmu.edu, psoni@andrew.cmu.edu
[2] NTT Research

Abstract. Blockchain technology has the potential of transforming cryptography. We study the problem of round-complexity of zero-knowledge, and more broadly, of secure computation in the *blockchain-hybrid model*, where all parties can access the blockchain as an oracle.

We study zero-knowledge and secure computation through the lens of a new security notion where the simulator is given the ability to "time-travel" or more accurately, to look into the future states of the blockchain and use this information to perform simulation. Such a *time-traveling simulator* gives a novel security guarantee of the following form: *whatever the adversary could have learnt from an interaction, it could have computed on its own shortly into the future (e.g., a few hours from now).*

We exhibit the power of time-traveling simulators by constructing round-efficient protocols in the blockchain-hybrid model. In particular, we construct:
1. Three-round zero-knowledge (ZK) argument for NP with a polynomial-time black-box time-traveling simulator.
2. Three-round secure two-party computation (2PC) for any functionality with a polynomial-time black-box time-traveling simulator for both parties.

In addition to standard cryptographic assumptions, we rely on natural hardness assumptions for Proof-of-Work based blockchains. In comparison, in the plain model, three-round protocols with black-box simulation are impossible, and constructions with non-black-box simulation for ZK require novel cryptographic assumptions while no construction for three-round 2PC is known. Our three-round 2PC result relies on a new, two-round extractable commitment that admits a time-traveling extractor.

1 Introduction

We are seeing a surging interest in blockchain as a vibrant technology: it powers cryptocurrencies like Bitcoin [38], Ethereum [47] and ZCash [10] which are valued in hundreds of billions of dollars, and it enables removing the central point of trust in existing banking infrastructure.

In a blockchain protocol, the goal of parties is to maintain a globally ordered sequence of records that are referred to as *blocks*. New blocks can only be added via a special mining procedure that simulates a puzzle solving race between participants and can be run by any participant or *miner*. For example, in Bitcoin, the puzzle is selected so that a block is mined (on average) every 10 minutes. The two widely used puzzles are Proof-of-Work (PoW) and Proof-of-Stake (PoS). A sequence of works [23,34,19,42,40,43,44,24,25] have analysed the security of current PoW- and PoS-based blockchains, and also proposed new ones. Understanding the security properties offered by these blockchains, and applications that such properties enable is a rapidly progressing direction.

Since its inception, blockchain has proven a successful testbed for a number of beautiful concepts in cryptography such as zero-knowledge proofs, digital signatures and hash functions. But, the

[⋆] A preliminary version of this work appears at ITCS 2022. This is the full version.

blockchain itself offers several attractive properties which are conducive to build cryptography, e.g., provides a decentralized alternative to a trusted setup, which traditionally is performed by a central trusted authority and is inherent for constructing important cryptographic primitives like non-interactive zero-knowledge. In fact, several works view blockchains as an "enabler" for cryptography, much like one-way functions, trapdoor permutations and indistinguishability obfuscation, leading to constructions of concurrent multi-party computation [17], one-time programs [30], time-lock encryption [37] and fair secure multi-party computation [18,3,11]. In this work, we take this direction and focus on understanding the round-complexity of two fundamental cryptographic objects: zero-knowledge proofs and secure computation. To study zero-knowledge and secure computation in the world of blockchains, we study the *blockchain-hybrid model*.

The Blockchain-Hybrid Model. Here, the blockchain is modelled as a ledger functionality, and all participants of the cryptographic protocol can access it as an oracle. In particular, the parties (including the adversary) can access the blockchain by posting and reading content, but no single party has any control over the blockchain. Our modeling follows the work of [17] who introduce the blockchain-hybrid model, who in turn adopt the blockchain ledger model from [5].

Compared to the plain model, the presence of blockchain as an oracle is, as we will shortly see, quite empowering. But we emphasize that this is distinct from trusted-setup models explored in cryptography where some trusted party samples a reference string from some good distribution, and then all parties are given access to the string. In particular, in the security proofs, the reduction/simulator is given the power to choose the reference string (e.g., sample along with a trapdoor to aid simulation). *This is in sharp contrast with the blockchain-hybrid model where the reduction/simulator has no control over the blockchain.*

Simulation-security in the Blockchain-Hybrid Model. Zero-knowledge proofs are a cornerstone of modern cryptography. Informally, they are interactive protocols between two parties *prover* and a *verifier*, where the prover is trying to convince the verifier that some string x (also known as an instance) belongs to a language. We want two security properties: (a) Soundness - no cheating prover can convince the verifier on false statements, and (b) Zero-knowledge. The philosophy behind the notion of Zero-Knowledge [29] is that: *whatever an adversary can learn from the proof of a true statement, it could have learnt by itself in polynomial-time.* This counter-intuitive notion of security is formalized by asking the existence of an efficient *simulator* that can simulate the view of any malicious verifier.

While some works [30,17] have considered zero-knowledge with blockchains, we take a different philosophical approach. We refer the reader to Section 2.3 for more details on these works.

Time-Traveling Simulation. We present a novel idea to quantify the knowledge an adversary learns. Following is our philosophy: *whatever the adversary could have learnt from this interaction, it could have computed on its own after some T units of time have passed.* Passage of T units of time results in some new blocks being added to the blockchain. E.g., for ZK, our guarantee would mean that whatever the adversarial verifier could have learnt from the given interaction today, it could have computed on its own tomorrow (once the new blocks have been added to the blockchain). More precisely, to prove the ZK property, we require the existence of a simulator who would be "capable of doing time travel", get information about the future blocks, and come up with an indistinguishable transcript. To model this, our simulator will be a given as auxiliary input any valid continuation of the current state of the blockchain for time T. We refer to such a simulator as T-time-traveling simulator, and the resulting notion as T-time-traveling zero-knowledge. We remark that T-time-traveling zero-knowledge coincides with the standard notion of zero-knowledge when $T = 0$, and becomes weaker as T increases.

The above time-traveling zero-knowledge guarantee is almost as meaningful as the original zero-knowledge guarantee as far as the "long-term knowledge" is concerned. As an example, consider the GMW compiler [28] which uses ZK proofs of "honest behavior" to compile a semi-honest secure computation protocol into one that achieves malicious security. Here the witness consists of the private input and the randomness of a party. Hence, passage of T units of time will not give the adversary any further advantage in gaining any knowledge related to the witness. However if a statement is such that for some reason, its witness will inherently be leaked tomorrow (e.g, a time-locked puzzle [45]), our ZK proofs could cause this witness to be leaked to the adversary today (at least as far as the security proofs are concerned). Our ideas can also be seen as being inspired by the construction of Dwork and Naor [20] who used the opening of a timed commitment as the trapdoor witness in a Zap. However, in contrast to their work which focuses on realizing the standard ZK definition, we propose a new notion of security and justify it in the blockchain model. In Section 1.2, we provide a detailed comparison of time-traveling simulation with other weakenings of standard simulation including super-polynomial simulation and majority simulation.

1.1 Our Results

Zero-Knowledge with Time-traveling Simulators. First, we address the round-complexity of zero-knowledge with time-traveling simulation. We show that there exists a simple three-round Zero-knowledge Argument with a strict polynomial time, black-box time-traveling simulator. In the plain model, all of these properties are known to be impossible to achieve using the standard zero-knowledge definition w.r.t. black-box simulation [27]. Even w.r.t. non-black-box simulation, known three round zero-knowledge argument protocols require novel cryptographic assumptions [12].

Informal Theorem 1 (Zero-Knowledge (See Theorem 1)) *Assuming existence of injective one-way functions, there exists a three-round ZK argument for NP with a polynomial-time black-box time-traveling simulator.*

Looking ahead, we remark that zero-knowledge of our protocol is based on the assumption that injective one-way functions exist. For soundness, we need to rely on a natural assumption on the mining procedure of the underlying blockchain. Informally, we require that an adversary controlling minority of the computational power cannot be too far ahead of the honest miners. Specifically, such an adversary cannot compute k^2 blocks in time the honest miners compute k blocks. We emphasize that if this assumption is violated, then the adversary controls majority of the computing power over the network. This means that the adversary can effectively take over, e.g., the Bitcoin network. Our exact assumption is a variant of this natural assumption for PoW-based blockchains and is described in Assumption 1.

Secure Two-Party Computation with Time-traveling Simulators. We also study the task of secure two-party computation in the blockchain hybrid model. Intuitively, it allows two mutually distrustful parties to compute a joint function over their secret inputs such that neither party learns anything about other party's input beyond what can be learnt already by the function output. Secure computation is foundational task, and round-complexity is an important measure of efficiency. We focus on building two-party protocols between a *sender* and a *receiver* where only the receiver gets the output, but guarantee security against both a malicious sender and a malicious receiver.

Our main result is a three-round protocol for any two-party functionality where we achieve time-traveling simulation security for both parties.

Informal Theorem 2 (2PC (See Theorem 2)) *Assuming the existence non-interactive statistically binding commitments, IND-CPA secure public-key encryption, EUF-CMA secure public-key signatures, and two-round oblivious transfer, there exists a three-round two-party protocol for any efficiently computable functionality that exhibits a time-traveling strict polynomial-time black-box simulator against malicious adversaries.*

As before, the security is based on standard cryptographic primitives and a natural assumption on the underlying mining procedure of the blockchain as described above (see Section 3 for a formal description). We note that three-round 2PC in the plain model was constructed by [2] based on sub-exponential hardness of indistinguishability obfuscation but they achieve standard simulation security only for adversaries with bounded non-uniformity. Further, in the blockchain-hybrid model, [17] rule out even constant-round 2PC w.r.t. standard simulation.

The core technical challenge in achieving simulation-security in three rounds is extracting the input from an adversary. We develop a new two-round commitment scheme that satisfies a weak form of extraction. Specifically, we show that for every cheating committer, there exists a time-traveling extractor that can extract the value committed by the committer. We show that such weak time-traveling extraction guarantees are sufficient to achieve time-traveling simulation.

1.2 Relation of Time-traveling Simulation to Other Security Notions

Super-Polynomial Time Simulation. In super-polynomial simulation or angel-based security models [41,8], the simulator is given access to a specific form of super-polynomial computing power. For example in [16], the simulator is given access to an "angel" which can break the security of certain commitment schemes. While such power allows for obtaining strong results by bypassing various impossibility results, a key downside is that it can be challenging to argue that such a computing power does not break the security of honest parties (since their security is computational and can potentially be broken using any super-polynomial computation). Arguing the security of other protocols or even that of the ideal functionality (in case the functionality is cryptographic in nature) is even more tricky.

Our simulator can be seen as having access to a very specific super polynomial time angel which can instantaneously compute and provide future blocks on the blockchain. However the key advantage is the guarantee that *any security the angel breaks is limited to whatever would anyway automatically be broken shortly in the future.* This arguably makes our security guarantee stronger, and easier to understand and work with. For example, regardless of which commitment scheme the parties use, the commitment to their input can never be broken by our simulator.

Simulation with a Common Reference String. In the Common Reference String (CRS) model, a trusted party publishes a CRS before the start of the protocol. In the blockchain-hybrid model, the blockchain serves much the same purpose, by allowing parties to agree on the blockchain state. Basing security on blockchains instead of a CRS makes use of existing real-world infrastructure which is already trusted with securing hundreds of billions of dollars. This pre-existing infrastructure does not need to be modified for every secure protocol we wish to run using it, whereas the CRS format may change significantly from protocol to protocol.

A second point of similarity is the extra power provided to the simulator. When formalizing security using the real/ideal world paradigm, often this trusted party also provides the simulator with some trapdoor information related to the CRS, or the simulator itself is able to control the trusted party. This is similar to how a time-traveling simulator receives a future state to aid in simulation. However, a time-traveling simulator does not have power over the blockchain and is not able to "program" the common state. More compellingly, whatever security our simulator breaks is limited to what would have anyway been broken shortly into the future. During the natural course of the blockchain execution, every party will eventually see a state which could have been used to simulate in the past.

Majority Simulation. Goyal and Goyal [30] proposed the notion of majority simulation for protocols based on the blockchain. In it, the simulator controls all honest parties on the blockchain, which is assumed to be the majority of parties. This notion is philosophically very similar to honest majority simulation except that it borrows a pre-existing honest majority setup (the blockchain). In contrast, time-traveling simulators control only a single party. In fact, majority simulation seems strictly stronger than time travel simulation, because control over all honest parties in the blockchain may allow computation of future states.

The ledger functionality in the majority simulation model must be *local* (or "private") to the protocol. This is because otherwise, a majority simulator for protocol Π_1 who controls the bulletin board can break the security of another protocol Π_2 that uses the same bulletin board. In contrast, we identify an achievable set of properties which allow multiple different protocols to use a global ledger functionality at the same time under time-travel simulation (see Section 2.2). Furthermore, the extent of "broken" security in time-travel simulation is very limited: anything which is leaked the adversary would anyway have become automatically public shortly into the future.

Time-Based Primitives Dwork and Naor [20] proposed using the opening of a timed commitment as the trapdoor witness in a Zap. Since time-based primitives open only with significant computation effort (or with the help of the committer), the simulator needs to put in quite a bit of computational resources. Equivalently, the proofs could reveal information which can only be obtained by using high computation. This information may not automatically be available in the future. On the other hand, in our model, any information which the adversary learns is *automatically* available in the future (without the *adversary* putting in large computational resources).

2 Technical Overview

In Section 2.1, we introduce time-traveling simulators through the lens of zero-knowledge and sketch our three round zero-knowledge argument construction. Then, in Section 2.2, we discuss time-traveling simulators in the context of two-party computation and sketch our three-round 2PC construction. Along the way, we highlight several technical issues specific to simulating in the blockchain hybrid model and constructing time-traveling simulators specifically. For simplicity of exposition, we assume all parties see the most recent $\mathcal{G}_{\mathsf{ledger}}$ state here and delay ephemeral consensus issues until the technical sections.

2.1 Time-Traveling Simulators for Zero-Knowledge

A zero-knowledge argument aims to prove the validity of an NP statement without revealing anything about the witness other than what the verifier could have computed on its own. Informally, this is realized by requiring a simulator which can simulate the view of the adversary without having access to the witness. Slightly more formally, the adversary's view in the following two experiments should be computationally indistinguishable. In the real experiment, the adversary interacts with the honest prover and any honest parties in the blockchain. In the simulated experiment, the adversary interacts only with the simulator, which acts as an interface between the adversary and the rest of the world.

Time-traveling simulation for zero-knowledge aims to capture the notion that the verifier does not learn anything other than what it could have computed on its own after waiting for some time. We formalize this using the same experiments as the standard definition in the blockchain hybrid model, except that the simulator is additionally provided with a future state of the blockchain (e.g. 24 hours into the future). Specifically, if the state of the blockchain at the start of the protocol is st_S, the simulator receives a state st_F which is recognized by blockchain's chain validity predicate as a valid extension of st_S by F blocks. Since the blockchain will, if left alone, generate extensions of itself which are independent of x or its witnesses, st_F is effectively harmless and contains no information about the witness beyond what is naturally leaked with the passage of time.

Dependence on Blockchains Though it may be extended to other contexts, the notion of time-traveling simulation pairs particularly well with blockchains. Blockchains give a natural notion of time, since blocks are continually added to the chain. Furthermore, since blockchains provide a method of checking whether new blocks are a valid extension of the chain, what it means to be a future state can be quantified as an NP language. Finally, blockchains provide the guarantee that an adversary cannot be too far ahead of the rest of the blockchain (e.g. 24 hours worth of blocks). In contrast, a time-traveling simulator gets a future state for free. This increased power difference allows us to overcome impossibility results for standard simulation.

Three Round Zero-Knowledge A variety of impossibility results are known for three round zero-knowledge both in the plain model and the blockchain-hybrid model. Time-travel simulation bypasses these impossibility results and gives rise to a remarkably simple three round construction with a strict-polynomial-time blackbox time-traveling simulator. To prove $x \in \mathcal{L}$, the prover and verifier engage in a three round witness indistinguishable proof of knowledge (WIPoK)[3] for the statement "$x \in \mathcal{L}$ or I know a blockchain state F blocks ahead of the current state st_S".

Given the future state st_F, the time-traveling simulator can easily compute the proof using a witness for the latter half of the statement. Witness indistinguishability guarantees that the adversarial verifier cannot distinguish between this and a proof using a real witness for x.

Soundness Despite how much time-travel simulation simplifies the simulation itself, soundness becomes more subtle. First off, note that such proofs can only be sound if it finishes within an a-priori bounded time.[4] To ensure soundness, we need to ensure that the adversarial prover cannot

[3] Here we refer to a WIPoK construction for the plain model. The proof of knowledge property is more complicated in the blockchain-hybrid model [17].

[4] If the prover is allowed to wait until F blocks are added then it immediately gets access to a trapdoor, and can break soundness.

produce an accepting proof using the trapdoor branch. We base this on the assumption that no adversary can get F blocks ahead of the global blockchain, even given access to an oracle which computes a small number of blocks (e.g. $\sqrt{F}$) identically distributed to those mined by honest miners. Intuitively, in order to break this assumption, the adversary would need to compute $k+F$ blocks in the time it takes the honest miners to compute k blocks. This requires significantly more computational power than the honest miners have, contradicting the PoW assumption that the adversary has less computational power than the honest miners [38,23,34,24,43]. Even given the ability to generate, say, $\sqrt{F}$ blocks for free, the adversary would still need to mine the remaining $F-\sqrt{F}+k$ blocks on its own.[5] This is formalized in Assumption 1.

Using this assumption, a natural approach to proving soundness is to rely on the oracle in conjunction with the proof of knowledge property of the WIPoK to extract a witness for the adversary's proof. Then an adversarial prover which breaks soundness must be able to produce a state F blocks ahead of the global blockchain, violating its security.

The WIPoK knowledge extraction works by rewinding the adversarial prover and replaying messages to obtain two different accepting proofs with the same first message. A witness can be efficiently computed from these. However, as observed by Choudhuri et. al. [17], an adversary may attempt to maintain state across rewinding by posting messages on the blockchain. Any extractor must therefore also rewind the blockchain in order to rewind the adversary. The extractor can do this by using the oracle to compute an indistinguishable fork of the blockchain (i.e. a private chain of blocks which differs from the global blockchain) which it presents to the rewound adversarial prover.[6] Using only a single rewinding, the probability of getting two accepting transcripts from an adversarial prover which breaks soundness is noticeable. Since two accepting transcripts define a witness, an adversarial prover which breaks soundness can also violate the security of the blockchain.

2.2 Time-Traveling Simulators for Secure Two-Party Computation

In two-party secure computation, P_1 and P_2 compute a function f of their inputs x_1, x_2 where P_2 learns the $f(x_1, x_2)$ but nothing else, and P_1 learns nothing. This is formalized using the real/ideal world paradigm. In the real experiment, the adversary interacts directly with the honest party and the blockchain in an execution of the protocol. In the ideal world, the adversary interacts only with the simulator, which acts as an interface to the rest of the world. In turn, the simulator interacts with the blockchain and a trusted third party which reveals only the prescribed output. It is required that the joint distribution of the adversary's view and the honest party's output (as given by the trusted third party) are computationally indistinguishable across the two experiments. Time-traveling simulators additionally receive a future state st_F which is F blocks longer than the current blockchain state.

A unique property of the blockchain-hybrid model which will become relevant later on is that a distinguisher might attempt to use the number of blocks mined during an execution as side channel information to distinguish the two experiments. Since the simulator cannot rewind the global blockchain, it cannot hide how long the simulation takes. Thus a good simulator must be careful not to take more time than a real execution.

[5] In blockchains such as Bitcoin where the difficulty parameter may vary, the adversary might attempt to use the oracle to decrease the difficulty parameter. However, because the oracle's blocks are identically distributed to those mined by honest miners, the change in difficulty parameter will be identically distributed to an execution where the adversary attempts to directly modify the blockchain's difficulty parameter.

[6] Without the oracle, the adversary might attempt to use data in the blocks about who the miners are (e.g. signatures) is to determine if is being rewound.

Extraction of Inputs To guarantee security against malicious adversaries, which can behave arbitrarily, the simulator must extract the adversaries input so it can force the appropriate output. In the case of three round protocols, extracting an adversarial P_2's input is particularly challenging for standard simulators, since P_2 sends only a single message. Current solutions rely on non-black-box simulation and only hold for adversaries with bounded auxiliary input [2], or require a simulator which runs in super-polynomial time [6,7]. Intriguingly, extracting an adversarial P_1's input also appears to be quite challenging in the blockchain-hybrid model. As discussed previously, rewinding based extraction approaches require computing an indistinguishable fork for each rewinding. Even assuming the simulator could do so on its own, the number of rewindings required would blow up the running time of the simulator. This makes the outputs of simulators based on such approaches quite easy to distinguish.

Time-Traveling Extractors (Proposition 1) To solve the issue of timely knowledge extraction, we introduce time-traveling extractors. We design a two round commitment scheme with a time-traveling knowledge extractor.[7] Looking ahead, this will allow parties to commit to their input without revealing any information about it (hiding), while also ensuring they cannot change it later on (binding). A time-traveling extractor for a commitment scheme intuitively shows that any committer will eventually "know" the value that they committed to. Informally, we require the existence of an efficient extractor that, upon receiving a state st_F which is F blocks ahead of the current blockchain, produces both a commitment transcript indistinguishable from a real execution as well as the value committed to in that transcript. This concept naturally synergizes with time-travel simulation, which is our main use case. A key point is that hiding must continue to hold even after the passage of any (polynomial) amount of time.

To construct a two round commitment scheme with a time-traveling extractor, we start from a two round Conditional Disclosure of Secrets (CDS) protocol. A CDS protocol is associated with an NP relation $R_{\mathcal{L}}$ for the language $\mathcal{L}$ and a statement x. In a CDS protocol for a language $\mathcal{L}$, one party P_1 inputs a witness w, and the other party P_2 inputs a secret. If $(x, w) \in R_{\mathcal{L}}$, then P_1 gets the secret as output. However, if $x \notin \mathcal{L}$, then P_1 learns nothing about the secret. Furthermore, P_2 learns nothing about the witness input by P_1.

The commitment scheme proceeds as follows. The receiver sends a non-interactive commitment com_{ni} to the committer and begins the 2 round CDS for the statement "com_{ni} is a commitment to a future state of the blockchain". To commit to a message m, the committer inputs m to the CDS as the secret. Additionally, to ensure binding, the committer non-interactively commits to m and sends this to the receiver as well.

To extract the message m, time-traveling extractor can commit to the received st_F in com_{ni} and provide the witness to the CDS that com_{ni} is a commitment to a future state of the blockchain. Then the CDS will output m to the extractor.

Hiding is more difficult to ensure. To do so, we need to guarantee that an adversarial receiver cannot commit to a future state. If this is the case, then the CDS statement is false, so an adversarial receiver can learn nothing about the CDS secret m. Similarly to our zero-knowledge protocol's soundness, making this guarantee seems to require some form of extraction from com_{ni}. Goyal and Goyal [30] encounter a similar problem in their construction of non-interactive zero-knowledge using majority simulation. We are able to leverage their insights to extract from com_{ni} in a way which

[7] This construction may be of independent interest.

should not break the blockchain - therefore, if com_{ni} contains a future state, the extractor would have produced a future state in a short period of time, violating the security of the blockchain.

They construct such commitments by secret-sharing the message and encrypting the shares under the public keys of parties which have recently mined blocks. This provides hiding against adversaries with static corruptions. Then, given the secret keys of parties which have recently mined blocks, it is straightforward to decrypt and recombine the shares to recover the message. Since access to only a few extra secret keys should not provide the computational power required to get significantly ahead of the global blockchain (see Assumption 2), we are able to show that an adversary which breaks hiding also violates the security of the blockchain.

Note that this extractor requires power which is denied to time-travel simulators. Specifically, time-travel simulators must make do with control over only a single party that has minimal computational power compared to the other miners. As such they cannot gain access to even a small number of other parties' secret keys.[8] However, this method of extraction is sufficient to show hiding for our two round commitment.

Three Round Two-Party Secure Computation (Theorem 2) Given the tools we have constructed so far, a natural approach to three round two-party secure computation is to compile a two round semi-malicious two-party secure computation protocol (which can be implemented using a garbled circuit and oblivious transfer) into the malicious setting. In the first two rounds, P_2 commits to its input using the two round commitment with a time-traveling extractor. In the last two rounds, P_1 also commits to its input using the two round commitment with a time-traveling extractor. Additionally, the parties run the semi-malicious protocol in the last two rounds. To ensure that P_2 behaves honestly, in the last two rounds the parties participate in a CDS protocol where P_2 receives the third message if and only if it inputs a witness to its honesty in the second round. To ensure that P_1 behaves honestly, it proves its honesty using the three round zero-knowledge argument with a time-traveling simulator. The zero-knowledge argument is run in parallel with everything.

The ideas behind simulation are simple, though we will discuss some subtleties later. For time-travel simulation against a corrupted P_2, the simulator can use its future state to extract P_2's input in round two. Then in round three, it uses the extracted input to simulate the semi-malicious protocol and the future state to simulate the zero-knowledge argument. The case for time-travel simulation against P_1 is similar. Using the future state, the simulator extracts P_1's input in round 3, then provides it to the trusted third party. Otherwise, it behaves honestly using a fixed input (since P_1 does not receive output, there is no need to simulate the semi-malicious protocol).

Malleability Unfortunately, relying on time-traveling simulators for security both against malicious P_1 and against malicious P_2 introduces malleability issues. In round two, P_2 commits to its input in com_2, which has a time-traveling extractor. In round three, P_1 commits to its input in com_3, which *also* has a time-traveling extractor. When simulating against a malicious P_2^*, we need to rely on the hiding of com_3 while simultaneously extracting P_2^*'s input from com_2. This suggests that com_3's hiding needs to be "stronger" than com_2's hiding ($\mathsf{com}_3 >> \mathsf{com}_2$).

To achieve this, we give an analogue of complexity leveraging for time-traveling simulators. We assume that for times $F_1 << F_2$ (e.g. k and k^2), knowledge of a state st_{F_1} which is F_1 blocks ahead

[8] We assume that honest parties continue to maintain the semantic security of encryption under their secret keys indefinitely.

of the current blockchain does not allow the adversary to get F_2 blocks ahead of the blockchain (see Assumption 2). As in our previous assumption, the adversary would still need to compute $F_2 - F_1 + k$ blocks in the time the honest miners compute k blocks, requiring the adversary to have much more computational power than the honest miners.

Complexity leveraging for time-traveling simulators can be applied to ensure $\mathsf{com}_3 >> \mathsf{com}_2$ in much the same way that ordinary complexity leveraging can. We set the parameters of com_3 so that it has an F_2-time-traveling extractor but remains hiding against F_1-time-traveling receivers[9], and set the parameters of com_2 so that it has an F_1-time-traveling extractor. With this setting, a simulator against a malicious P_2^* can use st_{F_1} to break com_2 without disturbing com_3's hiding. Looking ahead, in our secure two-party computation construction we will set up a hierarchy $F_1 << F_2 << F_3$, where messages in round one use the parameter F_1, messages in round two use the parameter F_2, and messages in round three use the parameter F_3.

When simulating against a malicious P_1^*, we have the opposite problem from before - we need to rely on the hiding of com_2 while simultaneously extracting from com_3. Combining this with our previous requirement, we need both $\mathsf{com}_3 >> \mathsf{com}_2$ and $\mathsf{com}_3 << \mathsf{com}_2$. However, complexity leveraging for time-traveling simulators only allows hardness in one direction. To break the circularity, we observe that the extractor for com_2 requires knowledge of the future state *in round one*, but com_3 doesn't start until *round two*. In other words, com_2's period of vulnerability is over before com_3 even begins! We formalize the idea of a limited period of vulnerability using the following security game:

1. The adversary interacts with a challenger who commits in com_2 to either m_0 or m_1.
2. The distinguisher receives com_2 and a future state st_F which is F blocks ahead of the blockchain. It wins if it can determine which message the challenger committed to.

If no efficient adversary/distinguisher pair can win this game with noticeable advantage, we say com_2 has hiding against a time-traveling distinguisher. To show hiding of com_2 while simultaneously extracting from com_3 during simulation against a malicious P_1^*, we can reduce to this property. At a high level, the reduction participates in the above security game, then in step two uses st_F to extract from a locally computed com_3 before deciding which message com_2 contains. To show that com_2 has this property, we rely on the security of the underlying components in the plain model as well as the observation that the ledger (and therefore knowledge of a future state) can be efficiently emulated in the plain model.

Two-Party Computation Construction We summarize the final construction here. It makes use of the following tools, where $F_1 << F_2 << F_3$.

– A two round commitment com_2 with a F_1-time-traveling extractor and hiding against time-traveling distinguishers.
– A two round commitment com_3 with a F_2-time-traveling extractor and hiding against time-traveling distinguishers.
– A three round zero-knowledge argument with a F_3-time-traveling simulator.
– A two round semi-malicious secure two-party computation protocol.
– A two round conditional disclosure of secrets protocol.

[9] F_1 time-traveling receivers which break hiding of com_2 can be used to obtain an F_2 future state (breaking assumption 2) using the same ideas we developed for showing hiding of the commitments.

In the first two rounds, P_2 commits to its input in com_2. It also begins the semi-malicious two-party computation protocol and the com_3, as well as provides a witness to its honest behavior into the CDS. P_1 completes the semi-malicious protocol and inputs the resulting message into the CDS as the secret. Additionally it commits to its input in com_3 during the third round and proves its honesty using the zero-knowledge argument, which is run in parallel with everything.

2.3 Related Works

Zero-knowledge Round-complexity of zero-knowledge protocols is a fundamental measure of efficiency, and a heavily research topic in the plain model. In fact, in the plain model, we can construct four-round zero-knowledge protocols with black-box expected polynomial-time simulators assuming only one-way functions [9].[10] This result is tight w.r.t. black-box simulation [27], and known three-round zero-knowledge protocols with non-black-box simulation require novel cryptographic assumptions [12].

In the blockchain-hybrid model, the complexity of zero-knowledge was only recently considered by Choudhari et al. [17]. Quite surprisingly, they show that constant-round zero-knowledge protocols are impossible in the blockchain-hybrid model even for expected polynomial-time black-box simulators. Although they give a super-constant round protocol assuming one-way functions, their lower-bound hints that low-interaction protocols are quite hard to come by in the blockchain hybrid model unless simulator is given more power. In another work, Goyal and Goyal [30] present a significant strengthening of simulator's power that allows to achieve non-interactive zero-knowledge. In particular, in their notion of simulation the simulator is given control of all honest miners in the blockchain protocol. Although non-trivial, we believe such a notion of honest-majority simulation as described in [30] to be quite weak.

Secure Two-Party Computation The round complexity of secure two-party computation is another vibrant area of research. In the plain model, we can construct four-round secure two-party computation protocols (where a single party receives output) with black-box expected-polynomial-time simulators, though three-round protocols with respect to black-box simulation do not exist [33]. Known three-round protocols with non-black-box simulation hold only against adversaries with bounded non-uniformity [2]. Furthermore, in the blockchain-hybrid model, [17] rule out even constant-round protocols w.r.t. standard black-box simulation.

A variety of works have achieved protocols with less than four rounds by considering relaxed security notions such as the common reference string model [32,1] and super-polynomial time simulation [41,7,6].

Organization The rest of the paper is organized as follows: In Section 3 we define our modelling of the blockchain, and describe the hardness assumptions we need from PoW-based blockchains, and in Section 4 we detail standard definitions. In Section 5 we give our two-round commitment with a time-traveling extractor. Then, in Section 6 we present our three-round zero-knowledge protocol (and also an extension to satisfy proof-of-knowledge property). Finally, in Section 7 we present our two-round secure computation protocol.

[10] A simulator is *black-box* if it only makes uses of the underlying malicious verifier as a black-box.

3 Blockchain Model

In a blockchain protocol, mutually distrustful parties attempt to maintain and agree upon a global append-only ledger. The ledger consists of an ordered set of blocks which each contain some data. New blocks can only be added by using a special mining procedure which any party (called a miner) can run. Currently, two broad categories of mining procedures exist: Proof of Work (PoW) and Proof of Stake (PoS). In this work, we primarily consider PoW instantiations, though it may be possible to extend our ideas to PoS instantiations.

As in previous work [17,35,4], we model the blockchain as a global ledger $\mathcal{G}_{\mathsf{ledger}}$ which internally keeps track of the agreed-upon sequence of blocks. Parties may interact with the ledger via one of the queries specified by the ledger functionality. We follow the ledger functionality of [5] as described by [17]. Similarly to [17], data is trivially validated, and so we do not need the Validate predicate.

The functionality maintains several variables internally. state represents the sequence of blocks which have become buried under enough later blocks to be considered agreed-upon by all parties. buffer consists of messages which parties wish to post to the ledger but which are not yet permanently part of it. When an adversary proposes a next block candidate, the functionality records it in NxtBC for update on the next clock tick. Periodically the functionality validates (and potentially modifies) NxtBC using ExtendPolicy and appends the result to state after formatting it using Blockify. When it does so, it updates the list of block add times $\vec{\tau}_{\mathsf{state}}$ according to the current time τ_L. Common policies include NxtBC not being empty too often and it being "honestly generated" (e.g. satisfying such properties as including all messages from buffer) with some regularity. We refer readers to [5] for a further discussion of ExtendPolicy.

Any registered party may see the state at any time, but is only guaranteed a sufficiently long prefix of it. Each party P_i is associated with a monotonically increasing pointer ptr_i which represents that P_i may see up to block ptr_i in state. However, ptr_i may not be more than windowSize behind the true state for synchronized parties. A party is said to be desynchronized if it was recently registered or de-registered from the clock. Due to network delays, the adversary can make desynchronized parties believe any value of the state until they receive further messages, which must be delivered within Delay time. The ledger uses the predict-time function to ensure that the ideal world execution advances with the same pace as the real world execution.

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

The functionality is parameterized by three algorithms ExtendPolicy, Blockify, and predict-time, along with two parameters windowSize, Delay $\in \mathbb{N}$. The functionality manages variables state, NxtBC, buffer, τ_L, and $\vec{\tau}_{\mathsf{state}}$, as described above. Initially, state $:= \vec{\tau}_{\mathsf{state}} :=$ NxtBC $:= \varepsilon$, $\tau_L = 0$.

The functionality maintains a set of registered parties $\mathcal{P}$, the (sub–)set of honest parties $\mathcal{H} \subseteq \mathcal{P}$, and the (subset) of desynchronized honest parties $\mathcal{P}_{DS} \subset \mathcal{H}$ (following the definition in the previous paragraph). 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 than is added to the party sets $\mathcal{H}$ and $\mathcal{P}$ and the current time of registration is also recorded; if the current time is $\tau_L > 0$, 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 letter maintains the invariant that it is registered (as a functionality) to the clock whenever $\mathcal{H} \neq \emptyset$. A party is considered fully registered if it is registered with the letter and the clock.

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

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

1. Let $\hat{\mathcal{P}} \subseteq \mathcal{P}_{DS}$ denote the set of desynchronized honest parties that have been registered continuously since time $\tau' < \tau_L - \mathsf{Delay}$ to both the ledger and clock. Set $\mathcal{P}_{DS} := \mathcal{P}_{DS} \backslash \hat{\mathcal{P}}$. On the other hand, for any synchronize party $P \in \mathcal{H} \backslash \mathcal{P}_{DS}$, if P is not registered with the clock, then set $\mathcal{P}_{DS} := \mathcal{P}_{DS} \cup \{P\}$.
2. If I was received from an honest party $P_i \in \mathcal{P}$:
   (a) Set $\vec{\mathcal{I}}_H^T = \vec{\mathcal{I}}_H^T || (I, P_i, \tau_L;$
   (b) Compute $\vec{N}) \vec{N}_1, \ldots, \vec{N}_\ell := \mathsf{ExtendPolicy}(\vec{\mathcal{I}}_H^T, \mathsf{state}, \mathsf{NxtBC}, \mathsf{buffer}, \vec{\tau}_{\mathsf{state}})$ and if $\vec{N} \neq \varepsilon$ set $\mathsf{state} := \mathsf{state} || \mathsf{Blockify}(\vec{N}_1) || \ldots || \mathsf{Blockify}(\vec{N}_\ell)$ and $\vec{\tau}_{\mathsf{state}} := \vec{\tau}_{\mathsf{state}} || \tau_L^\ell$ where $\tau_L^\ell = \tau_L || \ldots || \tau_L$.
   (c) If there exists $P_j \in \mathcal{H} \backslash \mathcal{P}_{DS}$ such that $|\mathsf{state}| - ptr_j > \mathsf{windowSize}$ or $ptr_j < |\mathsf{state}_j|$, then set $ptr_k := |\mathsf{state}|$ for all $P_k \in \mathcal{P} \backslash \mathcal{P}_{DS}$.
   (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 by an honest $P_i \in \mathcal{P}$ then set $\mathsf{state}_i := \mathsf{state} \lceil_{\min\{ptr_i, |\mathsf{state}|\}}$ and return $(\mathsf{READ}, \mathsf{sid}, \mathsf{state}_i)$ to the requester. If the requester is $\mathcal{A}$ then send $\mathsf{state}, \mathsf{buffer}$.
   – *Maintain ledger state*:
   If $(\text{MAINTAIN-LEDGER}, \mathsf{sid})$ is received by an honest $P_{i \in \mathcal{P}}$ and $\mathsf{predict\text{-}time}(\vec{\mathcal{I}}_H^T) = \tilde{\tau} > \tau_L$ then send $(\text{CLOCK-UPDATE}, \mathsf{sid}_\mathrm{C})$ to $\mathcal{G}_{\mathsf{clock}}$. Else send I to $\mathcal{A}$.
   – *The adversary proposing the next block*:
   If $I = (\text{NEXT-BLOCK}, \mathsf{hflag}, (\mathsf{uid}_1, \ldots, \mathsf{uid}_\ell))$ 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 $I = (\text{SET–SLACK}, (P_{i_1}, \widehat{ptr}_{i_1}), \ldots, (P_{i_\ell}, \widehat{ptr}_{i_\ell}))$ with $\{P_{i_1}, \ldots, P_{i_\ell}\} \subseteq \mathcal{H} \backslash \mathcal{P}_{DS}$ is received from the adversary, do the following:
   (a) If $\forall j \in [\ell]$: $|\mathsf{state}| - \hat{ptr}_{i_j} \leq \mathsf{windowSize}$ and $\hat{ptr}_{i_j} \geq |\mathsf{state}_{i_j}|$, set $ptr_{i_j} = \hat{ptr}_{i_j}$ for every $j \in [\ell]$ and return $(\text{SET-SLACK}, ok)$ to $\mathcal{A}$.

(b) Otherwise set $ptr_{i_j} = |\mathsf{state}|$ for all $j \in [\ell]$.

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

Remarks We remark on a few properties of $\mathcal{G}_{\mathsf{ledger}}$ and the blockchain which implements it.

– For simplicity of exposition, we ignore the notion of desynchronized parties. This can be made explicit by requiring a waiting period of Delay at the beginning of each protocol so that the participants can synchronize. We also consider $\mathcal{G}_{\mathsf{clock}}$ to be local to $\mathcal{G}_{\mathsf{ledger}}$ and ignore it.
– The blockchain implementing $\mathcal{G}_{\mathsf{ledger}}$ is associated with a predicate $\mathsf{isvalidchain}(\mathsf{st}_1, \mathsf{st}_2)$ which decides whether st_2 is a valid extension of st_1. For example, in Bitcoin this consists of verifying that st_1 is a prefix of st_2 and that all later blocks have an accepting proof of work with respect to the previous blocks. We overload this notation as $\mathsf{isvalidchain}(\mathsf{st}_1, \mathsf{st}_2, k)$ to also check that st_2 is at least k blocks longer than st_1 (i.e. $|\mathsf{st}_2| - |\mathsf{st}_1| \geq k$).
– Define $D_{\mathsf{fut}}(T, F)$ to be the distribution over the future states of the blockchain with length T+F, where the current state is $\mathcal{G}_{\mathsf{ledger}}$'s state at time T. Concretely, the distribution is over the random coins of the miners and parties submitting messages to the blockchain. Every state st_F drawn from $D_{\mathsf{fut}}(T, F)$ satisfies $\mathsf{isvalidchain}(\mathcal{G}_{\mathsf{ledger}}.\mathsf{st}, \mathsf{st}_F, F) = 1$.

Simulation in the $\mathcal{G}_{\mathsf{ledger}}$-Hybrid Model We consider simulators with the same power as other parties while accessing $\mathcal{G}_{\mathsf{ledger}}$, with the exception of being given a future state. Unlike the setting considered in [30,18], the simulator does not have full control over the blockchain and cannot "rewind" it by discarding and re-creating blocks. It also cannot quickly compute future states beyond what it was given (see Assumption 2).

The simulator acts as an interface between any parties (e.g. the adversary) it runs internal to itself and $\mathcal{G}_{\mathsf{ledger}}$. It can therefore choose which messages to deliver. Note that the adversary may attempt to post messages to $\mathcal{G}_{\mathsf{ledger}}$ as well as base its behavior on its view of $\mathcal{G}_{\mathsf{ledger}}$.

Since the protocols we construct begin at a specified time (in terms of the number of blocks in $\mathcal{G}_{\mathsf{ledger}}$'s state), the simulator begins at the same time. Without loss of generality, it knows the pointers of all parties at this time. Observe that for every adversary which exists before the protocol, there is another adversary which produces an identical ledger state and tracks the pointers of all parties, by forwarding and tracking messages between the original and $\mathcal{G}_{\mathsf{ledger}}$. Since the simulator inherits the state of the adversary upon starting, it also inherits the pointers of all parties at this time.

First, let us introduce some notation. We use $\mathcal{G}_{\mathsf{ledger}}.\mathsf{st}$ denotes the current state of the $\mathcal{G}_{\mathsf{ledger}}$, and it is a sequence $(B_1, B_2, \dots,)$ of blocks where each B_i contains some m, a proof-of-work π and we also assume the block contains the public-key of the miner who mined the block.[11] We denote by $|\mathcal{G}_{\mathsf{ledger}}.\mathsf{st}|$ the number of blocks in the $\mathcal{G}_{\mathsf{ledger}}$, and use the shorthand $\mathcal{G}_{\mathsf{ledger}}.\mathsf{size}$ to refer to $|\mathcal{G}_{\mathsf{ledger}}.\mathsf{st}|$. For any natural number $i \leq \mathcal{G}_{\mathsf{ledger}}.\mathsf{size}$, we denote by $\mathcal{G}_{\mathsf{ledger}}.\mathsf{st}[: i]$ as the state of the $\mathcal{G}_{\mathsf{ledger}}$ including only the first i blocks. Finally, we have a predicate $\mathsf{isledgerstate}$ that takes a candidate state st as input, and outputs 1 iff there exists $i \leq \mathcal{G}_{\mathsf{ledger}}.\mathsf{size}$ such that $\mathcal{G}_{\mathsf{ledger}}.\mathsf{st}[: i] = \mathsf{st}$.

[11] We believe our ideas can also be applied to proof-of-stake blockchains, though we only provide formalization for proof-of-work blockchains.

By $\mathsf{EXEC}^{\mathcal{G}_{\mathsf{ledger}}}(Z, A, 1^\lambda)$ be the random variable denoting an execution of the $\mathcal{G}_{\mathsf{ledger}}$ with the environment Z and some adversary A on security parameter λ. We let $(\mathsf{st}, z, y) \leftarrow\$ \mathsf{EXEC}^{\mathcal{G}_{\mathsf{ledger}}}(Z, A, 1^\lambda)$ denote the random process of running an execution of $\mathcal{G}_{\mathsf{ledger}}$ with (Z, A) until A outputs y where st is the current state of $\mathcal{G}_{\mathsf{ledger}}$ and z is output of Z.

Hardness Assumptions. In this section, we describe the specific hardness we require from the underlying mining procedure. For this, first consider the following relation $R^{\mathcal{G}_{\mathsf{ledger}}} = \{R_\lambda\}_{\lambda \in \mathbb{N}}$ containing pairs defined by $\mathcal{G}_{\mathsf{ledger}}$ functionality.

$$R_\lambda^{\mathcal{G}_{\mathsf{ledger}}} = \left\{ ((\mathsf{st}_s, T), \mathsf{st}_f) : \begin{array}{c} \mathsf{isledgerstate}(st_s) = 1, T \in \mathbb{N}, \\ \mathsf{isvalidchain}(\mathsf{st}_s, \mathsf{st}_f, T) \end{array} \right\} . \tag{1}$$

Intuitively, we want to capture that any adversary, performing bounded number of operations, cannot compute F blocks by the time $k << F$ blocks are added to $\mathcal{G}_{\mathsf{ledger}}$ by the honest miners. We refer to this as (k, F)-security of $\mathcal{G}_{\mathsf{ledger}}$, and in Definition 1 we formalize the notion of what it means for an adversary performing t operations to break (k, F)-security of $\mathcal{G}_{\mathsf{ledger}}$. To allow the adversary to choose the starting state, we consider a two-stage adversary $A = (A_0, A_1)$ where A_0 is arbitrary PPT and participates in an execution of $\mathcal{G}_{\mathsf{ledger}}$ (with the honest miners) to arrive at a starting state st_s, and A_1 is supposed to find an extension of st_s by F blocks, by the time k blocks are added to $\mathcal{G}_{\mathsf{ledger}}$. We overload notation of EXEC by giving it an fourth parameter k which it uses to terminate the execution if k additional blocks are added.

Definition 1. *We say that $A = (A_0, A_1)$ t-breaks the (k, F)-security of $\mathcal{G}_{\mathsf{ledger}}$ if there exists some non-negligible function ϵ such that for all $\lambda \in \mathbb{N}$*

$$\Pr\left[((\mathsf{st}_s, F), \mathsf{st}_f) \in R : \begin{array}{c} (\mathsf{st}_s, z, a) \leftarrow\$ \mathsf{EXEC}^{\mathcal{G}_{\mathsf{ledger}}}(Z, A_0, 1^\lambda) \\ (\mathsf{st}_i, z', \mathsf{st}_f) \leftarrow\$ \mathsf{EXEC}^{\mathcal{G}_{\mathsf{ledger}}, \mathcal{O}_{\mathsf{blockify}}}(Z(z), A_1(\mathsf{st}_s), 1^\lambda, k(\lambda)) \end{array} \right] \geq \epsilon(\lambda) \tag{2}$$

where A_0 is arbitrary PPT, and A_1 performs at most $t(\lambda)$ operations and makes at most k queries to $\mathcal{O}_{\mathsf{blockify}}$, where $\mathcal{O}_{\mathsf{blockify}}$ takes as input a block B and a message m, and outputs a new block B' that contains message m such that $\mathsf{isvalidchain}(B, B') = 1$*, and R is as defined in Equation (1).*

Assumption 1 *For all polynomially bounded functions k, t, there exists some polynomially bounded function F such that no PPT adversary $A = (A_0, A_1)$ t-breaks the (k, F)-security of $\mathcal{G}_{\mathsf{ledger}}$ (as per Definition 1.*

Remark 1. Intuitively, t represents the number of mining operations that an adversary can perform in the time it takes the honest miners to mine k blocks. For the adversary to be able to produce $F > k$ blocks (e.g. $F = k^2$ or even ck, for large enough constant c) in this time, t should be significantly larger than the honest mining power. This would violate the PoW assumption that the adversary controls less mining power than the honest miners. Even given access to an oracle which mines up to k blocks for free (either in the past or currently), the computational requirement on the adversary does not change too much. The adversary would still need to mine the remaining $F - k$ blocks on its own. Since the oracle creates blocks according to the same distribution that honest parties would have, the adversary also cannot use it to decrease the difficulty parameter from what it would have been in a real execution, meaning the work it needs to do for each block does not change.

In fact, for some of security proofs, we will require a stronger notion of security from $\mathcal{G}_{\mathsf{ledger}}$. Specifically, we want it to be computationally hard for A_1 to compute a state F blocks ahead even if its given as auxiliary information (a) some $F' << F$ future blocks, and (b) secret-keys of the most recent miners. As before, we first formalize what it means for such an A_1 to break the security of $\mathcal{G}_{\mathsf{ledger}}$, and next state the exact assumption which we conjecture to hold for PoW-based blockchains, e.g., Bitcoin.

Let $t, \ell, F', k, F : \mathbb{N} \to \mathbb{N}$ be polynomially bounded functions.

Definition 2. *We say that $A = (A_0, A_1)$ (t, ℓ, F')-breaks the (k, F)-security of $\mathcal{G}_{\mathsf{ledger}}$ if there exists some non-negligible function ϵ such that for all $\lambda \in \mathbb{N}$*

$$\Pr\left[((\mathsf{st}_s, F), \mathsf{st}_f) \in R : \begin{array}{c} (\mathsf{st}_s, z, a) \leftarrow\!\$\ \mathsf{EXEC}^{\mathcal{G}_{\mathsf{ledger}}}(Z, A_0, 1^\lambda) \\ \mathsf{st}_i \leftarrow\!\$\ \mathcal{S}(\mathsf{st}_s, F'(\lambda)) \\ (\mathsf{st}_j, a', \mathsf{st}_f) \leftarrow\!\$\ \mathsf{EXEC}^{\mathcal{G}_{\mathsf{ledger}}, \mathcal{O}_{\mathsf{blockify}}}(Z(a), A_1(\mathsf{st}_s, \mathsf{st}_i, \vec{sk}_\ell), 1^\lambda, k(\lambda)) \end{array}\right] \geq \epsilon(\lambda) \tag{3}$$

where A_0 is arbitrary PPT, A_1 performs at most $t(\lambda)$ operations and makes at most k queries to $\mathcal{O}_{\mathsf{blockify}}$, where $\mathcal{O}_{\mathsf{blockify}}$ takes as input a block B and a message m then outputs a new block B' which contains message m such that $\mathsf{isvalidchain}(B, B') = 1$, $\mathcal{S}$ on input a starting state st_s outputs an intermediate future state state st_i such that $\mathsf{isvalidchain}(\mathsf{st}_s, \mathsf{st}_i, F') = 1$ and $\vec{sk}_\ell$ are the secret-keys of the miners mining the last ℓ blocks in st_s.

Assumption 2 *For all polynomially bounded functions k, t, ℓ, F', there exists polynomially bounded functions F such that no PPT adversary $A = (A_0, A_1)$ (t, ℓ, F')-breaks the (k, F)-security of $\mathcal{G}_{\mathsf{ledger}}$ (as per Definition 2).*

Although Assumption 2 is a strengthening of Assumption 1 we believe, especially in the context of bitcoin, that having access to the secret-keys of the miners and/or some of the future states doesn't make computing states much further in the future any easier. For PoW blockchains such as Bitcoin, having the private keys of recent miners does not make the computational puzzle (based on hashing) any easier to solve. Knowledge of the future state can be seen as a special case of the oracle provided in Assumption 1.

3.1 Chain Quality

Chain quality is a thoroughly analyzed property for many blockchains which is closely related to the fraction of mining power which the adversary controls [23,34,24,43]. Intuitively, it requires that the blockchain consists of a minimal fraction of honestly mined blocks (commonly proportional to the honest fraction of total mining power).

We define the predicate $\mathsf{quality}(\mathsf{st}, \beta, \ell)$ such that $\mathsf{quality}(\mathsf{st}, \beta, \ell) = 1$ if and only if the last ℓ blocks of st were mined by honest miners.

Definition 3 (Chain Quality). *$\mathcal{G}_{\mathsf{ledger}}$ satisfies (β, ℓ)-chain quality with environment Z and adversary A if for all $i \in \mathbb{N}$ and all $\ell' > \ell$, the following holds:*

$$\Pr[\mathsf{quality}(\mathsf{st}, \beta, \ell) = 1 | (\mathsf{st}_s, z, a) \leftarrow\!\$\ \mathsf{EXEC}^{\mathcal{G}_{\mathsf{ledger}}}(Z, A, 1^\lambda)] \geq 1 - \mathsf{negl}(\lambda)$$

4 Definitions and Preliminaries

The security parameter is λ. We denote cryptographic indistinguishability by $\approx_c$. Two distributions X_0, X_1 satisfy this if for all PPT D,

$$\left|\Pr[D(x) = 1 : x \leftarrow\$ X_0] - \Pr[D(x) = 1 : x \leftarrow\$ X_1]\right| \leq \mathsf{negl}(\lambda)$$

Though our definitions specify the circuit class for sake of generality, in this work we only consider adversaries which are unable to break security of the blockchain. Distinguishers, however, are allowed to be arbitrary PPT machines.

We provide definitions directly relating to time traveling simulators and extractors below. See appendix A for standard definitions.

Blockchain Awareness Unless otherwise specified, we consider adversaries which have access to the global functionality $\mathcal{G}_{\mathsf{ledger}}$. For indistinguishability of two distributions, this means the distinguisher also has access to $\mathcal{G}_{\mathsf{ledger}}$ as soon as the samples are generated. One consequence of this is that the distinguisher can check the time an interaction ends as well as wait for future states to become available before attempting to distinguish. Additionally, this means that the distinguisher can immediately distinguish any view generated by a simulator using a privately initialized $\mathcal{G}_{\mathsf{ledger}}$ from the real execution by checking the state of the global $\mathcal{G}_{\mathsf{ledger}}$.

Many of our protocol definitions require some quantification of the time an execution is started, with respect to $\mathcal{G}_{\mathsf{ledger}}$. We denote that an execution of an algorithm or protocol Π is started when the internally held state of $\mathcal{G}_{\mathsf{ledger}}$ has T blocks ("time T") by Π_T. The algorithm may not be aware of the actual state or time of $\mathcal{G}_{\mathsf{ledger}}$.

Similar to [30], we only consider statically corrupting adversaries, though the blockchain may be secure against adaptive corruptions. We believe our assumptions and results also hold against adaptive corruptions with erasures. Achieving our results against adaptively corrupting adversaries without erasures is an interesting open question.

4.1 Time-Travel Simulation for Zero Knowledge

For interactive algorithms P_1, P_2, we denote by $\mathsf{out}_{P_1}(P_1(x), P_2(y))$ the output of P_1 in an interaction with P_2 where P_1 has input x and P_2 has input y. The corresponding view is denoted by $\mathsf{view}_{P_1}(P_1(x), P_2(y))$.

Definition 4 (Argument Systems). *Let $\mathcal{C}$ be a circuit class. An interactive protocol (P, V) is a $\mathcal{C}$-argument system for a language L with NP relation R_L if it satisfies the following properties:*

- ***Completeness:*** *For every $(x, w) \in R_L$,*

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

- ***$\mathcal{C}$-Soundness:*** *For every $x \notin L$ and every adversary $P^* \in \mathcal{C}$,*

$$Pr\left[\mathsf{out}_V(P^*(1^\lambda, x), V(1^\lambda, x)) = 1\right] = \mathsf{negl}(\lambda)$$

This definition is in the plain model. If these properties hold in the $\mathcal{G}_{\mathsf{ledger}}$-hybrid model and (P, V) may require interaction with $\mathcal{G}_{\mathsf{ledger}}$, then we call (P, V) a blockchain-aware argument system. If soundness holds against unbounded provers, (P, V) is called a proof system. If the prover may choose the statement to be proven in the last round, it is called a delayed-input argument/proof system.

A time-traveling simulator is given a randomly sampled future state of the blockchain and must produce an indistinguishable transcript from the real-world protocol.

Definition 5 (Zero Knowledge with Time-Travel Simulation). *Let $\mathcal{C}$ be a circuit class. An argument system (P, V) for a language L is **$\mathcal{C}$-Zero Knowledge with an F-Time-Traveling Simulator** if there is a PPT algorithm Sim such that for all adversaries $V^* \in \mathcal{C}$, every $(x, w) \in R_L$, and all times T, the following holds in the $\mathcal{G}_{\mathsf{ledger}}$-hybrid model:*

$$\{\mathsf{view}_{V^*}(P_T(1^\lambda, x, w), V_T^*(1^\lambda, x))\} \approx_c \{\mathsf{Sim}_T(1^\lambda, x, \mathsf{st}_F) : \mathsf{st}_F \leftarrow\$\, D_{\mathsf{fut}}(T, F)\}$$

Proofs of time-traveling knowledge intuitively show that the prover will soon "know" the witness to the statement they proved. We formalize this by a time-traveling extractor which receives a future state and must produce both a transcript indistinguishable from the real world and a valid witness.

We denote the decision bit for whether a proof π for a statement x is accepted by $\mathsf{Acc}(x, \pi)$.

Definition 6 (Arguments of Time-Traveling Knowledge). *Let $\mathcal{C}$ be a circuit class. A delayed-input argument system (P, V) for a language L with relation R_L is **a $\mathcal{C}$-argument of time-traveling knowledge** if there exists a PPT extractor Ext such that for all adversaries $P^* \in \mathcal{C}$ and all times T the following two properties hold:*

1. ***$\mathcal{C}$-View Indistinguishability.***

$$\{\mathsf{view}_{P^*}(P_T^*(1^\lambda, x, w), V_T(1^\lambda, x))\} \approx_c \{\tilde{\pi} : (\tilde{\pi}, w) \leftarrow \mathsf{Ext}_T(1^\lambda, x, P^*, \mathsf{st}_F), \mathsf{st}_F \leftarrow\$\, D_{\mathsf{fut}}(T, F)\}$$

2. ***$\mathcal{C}$-Extraction.***

$$\Pr[\mathsf{out}_V(P^*(1^\lambda, x), V(1^\lambda, x)) = 1] \leq \Pr[\mathsf{Ext}_T(1^\lambda, x, P^*, \mathsf{st}_F) \in R_L(x) : \mathsf{st}_F \leftarrow\$\, D_{\mathsf{fut}}(T, F)] + \mathsf{negl}(\lambda)$$

We emphasize that the witness-extraction process must not take too long, or else view indistinguishability will fail.

For our secure two-party computation construction, we will also require this to be a delayed-input argument, where the adversary may choose the statement x in the last round. Without loss of generality, x is included in the transcript in the last round.

4.2 Time-Travel Simulation for Secure Computation

A secure two-party computation protocol for a functionality f is carried out by two parties P_1, P_2 with inputs x_1, x_2, respectively. At the end of the protocol, P_2 gets the output $f(x_1, x_2)$. We formalize security using the real/ideal world paradigm in the $\mathcal{G}_{\mathsf{ledger}}$-hybrid model.

Ideal World The ideal world contains P_1, P_2, and a trusted third party. At most one of P_1 and P_2 are controlled by the adversary. Between any messages, parties may do a polynomial amount of interaction with $\mathcal{G}_{\mathsf{ledger}}$. The ideal world execution proceeds as follows:

1. **Input Distribution:** P_1 and P_2 receive their respective inputs x_1 and x_2 from the environment.
2. **Inputs to Trusted Third Party:** P_1 and P_2 send their inputs to the trusted third party. An honest party always immediately sends the input they received from the environment. The corrupted party may send any input of their choice.
3. **Optional Abort:** The adversary may send a message to the trusted third party instructing it to abort. Immediately upon receiving this, the trusted third party terminates the execution. Otherwise the execution continues.
4. **Trusted Third Party Answers P_2:** If it is not been instructed to abort, the trusted third party sends $f(x_1', x_2')$ to P_2, where x_1' and x_2' are the inputs it received in step 2.
5. **Output:** If P_2 is honest, it outputs $f(x_1', x_2')$ as received from the trusted third party. If P_1 is honest, it outputs $\perp$. The adversarial party always outputs its entire view.

We define $\mathsf{Ideal}_{T,f,A}(x_1, x_2)$ to be the joint distribution over the outputs of the adversary A and the honest party according to the above ideal execution started at time T.

Real World Let Π be a two-party protocol computing f. In the real process, both parties execute the protocol Π. This may involve some amount of communication with $\mathcal{G}_{\mathsf{ledger}}$. At most one of P_1 and P_2 are controlled by the adversary. Between any messages, parties may do a polynomial amount of interaction with $\mathcal{G}_{\mathsf{ledger}}$. As in the ideal process, they receive inputs from the environment. The honest party outputs according to the Π specification, while the adversary outputs its entire view.

We define $\mathsf{Real}_{T,\Pi,A}(x_1, x_2)$ to be the joint distribution over the outputs of the adversary A and the honest party according to the above real execution started at time T.

Definition 7 (Secure Computation with Time-Travel Simulation). *Let $\mathcal{C}$ be a circuit class. Let Π be a two party protocol computing the two party functionality f. We say Π* ***$\mathcal{C}$-securely computes*** *f with* ***F-time-traveling simulation*** *if there is a stateful PPT algorithm* Sim *such that for all $A \in \mathcal{C}$, all inputs x_1, x_2, and all times T, the following holds:*

$$\{\mathsf{Real}_{T,\Pi,A}(x_1, x_2)\} \approx_c \{\mathsf{Ideal}_{T,f,\mathsf{Sim}(\mathsf{st}_F)}(x_1, x_2) : \mathsf{st}_F \leftarrow\!\$\, D_{\mathsf{fut}}(T, F)\}$$

As mentioned previously, the distinguisher has access to $\mathcal{G}_{\mathsf{ledger}}$, and is able to time the interactions. *This includes the time of the honest party's output*, which is included in the joint distribution. Since an honest P_2's output in the ideal world depends on the ideal functionality, the simulator against P_1 must therefore extract the corrupted P_1's output and provide it to the ideal functionality without wasting too much time.

4.3 Time-Travel Extraction for Commitments

Definition 8 (Statistically Binding Commitment Scheme). *A two round commitment scheme is a triple* (Com, Rec, Open) *of three algorithms with the following syntax:*

1. Rec *is a PPT algorithm that on input 1^λ outputs the first message c_1.*
2. Com *is a PPT algorithm that on inputs 1^λ, the receiver message c_1, and a value m outputs a message c_2 for the receiver and retains decommitment information d.*

3. Open *is a deterministic function that on input a commitment* $\tau = (c_1, c_2)$ *(transcript of the interaction between* Com *and* Rec*), value* m *and decommitment information* d *outputs a decision bit.*

We need the following properties:

– ***Binding****. For every cheating (potentially unbounded) adversary* C^**, we have*

$$\Pr[\exists m_0, d_0, m_1, d_1 \text{ such that } m_0 \neq m_1 \wedge (\mathsf{Open}(\tau, m_0, d_0) = \mathsf{Open}(\tau, m_1, d_1) = 1)] \leq \mathsf{negl}(\lambda)$$

where the probability is taken over the random coins used in sampling the transcript τ *between* C^* *and the honest receiver* Rec.

– $\mathcal{C}$***-Hiding****. For every adversary* $R^* \in \mathcal{C}$*, and every pair of strings* m_0, m_1

$$\left\{\mathsf{view}_{R^*}\left(\mathsf{Com}_T(1^\lambda, m_0), R^*_T\right)\right\} \approx_c \left\{\mathsf{view}_{R^*}\left(\mathsf{Com}_T(1^\lambda, m_1), R^*_T\right)\right\}$$

The above definition is in the plain model. If these properties hold in the $\mathcal{G}_{\mathsf{ledger}}$-hybrid model, then $(\mathsf{Com}, \mathsf{Rec}, \mathsf{Open})$ is a commitment scheme in the $\mathcal{G}_{\mathsf{ledger}}$-hybrid model.

A non-interactive commitment scheme consists only of $(\mathsf{Com}, \mathsf{Open})$, but otherwise follows the above definition.

Definition 9 (Value of Commitment). *Given a commitment transcript* $c = (c_1, c_2)$ *generated by an interaction between cheating committer* C^* *and an arbitrary receiver* R^* *(which may not be the honest receiver), we define the value of the commitment* $\mathsf{val}(c)$ *as* m *if there exists a unique value* m *such that* $c_2 = \mathsf{Com}(c_1, m; r)$ *for some random tape* r*, otherwise* $\mathsf{val}(c) = \bot$.

Definition 10 (Time-Travel Extraction). *Let* $\mathcal{C}$ *be a circuit class. We say that a two round commitment scheme* $(\mathsf{Com}, \mathsf{Rec}, \mathsf{Open})$ *has an* F***-time-traveling*** $\mathcal{C}$***-extractor*** *if there exists a PPT extractor* Ext *such that for all adversaries* $C^* \in \mathcal{C}$ *and all times* T*:*

1. $\mathcal{C}$***-View Indistinguishability.***

$$\{\mathsf{view}_{C^*}(C^*_T(1^\lambda), \mathsf{Rec}_T(1^\lambda))\} \approx_c \{\tilde{\tau} : (\tilde{\tau}, \tilde{m}) \leftarrow\!\!\$\ \mathsf{Ext}_T(1^\lambda, C^*, \mathsf{st}_F),\ \mathsf{st}_F \leftarrow\!\!\$\ D_{\mathsf{fut}}(T, F)\}$$

2. ***Over-extraction.***

$$\Pr[\tilde{m} \neq \mathsf{val}(\tilde{\tau}) \wedge \mathsf{val}(\tilde{\tau}) \neq \bot : (\tilde{\tau}, \tilde{m}) \leftarrow\!\!\$\ \mathsf{Ext}_T(1^\lambda, C^*, \mathsf{st}_F),\ \mathsf{st}_F \leftarrow\!\!\$\ D_{\mathsf{fut}}(T, F)] \leq \mathsf{negl}(\lambda)$$

Remark 2. We emphasize that our extractor is allowed to sample a transcript/view from a distribution that is only computationally close to $\mathsf{view}_{C^*}(C^*, \mathsf{Rec})$. In particular, the distribution can be statistically far away from $\mathsf{view}_{C^*}(C^*, \mathsf{Rec})$. This is different from the extractability notions considered in prior works [31,13,14] where extractor is required to sample from a distribution that is statistically close to $\mathsf{view}_{C^*}(C^*, \mathsf{Rec})$. Some works also consider the stronger notion where extractor is given a transcript from $\mathsf{view}_{C^*}(C^*, \mathsf{Rec})$ and asked to recover the underlying value.

4.4 Indistinguishability Against Time-Traveling Distinguishers

A time-traveling distinguisher receives both a sample from the distribution and a randomly sampled future state. Given these, it may interact with the ledger in order to learn more information, or immediately attempt to distinguish the samples. Note that the distinguisher is an arbitrary PPT machine, and so may have significantly more mining power than the honest miners. However, it cannot necessarily create future states from the same distribution as a real execution of the blockchain (where the adversary controls strictly less than half of the mining power), e.g. because it cannot forge signatures of frequently successful or otherwise well-known honest miners.

Definition 11 (Indistinguishability Against Time-Traveling Distinguishers). *Let $\mathcal{C}$ be a circuit class. We say two time-parameterized distributions $X_0(T), X_1(T)$ are **$\mathcal{C}$-time-traveling indistinguishable** if for all PPT distinguishers D, all times T, and all $F = \mathsf{poly}(\lambda)$:*

$$\Big| \Pr[D(x, \mathsf{st}_F) = 1 : x \leftarrow\!\!\$\, X_0(T), \mathsf{st}_F \leftarrow\!\!\$\, D_{\mathsf{fut}}(T, F)] \\ F - \Pr[D(x, \mathsf{st}_F) = 1 : x \leftarrow\!\!\$\, X_1(T), \mathsf{st}_F \leftarrow\!\!\$\, D_{\mathsf{fut}}(T, F)] \Big| \le \mathsf{negl}(\lambda)$$

The distributions are parameterized by a time T (i.e. number of blocks in the ledger). The distinguisher receives a future state which extends the state of the blockchain at time T by F blocks. Note that hiding is required against distinguishers which receive states extended by any polynomial number of blocks.

Definition 12. *(Hiding Against Time-Traveling Distinguishers) Let $\mathcal{C}$ be a circuit class. We say a commitment scheme* $(\mathsf{Com}, \mathsf{Rec}, \mathsf{Open})$ *is **$\mathcal{C}$-hiding against time-traveling distinguishers** if for all $R^* \in \mathcal{C}$ the following two distributions are $\mathcal{C}$-time-traveling indistinguishable:*

- $\{\mathsf{view}_{R^*}\left(\mathsf{Com}_T(1^\lambda, m_0), R^*_T\right)\}$
- $\{\mathsf{view}_{R^*}\left(\mathsf{Com}_T(1^\lambda, m_1), R^*_T\right)\}$

Definition 13. *(Extraction Against Time-Traveling Distinguishers) Let $\mathcal{C}$ be a circuit class. We say an commitment scheme* $(\mathsf{Com}, \mathsf{Rec}, \mathsf{Open})$ *with an F-time-traveling $\mathcal{C}$-extractor has an **F-time-traveling $\mathcal{C}$-extractor against time-traveling distinguishers** if for all $C^* \in \mathcal{C}$ the following two distributions are $\mathcal{C}$-time-traveling indistinguishable:*

- $\{\mathsf{view}_{C^*}(C^*_T(1^\lambda), \mathsf{Rec}_T(1^\lambda))\}$
- $\{\tilde{\tau} : (\tilde{\tau}, \tilde{m}) \leftarrow\!\!\$\, \mathsf{Ext}_T(1^\lambda, C^*, \mathsf{st}_F),\ \mathsf{st}_F \leftarrow\!\!\$\, D_{\mathsf{fut}}(T, F)\}$

Note that both hiding and extraction commitments against time-traveling distinguishers imply their non-time-traveling-distinguisher counterparts.

5 Two-Round Commitments with Time-Traveling Extractors

We construct a 2 round commitment which has an F-time-traveling extractor against time-traveling distinguishers and is hiding against time-traveling distinguishers. At a high level, the receiver non-interactively commits to an empty string using ideas from [30], which we informally refer to as a "GG commitment". It makes use of a threshold secret sharing scheme and a public key integrated encryption-signature scheme. Then the receiver and committer engage in a CDS protocol where the receiver gets the committer's message if and only if they can provide a witness that their non-interactive commitment was to a future state. Additionally, the committer sends a separate non-interactive commitment to their message.

5.1 Protocol

Our construction makes use of a threshold secret-sharing scheme $\mathsf{SS} = (\mathsf{Share}, \mathsf{Rec})$, a public key integrated encryption-signature scheme $\mathsf{HS} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Sign}, \mathsf{Vf})$, a Conditional Disclosure of Secrets scheme CDS, and a non-interactive statistically-binding commitment $(\mathsf{Com}_{ni}, \mathsf{Open}_{ni})$.

Given a state st, let $\mathcal{M}_\ell(\mathsf{st})$ be the set of miners which mined the last ℓ blocks of st. Also, let $\mathsf{pk}_{\mathsf{id}}$ be the public key of party P_{id}. The protocol parameters params are the length of the future state F, the window size $\mathsf{windowSize}$, the chain-quality parameters (ℓ, β), and the security parameter λ.

The language $\mathcal{L}_{\mathsf{params}}$ used for $\mathsf{CDS}_{\mathsf{params}}$ is defined as follows:

$$\mathcal{L}_{\mathsf{params}} = \left\{ (\mathsf{ggcom}, \mathsf{st}) : \begin{array}{c} \exists(\mathsf{st}_F, r_{ss}, (r_{\mathsf{id}})_{\mathsf{id} \in \mathcal{M}_\ell(\mathsf{st})}) \text{ such that} \\ (\mathsf{sh}_{\mathsf{id}})_{\mathsf{id} \in M_\ell(\mathsf{st}_R)} = \mathsf{Share}(\mathsf{st}_F, \ell, (1-\beta)\ell + 1; r_{ss}) \\ \mathsf{ggcom} = (\mathsf{Enc}(\mathsf{pk}_{\mathsf{id}}, \mathsf{sh}_{\mathsf{id}}; r_{\mathsf{id}}))_{\mathsf{id} \in M_\ell(\mathsf{st}_R)} \\ \mathsf{isvalidchain}(\mathsf{st}, \mathsf{st}_F) = 1 \\ |\mathsf{st}_F| \geq |\mathsf{st}| + F(\lambda) + 2\mathsf{windowSize} \end{array} \right\}. \quad (4)$$

Protocol $(\mathsf{Rec}, \mathsf{Com}, \mathsf{Open})$

Let $i_C = |\mathsf{st}_C|$ and $i_R = |\mathsf{st}_R|$ denote the size of the respective states of the committer and the receiver at the start of the protocol. Let $i_V = |\mathsf{st}_V|$ denote the size of the state of the verifier, who runs Open. Let $\mathsf{params} = (F, \mathsf{windowSize}, \ell, \beta, \lambda)$ be the parameters of the protocol.

Rec(params) The receiver does the following:
- Compute the secret-sharing $(\mathsf{sh}_{\mathsf{id}})_{\mathsf{id} \in M_\ell(\mathsf{st}_R)} \leftarrow \mathsf{Share}(0^{|i_R| + F(\lambda) + 2\mathsf{windowSize}}, \ell, (1-\beta)\ell + 1)$
- Compute $\mathsf{ggcom} \leftarrow (\mathsf{Enc}(\mathsf{pk}_{\mathsf{id}}, \mathsf{sh}_{\mathsf{id}}))_{\mathsf{id} \in M_\ell(\mathsf{st}_R)}$
- Compute $\mathsf{cds}_1 \leftarrow \mathsf{CDS}_{1,\mathsf{params}}(1^\lambda, (\mathsf{ggcom}, \mathsf{st}_R), \perp)$
- Send $(\mathsf{ggcom}, \mathsf{cds}_1, i_R)$ to the committer.

Com$(m, \tau_1, \mathsf{params})$ Upon receiving the first message $\tau_1 = (\mathsf{ggcom}, \mathsf{cds}_1, i)$, the committer checks if $i_C - i > \mathsf{windowSize}$, and aborts if so. Otherwise, it waits until the size of its state st_C is at least i, then does the following:
- Sample fresh randomness r_{cds}.
- Compute $\mathsf{cds}_2 \leftarrow \mathsf{CDS}_{2,\mathsf{params}}(1^\lambda, (\mathsf{ggcom}, \mathsf{st}_C[: i]), \mathsf{cds}_1, m; r_{cds})$
- Compute $(\mathsf{com}_{ni}, d_{ni}) \leftarrow \mathsf{Com}_{ni}(1^\lambda, m)$
- Send $(\mathsf{cds}_2, \mathsf{com}_{ni})$ to the receiver and keep (r_{cds}, d_{ni}) as the decommitment information.

Open$(\tau, m, d, \mathsf{params})$ Given the transcript $\tau = ((\mathsf{ggcom}, \mathsf{cds}_1, i), (\mathsf{cds}_2, \mathsf{com}_{ni}))$, decommitment information $d = (r_{cds}, d_{ni})$, and message m, the verifier waits until $i_V \geq i$. It outputs 1 if

$$\mathsf{cds}_2 = \mathsf{CDS}_{2,\mathsf{params}}(1^\lambda, (\mathsf{ggcom}, \mathsf{st}_V[: i]), \mathsf{cds}_1, m; r_{cds}) \wedge \mathsf{Open}_{ni}(1^\lambda, \mathsf{com}_{ni}, m, d)$$

and 0 otherwise.

5.2 Analysis

Proposition 1. *Assume that* HS *is a public key integrated encryption-signature scheme, that* CDS *is a Conditional Disclosure of Secrets protocol, that* $(\mathsf{Com}_{ni}, \mathsf{Open}_{ni})$ *is a non-interactive statistically-binding commitment scheme, that* SS *is a threshold secret-sharing scheme, and that* $\mathcal{G}_{\mathsf{ledger}}$ *has*

(β, ℓ)-chain-quality. Let t, F' be polynomially bounded functions as in Assumption 2 and let $\mathcal{C}_{t,F'}$ be the class of adversaries which receives an F'-future state and performs at most $t(\lambda)$ operations.

Assuming that no PPT adversary $(2t, \ell + 2\mathsf{windowSize}, F')$-breaks the $(1, F)$-security of $\mathcal{G}_{\mathsf{ledger}}$ (Assumption 2), the protocol $(\mathsf{Com}, \mathsf{Rec}, \mathsf{Open})$ is a statistically binding commitment scheme in the $\mathcal{G}_{\mathsf{ledger}}$-hybrid model with $\mathcal{C}_{t,F'}$-hiding against time-traveling distinguishers. Furthermore, it has an $(F + 2\mathsf{windowSize})$-time-traveling extractor against time-traveling distinguishers.

Proof. Statistical binding follows immediately from the statistical binding of Com_{ni}. We divide the remainder of the proof (hiding and time-traveling extraction) into two lemmas.

Lemma 1. *For every cheating (receiver, time-traveling distinguisher) pair that breaks hiding while performing t operations, there exists an adversary that $(2t, \ell + 2\mathsf{windowSize}, F')$-breaks the $(1, F)$-security of $\mathcal{G}_{\mathsf{ledger}}$.*

Proof. It suffices to show only hiding against time-traveling distinguishers, since hiding against non-time-traveling distinguishers follows from this. Consider any PPT $R^* \in \mathcal{C}_{t,F'}$. Without loss of generality, we can ignore those which would cause the committer to abort with $1 - \mathsf{negl}(\lambda)$ probability. First we show that

$$Pr\left[(\mathsf{ggcom}, \mathsf{st}_C[:i]) \in \mathcal{L}_{\mathsf{params}} : (\mathsf{ggcom}, \mathsf{cds}_1, i, \mathsf{cds}_2, \mathsf{com}_{ni}) \leftarrow \langle R^*, C\rangle(\mathsf{params})\right] \leq \mathsf{negl}(\lambda)$$

where $\mathsf{st}_C[:i]$ is the first i blocks of the committer's state at the point when they send their message. Note that the experiment is only over the executions where the committer does not abort.

Assume that this is not the case. Then we construct an adversary A which $(2t, \ell+2\mathsf{windowSize}, F')$-breaks the $(1, F)$-security of $\mathcal{G}_{\mathsf{ledger}}$. Given an F'-future state $\mathsf{st}_{F'}$, A computes $(\mathsf{ggcom}, \mathsf{cds}_1, i, \mathsf{cds}_2, \mathsf{com}_{ni}) \leftarrow \langle R^*(\mathsf{st}_{F'}), C\rangle(\mathsf{params})$ by internally simulating R^* and C. By assumption, $(\mathsf{ggcom}, \mathsf{st}_C[:i]) \in \mathcal{L}_{\mathsf{params}}$ with noticeable probability. When this occurs, there exists randomness r_{ss} and shares $(\mathsf{sh}_{\mathsf{id}})_{\mathsf{id} \in \mathcal{M}_\ell(\mathsf{st}_R)}$ such that $(\mathsf{sh}_{\mathsf{id}})_{\mathsf{id} \in \mathcal{M}_\ell(\mathsf{st}_R)} = \mathsf{Share}(\mathsf{st}_F, \ell, (1-\beta)\ell; r_{ss})$ where st_F is a valid extension of $\mathsf{st}_C[:i]$ with $F + 2\mathsf{windowSize}$ more blocks (i.e. $\mathsf{isvalidchain}(\mathsf{st}_F, \mathsf{st}_C[:i], F + 2\mathsf{windowSize}) = 1$). Furthermore, these shares are encrypted under the public keys $(\mathsf{pk}_{\mathsf{id}})_{\mathsf{id} \in \mathcal{M}_{(\ell)}(\mathsf{st}_C[:i])}$ of the miners of the last ℓ blocks of $\mathsf{st}_C[:i] = \mathcal{G}_{\mathsf{ledger}}.\mathsf{st}[:i]$. Since the committer does not abort, $i_C - i \leq \mathsf{windowSize}$. Since $\mathcal{G}_{\mathsf{ledger}}.\mathsf{size} - i_C \leq \mathsf{windowSize}$, we have $i \geq \mathcal{G}_{\mathsf{ledger}}.\mathsf{size} - 2\mathsf{windowSize}$.

Recall that in the $(2t, \ell + 2\mathsf{windowSize}, F')$-breakage of $(1, F)$-security experiment, A is given the secret keys $(\mathsf{sk}_{\mathsf{id}})_{\mathsf{id} \in \mathcal{M}_{(\ell+2\mathsf{windowSize})}(\mathcal{G}_{\mathsf{ledger}}.\mathsf{st})}$ of the miners for the last $\ell + 2\mathsf{windowSize}$ blocks of $\mathcal{G}_{\mathsf{ledger}}$. Since $i \geq \mathcal{G}_{\mathsf{ledger}}.\mathsf{size} - 2\mathsf{windowSize}$, these include the secret keys of $(\mathsf{pk}_{\mathsf{id}})_{\mathsf{id} \in \mathcal{M}_{(\ell)}(\mathsf{st}_C[:i])}$. Using these secret keys, A decrypts the shares $(\mathsf{sh}_{\mathsf{id}})_{\mathsf{id} \in M_\ell(\mathsf{st}_R)}$ and uses them to reconstruct st_F. We assume t to be large enough that the rest of A's computations (in particular decrypting ciphertexts and reconstructing the shares) take at most t operations. Since A is allowed $2t$ operations before the honest miners mine a single block, it can finish its computations before that occurs. Since $(\mathsf{ggcom}, \mathsf{st}_C[:i]) \in \mathcal{L}_{\mathsf{params}}$ by assumption, the state obtained is $F + 2\mathsf{windowSize}$ blocks longer than i, which is at most $2\mathsf{windowSize}$ blocks behind $\mathcal{G}_{\mathsf{ledger}}.\mathsf{st}$. Thus A obtains an F-future state st_F which is F blocks ahead of the global blockchain within the parameters of the $(2t, \ell + 2\mathsf{windowSize}, F')$-breakage $(1, F)$-security security game.

The rest of the proof proceeds in 3 hybrids, where H_0 is the commitment to m_0.

H_1 The committer computes cds_2 using m_1 instead of m_0, i.e.

$$\mathsf{cds}_2 \leftarrow \mathsf{CDS}_{2,\mathsf{params}}(1^\lambda, (\mathsf{ggcom}, \mathsf{st}_C[:i], \mathsf{cds}_1, m_1; r)$$

Indistinguishability against time-traveling distinguishers follows from the previous claim and the security of CDSin the plain model. In more detail, we can construct an adversary A for CDS in the plain model if this is not the case. A internally emulates the ledger, pausing the hybrid adversary and distinguisher to compute whatever future states they require. It computes the commitment transcript, forwarding the CDS messages between the commitment adversary and the CDS challenger. It outputs the distinguishers output.

H_2 In Com, compute com_{ni} using m_1 instead of m_0, i.e.

$$(\mathsf{com}_{ni}, d) \leftarrow \mathsf{Com}_{ni}(1^\lambda, m_1)$$

Indistinguishability against time-traveling distinguishers follows from a similar argument to the previous hybrid.

Observe that H_2 is an honest commitment to m_1.

Lemma 2. *The protocol* $(\mathsf{Com}, \mathsf{Rec}, \mathsf{Open})$ *has an* $(F{+}2\mathsf{windowSize})$*-time-traveling extractor against time-traveling distinguishers.*

Proof. Without loss of generality, the extractor knows the $\mathcal{G}_{\mathsf{ledger}}$ pointer for every party, since it acts as the adversary with respect to $\mathcal{G}_{\mathsf{ledger}}$. In particular, it knows $i_R = |\mathsf{st}_R|$ for the honest receiver R (this also implies it knows st_R, since $\mathsf{st}_R = \mathcal{G}_{\mathsf{ledger}}.\mathsf{st}[: i_R]$). Note that it could be the case that $i_R = \mathcal{G}_{\mathsf{ledger}}.\mathsf{size}$ exactly.

When given a future state st_F such that $\mathsf{isvalidchain}(\mathsf{st}_F, \mathcal{G}_{\mathsf{ledger}}.\mathsf{st}, F + 2\mathsf{windowSize})$, the extractor computes ggcom using st_F instead of 0s during Rec, i.e. it secret-shares st_F[12] and encrypts the resulting shares using the public keys of miners who mined the last ℓ blocks of st_R. Let r be the randomness it uses in the secret sharing and encryption. The extractor computes cds_1 using the witness (st_F, r), then sends $(\mathsf{ggcom}, \mathsf{cds}_1, i_R)$. It internally runs C^* and receives the Com message $(\mathsf{cds}_2, \mathsf{com}_{ni})$. Finally, it computes the CDS output for $\mathsf{cds}_1, \mathsf{cds}_2$ and outputs this alongside the transcript.

Over-Extraction It suffices to show that conditioned on the extracted transcript having a non-$\perp$ value, the extracted message is the value of the transcript. Consider the extracted transcript $\tau = ((\mathsf{ggcom}, \mathsf{cds}_1, i), (\mathsf{cds}_2, \mathsf{com}_{ni}))$ If τ has a value $\mathsf{val}(\tau) \neq \perp$, then for some randomness r_{cds_2} it holds that $\mathsf{cds}_2 = \mathsf{CDS}_{2,\mathsf{params}}(1^\lambda, (\mathsf{ggcom}, \mathcal{G}_{\mathsf{ledger}}.\mathsf{st}[: i]), \mathsf{cds}_1, \mathsf{val}(\tau); r_{\mathsf{cds}_2})$. Note that for extracted transcripts it holds that $(\mathsf{ggcom}, \mathcal{G}_{\mathsf{ledger}}.\mathsf{st}[: i]) \in \mathcal{L}_{\mathsf{params}}$ with witness (st_F, r). Therefore, by the correctness of CDS, the extracted message is $\mathsf{val}(\tau)$ with probability 1.

View Indistinguishability To show view indistinguishability against time-traveling distinguishers, we proceed in 4 hybrids, where H_0 is the real interaction.

H_1 Let $\mathcal{M}^H_\ell(\mathsf{st}_C)$ be a set of honest miners which mined blocks in the last ℓ blocks of st_C. The committer encrypts 0s under the keys of these parties instead of the shares.

Indistinguishability against time-traveling distinguishers follows from the semantic security of the public key integrated encryption-signature scheme HS in the plain model. In more detail,

[12] Without loss of generality we assume st_F is exactly $F + 2\mathsf{windowSize}$ blocks longer than st_R, since it can be truncated without affecting its validity.

we can construct an adversary A for HS in the plain model if this is not the case. A internally simulates the ledger, pausing the hybrid adversary and distinguisher to compute whatever future states they require. The public keys it uses for the honest parties in the ledger are chosen by the HS challenger, which also provides any signatures or ciphertexts under these keys required to indistinguishably emulate the ledger. It computes the commitment transcript, forwarding the appropriate ciphertexts from the HS challenger, which encrypts either 0s or the shares under the public keys $\mathsf{pk}_{\mathsf{id}}$ for $\mathsf{id} \in \mathcal{M}_\ell^H(\mathsf{st}_C)$. Finally, it outputs the output of the hybrid distinguisher.

H_2 In this hybrid, the committer receives and secret-shares st_F instead of sharing $0^{|i_R|+F(\lambda)+2\mathsf{windowSize}}$. Note that these have the same length.
Indistinguishability against time-traveling distinguishers follows from a similar argument as above, relying on the perfect security of secret-sharing and the (β,ℓ)-chain-quality of $\mathcal{G}_{\mathsf{ledger}}$. If this is not the case, we construct an adversary A for secret-sharing in the plain model by internally emulating the ledger, the hybrid adversary, and the hybrid distinguisher, including the future states provided to them. Note that the internally emulated ledger is indistinguishable from a real one, so it obeys the (β,ℓ)-chain-quality property of $\mathcal{G}_{\mathsf{ledger}}$. Therefore there are at least $\beta\ell$ honest miners in $\mathcal{M}_\ell^H(\mathsf{st}_C)$. Since this is the case, A requires no more than $(1-\beta)\ell$ shares from the secret-sharing challenger, which is what it receives. A therefore can compute the transcript and output what the hybrid distinguisher does.

H_3 In this hybrid, the receiver computes cds_1 using the witness $(\mathsf{st}_F, r_{ss}, (r_{\mathsf{id}})_{\mathsf{id}\in\mathcal{M}_\ell(\mathsf{st})})$, where r_{ss} is the randomness used to secret-share st_F and r_{id} is the randomness used to encrypt the share $\mathsf{sh}_{\mathsf{id}}$ for each $\mathsf{id} \in \mathcal{M}_\ell(\mathsf{st})$.
Indistinguishability against time traveling distinguishers follows from the CDS receiver's security in the plain model. As before, if this does not hold, we can construct an adversary A that breaks the CDS receiver's security in the plain model. It internally emulates the ledger, computing future states as necessary to compute the hybrid adversary's view while forwarding messages to and from the CDS challenger.

Observe that H_3 precisely produces the transcripts produced by the extractor.

6 Three-Round Time-Traveling Zero-Knowledge

In Section 6.1 we describe a three-round argument that satisfies time-traveling ZK. Looking ahead, this protocol is not an argument of knowledge. In Section 6.2, we extend the protocol from Section 6.1 to be a time-traveling argument of knowledge.

6.1 Three-Round Time-Traveling Zero-Knowledge Argument

We follow the FLS [21] paradigm that allows to transform a proof for $x \in \mathcal{L}$ into a witness-indistinguishable proof for "$x \in L$ or I know a trapdoor". To construct a three-round argument for $\mathcal{L}$, we use a three-round WI proof-of-knowledge (WIPOK) and use the current state of the $\mathcal{G}_{\mathsf{ledger}}$ to generate a trapdoor. In particular, for some parameter F, prover gives a three-round WI for the statement "$x \in \mathcal{L}$ OR I know a valid state st_F of $\mathcal{G}_{\mathsf{ledger}}$ that extends the current state by F blocks". We give a formal version of the protocol below.

First we define the NP language $\mathcal{L}_{\mathsf{wipok}}$ which is used in the protocol.

$$\mathcal{L}_{\mathsf{wipok}} = \left\{ (i,F,x) : \begin{array}{c} \exists(\mathsf{st}, w) \text{ such that} \\ \text{either } R(x,w)=1 \\ \text{or } (\mathsf{isvalidchain}(\mathcal{G}_{\mathsf{ledger}}.\mathsf{st}[:i], \mathsf{st}, F + 2\mathsf{windowSize}) \end{array} \right\}. \quad (5)$$

Protocol $\langle P, V\rangle$

Common input: An instance x from $\mathcal{L}$ with witness relation $R_{\mathcal{L}}$, security parameter λ, time-out parameter k and parameter F.

Auxiliary input to Prover: witness w such that $R_{\mathcal{L}}(x, w) = 1$, and size of prover's local state from $\mathcal{G}_{\mathsf{ledger}}$ $i_P = |\mathsf{st}_P|$.

Auxiliary input to Verifier: size of verifier's local state from $\mathcal{G}_{\mathsf{ledger}}$ $i_V = |\mathsf{st}_V|$.

Protocol: Before sending any message, either party checks whether their state has expanded by more than k blocks. If so, they abort.

1. $P \to V$: On input i_P, P sends over i_P along with the first message π_1 of WIPOK to V.
2. $V \to P$: Upon receiving (i_P, π_1), V checks whether $|i_P - i_V| > \mathsf{windowSize}$, if so it aborts. It then sends the second message π_2 of WIPOK.
3. $P \to V$: P computes the third message as follows:
   - P samples an instance and a witness (x, w) such that $(x, w) \in R$.
   - P computes the third message π_3 of WIPOK for the statement $(i_P, F, x) \in \mathcal{L}_{\mathsf{wipok}}$. The honest prover uses w as the witness. Looking ahead, our time-traveling simulator will use a valid extension st_F of st_P.
4. Output of V: If the size of current state of V is less than i_P, V waits until the size of its state is at least i_P.[a] V verifies the WIPOK proof w.r.t. the statement (i_P, F, x).

[a] This ensures that it has the statement defined by (i_P, F).

Theorem 1. *Let k be an upper-bound on the numbers of blocks added in an honest execution of $\langle P, V\rangle$. Let t, F be polynomially bounded function such that no PPT A $3t$-breaks the (k, F)-security of $\mathcal{G}_{\mathsf{ledger}}$ (as per Definition 1). Further, assuming the witness-indistinguishability of* WIPOK*, the protocol $\langle P, V\rangle$ is a 3-round argument in the $\mathcal{G}_{\mathsf{ledger}}$ model where soundness holds against cheating prover P^* performing t operations and $(F + 2\mathsf{windowSize})$-time-traveling zero-knowledge holds against all PPT V^*.*

Completeness. The time-out parameter is set so that honest parties will complete the execution within k blocks being added. Given this, the completeness follows from the completeness of the WIPOK.

Time-traveling Zero-Knowledge. Let V^* be a cheating prover such that its initial state is st_V. Let x be some statement in $\mathcal{L}$. Then, we design a simulator Sim that gets the current states st_P and st_{V^*} of the parties, and a valid extension st_F of st_P as advice. Such a simulator interacts with V^* identically to the honest prover on state st_P except

- it computes the third message of WIPOK for the statement $(|\mathsf{st}_P|, F, x)$ using st_F as witness.
- it forwards all queries by V^* to the ledger to its own oracle $\mathcal{G}_{\mathsf{ledger}}$.

Sim then outputs the entire transcript of the interaction. We claim that the transcript output by Sim is indistinguishable from the real world interaction to even distinguishers that have access to the $\mathcal{G}_{\mathsf{ledger}}$. If there exists a (V, D) pair for which the above is not true, then we can construct an adversary that breaks the WI of the WIPOK: such an A internally emulates the ledger, acting as the adversary in the WI game.

Soundness. Let us assume for simplicity that the cheating prover P^* promises to not access the $\mathcal{G}_{\mathsf{ledger}}$ during the execution of the protocol. Such a P^* cannot use the ledger to maintain state during the execution (e.g., prover sends the third message only if the transcript so far is not present on the ledger, and submits the partial transcript to be posted on the ledger), and hence can be freely rewound. Then, keeping the first message same we run two executions of the prover sending different second messages. If P^* succeeds in convincing V about x with probability ϵ, then with probability roughly ϵ^2 the two transcripts are accepting. This allows us to, via special-soundness of WIPOK, extract a witness for WIPOK with probability ϵ^2. It remains to show that the extracted witness is a witness for x. We claim that the extracted witness cannot be a future state as otherwise we would have constructed an A that contradicts the hardness of the underlying mining puzzle. However, note that the above reduction crucially relies on being able to rewind P^*. As discussed before, this is not straightforward for P^* that query the $\mathcal{G}_{\mathsf{ledger}}$.

To prove soundness against provers that have access to $\mathcal{G}_{\mathsf{ledger}}$, we take almost the same approach, except that we mimic rewinding by relying on two things (a) given oracle access to $\mathcal{O}_{\mathsf{blockify}}$, a miner can compute a fork of size k that is indistinguishable from the one maintained by $\mathcal{G}_{\mathsf{ledger}}$ to P^*, (b) such a miner (by (k, F)-security of the $\mathcal{G}_{\mathsf{ledger}}$) cannot mine blocks way ahead in the future, thereby allowing us to argue correctness of the extracted witness. A more formal analysis follows.

Lemma 3. *For every cheating prover that breaks soundness while performing t operations, there exists an $A = (A_0, A_1)$ that $3t$-breaks the (k, F)-security of $\mathcal{G}_{\mathsf{ledger}}$.*

Proof. Given P^* with state st_P, let $i_P = |\mathsf{st}_P|$. We build a ledger adversary A that internally runs two copies of P^* acting as the verifier. Let the two copies be P_1^* and P_2^*, let $\mathcal{O}_1$ and $\mathcal{O}_2$ be the ledger functionality that $A = (A_0, A_1)$ simulates for P_1^* and P_2^*. Specifically, $A = (A_0, A_1)$ behaves as follows:

- A_0 ensures that $\mathcal{O}_1$ and $\mathcal{O}_2$ behave identically as $\mathcal{G}_{\mathsf{ledger}}$ until P_1^* and P_2^* send their first message. Note that this ensures that the first message sent by P_1^* and P_2^* is the same, in fact their entire view is identical so far. Let i be the current state of P_1^* and P_2^* and let π_1 be the first message.
- From this point on, A_1 takes over: in particular, it continues the execution with P_1^* and P_2^*. It forwards any query to $\mathcal{O}_1$ by P_1^* to $\mathcal{G}_{\mathsf{ledger}}$, and answers queries to $\mathcal{O}_2$ by locally simulating a ledger consistent with i by relying on access to $\mathcal{O}_{\mathsf{blockify}}$.
- A_1 sends two different messages $\pi_2 \neq \pi_2'$ and receives $((x, i_P, F), \pi_3)$ and $((x', i_P, F), \pi_3')$.
- If either of the two transcripts (π_1, π_2, π_3) and (π_1, π_2', π_3') is rejecting then A_1 aborts. Otherwise, A_1 computes a witness w from x using the two transcripts, and outputs w.

To conclude the soundness proof, we claim that w cannot be a witness for the statement (i_P, F). To argue this, we first note that A_1 performs at most $2t$ operations in running the two instances of P^* and we assume that t is large enough so that rest of A_1's computation (in particular, the computing the witness from the two transcripts) takes at most t operations. Overall, A performs $3t$ operations. Furthermore, since the verifier did not abort, $i_P \geq i_V - \mathsf{windowSize}$, and since $i_V \geq \mathcal{G}_{\mathsf{ledger}}.\mathsf{size} - \mathsf{windowSize}$, we have $i_P \geq \mathcal{G}_{\mathsf{ledger}}.\mathsf{size} - \mathsf{windowSize}$. Therefore if w is a witness for the statement (i_p, F), it contains a future state st_F such that $\mathsf{isvalidchain}(\mathcal{G}_{\mathsf{ledger}}.\mathsf{st}, \mathsf{st}_F, F + 2\mathsf{windowSize} - 2\mathsf{windowSize})$. Now, the claim follows by the assumption that no PPT A $3t$-breaks the (k, F)-security of $\mathcal{G}_{\mathsf{ledger}}$ (as per Definition 1).

Remark 3 (On achieving knowledge-soundness.). For knowledge-soundness, we need a construct a knowledge-extractor that given access to the cheating prover, outputs a witness along with an accepting transcript that is indistinguishable from the real world to a distinguisher that has access to $\mathcal{G}_{\mathsf{ledger}}$. This, in particular, means that the extractor should finish in roughly the same time (in terms of number of blocks) as the cheating prover. In particular, rewinding-based extractor will need to run all threads in parallel. Given that rewinding-based extraction in constant rounds is known to require an *expected* polynomial time extractor, there is no fixed polynomial upper-bound on the number of threads needed for extraction. Running an exponential number of threads in parallel will allow extraction, but such an exponential-sized reduction can take over the ledger and we cannot argue correctness of the witness like before.

6.2 Three-Round Time-Traveling Zero-Knowledge Argument-of-Knowledge

To obtain an argument-of-knowledge, we combine the 3-round ZK argument $\langle P, V \rangle$ from Section 6.1 with the 2-round extractable commitment $(\mathsf{Com}, \mathsf{Rec})$ from Section 5 in a natural way: prover commits to the witness using $(\mathsf{Com}, \mathsf{Rec})$ and proves using $\langle P, V \rangle$ that it has indeed committed to the witness.

For some functions $F' < F$, let $\langle P, V \rangle$ be a 3-round F-time-traveling zero-knowledge for the language $\mathcal{L}'$

$$\mathcal{L}' = \{(\mathsf{com}_1, \mathsf{com}_2, x) : \exists (w, d) \text{ such that } \mathsf{Open}((\mathsf{com}_1, \mathsf{com}_2), w, d) = 1 \wedge R(x, w) = 1\} \ . \tag{6}$$

Further, let $(\mathsf{Com}, \mathsf{Rec})$ be 2-round F'-time-traveling extractable commitment from Section 5.

Protocol $\langle P_{\mathsf{zkaok}}, V_{\mathsf{zkaok}} \rangle$

Common input: An instance x from $\mathcal{L}$ with witness relation $R_{\mathcal{L}}$, security parameter λ, time-out parameter k and parameter F.

Auxiliary input to Prover: witness w such that $R_{\mathcal{L}}(x, w) = 1$, and size of prover's local state from $\mathcal{G}_{\mathsf{ledger}}$ $i_P = |\mathsf{st}_P|$.

Auxiliary input to Verifier: size of verifier's local state from $\mathcal{G}_{\mathsf{ledger}}$ $i_V = |\mathsf{st}_V|$.

Protocol: Before sending any message, either party checks whether their state has expanded by more than k blocks. If so, they abort.

1. $P_{\mathsf{zkaok}} \to V_{\mathsf{zkaok}}$: On input i_P, P_{zkaok} sends over i_P along with the first message π_1 of $\langle P, V \rangle$ to V_{zkaok} acting as P.

2. $V_{\mathsf{zkaok}} \to P_{\mathsf{zkaok}}$: Upon receiving (i_P, π_1), V_{zkaok} checks whether $|i_P - i_V| > \mathsf{windowSize}$, if so it aborts. It then sends the second message π_2 of $\langle P, V \rangle$ acting as V, and sends the first message com_1 acting as Rec.
3. $P_{\mathsf{zkaok}} \to V_{\mathsf{zkaok}}$: P computes the third message as follows:

- P_{zkaok} samples an instance and a witness (x, w) such that $(x, w) \in R$.
- P_{zkaok} commits to the witness w acting as Com, let com_2 be the message computed by Com and d be the corresponding decommitment.
- P_{zkaok} computes the third message π_3 of $\langle P, V\rangle$ for the statement $(\mathsf{com}_1, \mathsf{com}_2, x) \in \mathcal{L}'$. The honest prover uses (w, d) as the witness.

4. Output of V_{zkaok}: If the size of current state of V_{zkaok} is less than i_P, V_{zkaok} waits until the size of its state is at least i_P. V_{zkaok} verifies the WIPOK proof w.r.t. the statement $(\mathsf{com}_1, \mathsf{com}_2, i_P, F, x)$.

Proposition 2. *Let k be an upper-bound on the numbers of blocks added in an honest execution of $\langle P, V\rangle$. Let t, ℓ, F', F be polynomially bounded functions such that no PPT A $(3t, \ell, F')$-breaks the (k, F)-security of $\mathcal{G}_{\mathsf{ledger}}$ (as per definition 2). Assume* $(\mathsf{Com}, \mathsf{Rec}, \mathsf{Open})$ *is hiding against all PPT receivers which perform at most $t(\lambda)$ operations and that it has an F'-time-traveling extractor.*

Then $\langle P_{\mathsf{zkaok}}, V_{\mathsf{zkaok}}\rangle$ is a 3-round argument in the $\mathcal{G}_{\mathsf{ledger}}$ model such that $(F + 2\mathsf{windowSize})$-time-traveling zero-knowledge holds against all PPT verifiers which perform at most $t(\lambda)$ operations. Furthermore, it has an F'-time-traveling extractor against every cheating prover P^* performing at most $t(\lambda)$ operations.

Time-Traveling Zero-Knowledge. Let V^* be a cheating prover such that its initial state is st_V. Let x be some statement in $\mathcal{L}$. Then, we design a simulator Sim that gets the current states st_P and st_{V^*} of the parties, and a valid extension $\mathsf{st}_F = (B_1, \ldots, B_F)$ of st_P as advice. Such a simulator interacts with V^* identically to the honest prover on state st_P except

- it runs the $(F + 2\mathsf{windowSize})$-time-traveling simulator for $\langle P, V\rangle$ to compute the third message of $\langle P, V\rangle$.
- it commits to a random string inside com_2 (note the honest prover commits to the witness for x inside com_2).
- it forwards all queries by V^* to the ledger to its own oracle $\mathcal{G}_{\mathsf{ledger}}$.

Sim then outputs the entire transcript of the interaction. We claim that the transcript output by Sim is indistinguishable from the real world interaction to even distinguishers that have access to the $\mathcal{G}_{\mathsf{ledger}}$. This follows from $(F + 2\mathsf{windowSize})$-time-traveling ZK against all PPT verifiers, and the hiding of $\langle C, R\rangle$ against R^* that performs $t(\lambda)$ operations.

Argument-of-Knowledge. Let P^* be some cheating prover. We need to exhibit a knowledge-extractor K that on input the current state st_P of P^*, and a future state st' such that $\mathsf{isvalidchain}(\mathsf{st}_P, \mathsf{st}', F') = 1$, outputs a tuple (τ', w') of transcript and a witness such that

1. τ' is indistinguishable from $\langle P^*, V_{\mathsf{zkaok}}\rangle$ to even distinguishers having access to $\mathcal{G}_{\mathsf{ledger}}$,
2. $\Pr[R(x', w') = 1] \approx \Pr[\langle P^*, V_{\mathsf{zkaok}}\rangle = 1]$ where x' is the statement sampled by P^* in τ'.

The extractor K is as follows: it interacts with P^* as the honest verifier V_{zkaok} except that instead of computing com_1 acting as the honest receiver Rec, it uses the F'-time-traveling extractor of $(\mathsf{Com}, \mathsf{Rec})$ to compute com_1 using st'. Let $\tau' = (\pi_1, (\pi_2, \mathsf{com}_1), (\pi_3, x, \mathsf{com}_2))$ be the transcript of the interaction between K and P^*. K then runs the extractor for $(\mathsf{Com}, \mathsf{Rec})$ on $(\mathsf{com}_1, \mathsf{com}_2)$ and outputs whatever it extracted as w'.

Claim 1 *τ' is indistinguishable from $\langle P^*, V_{\mathsf{zkaok}}\rangle$ to even distinguishers having access to $\mathcal{G}_{\mathsf{ledger}}$.*

Proof. The only difference in both distributions is the second message received by P^*, and more specifically the first message com_1 of $(\mathsf{Com}, \mathsf{Rec})$. In $\langle P^*, V_{\mathsf{zkaok}}\rangle$, com_1 is computed by R whereas K computes com_1 acting as the extractor for $(\mathsf{Com}, \mathsf{Rec})$. Then, the claim follows from the indistinguishability of the transcripts produced by the extractor of $\langle C, R\rangle$.

Claim 2 $\Pr[R(x', w') = 1] \approx \Pr[\langle P^*, V_{\mathsf{zkaok}}\rangle = 1]$ *where* x' *is the statement sampled by* P^* *in* τ'*.*

Firstly, note again by indistinguishability of the transcripts produced by the extractor of $\langle C, R\rangle$, we have

$$| \Pr[\tau' \text{ is accepting }] - \Pr[\langle P^*, V_{\mathsf{zkaok}}\rangle = 1] | \leq \mathsf{negl}(\lambda) \ . \tag{7}$$

All it remains to show is that whenever τ' is accepting, w' is actually the witness. Assume that P^* doesn't break soundness of $\langle P, V\rangle$ while interacting with K. Then, since τ' is accepting, we know that com_2 is well-formed and furthermore is a commitment to the witness for x'. Then, by the correctness of the extractor of $\langle C, R\rangle$, we have that $R(x', w') = 1$. All that remains to be shown is that P^* doesn't break soundness of $\langle P, V\rangle$ when interacting with K. For sake of contradiction, assume that P^* indeed breaks soundness of $\langle P, V\rangle$. Then we can construct an adversary $A = (A_0, A_1)$ that $(3t, \ell, F')$-breaks the (k, F)-security of $\mathcal{G}_{\mathsf{ledger}}$, similar to the proof of Lemma 3. The only difference from the previous proof is that the constructed adversary A_1 must have access to an F'-future state in order to run the extractor.

7 Three Round Secure Two-Party Computation with a Time-Traveling Simulator

At a high level, our approach is to compile a two round semi-malicious secure two-party computation protocol into a malicious one using the tools with built so far. We instantiate the semi-malicious protocol using a two round oblivious transfer protocol and a garbled circuit. This blueprint was also followed in [2].

7.1 Building Blocks

We use protocols with various future state parameters as building blocks. Consider the following hierarchy of polynomially bounded functions,

$$F_1 < F_2 < F_3 \tag{8}$$

We use the following building blocks:

1. three round delayed-input ZKAoK $\langle P, V\rangle$ (from Section 6.2) such that
   (a) F_3-time-traveling simulator for ZK,
   (b) F_2-time-traveling knowledge-extractor for cheating provers doing t operations.
2. two round commitment scheme $(\mathsf{Com}, \mathsf{Rec}, \mathsf{Open})$ (from Section 5) such that
   (a) Hiding against (R^*, D^*) where R^* performs t operations and D^* is F_2-time-traveling distinguisher.
   (b) An F_1-time-traveling extractor, produces transcripts indistinguishable to F_3-time-traveling distinguishers.
3. Standard cryptographic tools: a two round oblivious transfer protocol $\mathsf{OT} = (\mathsf{OT}_1, \mathsf{OT}_2)$, two round conditional disclosure of secret protocol $\mathsf{CDS} = (\mathsf{CDS}_1, \mathsf{CDS}_2)$, garbled circuit $\mathsf{GC} = (\mathsf{Garble}, \mathsf{Eval})$, a PRF family F, a statistically binding non-interactive commitment SBCom.

7.2 NP languages.

We consider two NP languages. First, $\mathcal{L}_{\mathsf{cds}}$ parameterized by OT, $(\mathsf{Com}, \mathsf{Rec}, \mathsf{Open})$ and PRF family F

$$\mathcal{L}_{\mathsf{cds}} = \left\{ (\mathsf{ot}_1, \mathsf{com}_1, \mathsf{com}_2, c) : \begin{array}{c} \exists (K, x, r) \text{ such that} \\ \mathsf{com}_2 = \mathsf{Com}(1^\lambda, \mathsf{com}_1, K; r) \\ \mathsf{ot}_1 = \mathsf{OT}_1(x; F(k,1)) \\ c = x \oplus F(K, 2) \end{array} \right\} . \tag{9}$$

Intuitively, $\mathcal{L}_{\mathsf{cds}}$ contains the transcripts of the first two rounds where P_2 behaves honestly. Second, $\mathcal{L}_{\mathsf{zk}}$ parameterized by OT, CDS and GC

$$\mathcal{L}_{\mathsf{zk}} = \left\{ (c_1, \mathsf{ot}_1, \mathsf{ot}_2, \mathsf{cds}_1, \mathsf{cds}_2, \mathsf{stmt}) : \begin{array}{c} \exists (x, r_{\mathsf{gc}}, r_{\mathsf{cds}}, r_{\mathsf{ot}}, r_{\mathsf{sbcom}}) \text{ such that} \\ \mathsf{ot}_2 = \mathsf{OT}_2(1^\lambda, \mathsf{ot}_1, \{\vec{l_i}\}_{i \in [n]}; r_{\mathsf{ot}}) \\ \mathsf{cds}_2 = \mathsf{CDS}_2(\mathsf{cds}_1, \mathsf{stmt}, C; r_{\mathsf{cds}}) \\ ((\vec{l_1}, \ldots, \vec{l_n}), C) = \mathsf{Garble}(f(x, \cdot); r_{\mathsf{gc}}) \\ c_1 = \mathsf{SBCom}(x; r_{\mathsf{sbcom}}) \end{array} \right\} . \tag{10}$$

Intuitively, $\mathcal{L}_{\mathsf{zk}}$ contains the transcripts where P_1 behaves honestly.

7.3 Protocol

Protocol $\mathit{\Pi}$

1. P_1, on input x_1, computes a non-interactive commitment $c_1 \leftarrow\$ \mathsf{SBCom}(x_1; r_{\mathsf{sbcom}})$ to its input, samples the first message com_1 of the two-round commitment by running Rec, and computes the first message zk_1 of ZKAoK acting as P. It sends $m_1 = (c_1, \mathsf{com}_1, \mathsf{zk}_1)$ to P_2.
2. On receiving m_1, P_2 runs Com computes a commitment $\mathsf{com}_2 = \mathsf{Com}(\mathsf{com}_1, K; r)$ to a random PRF key K using randomness r. It derives randomness $r_1 = F(K, 1)$ and $r_2 = F(K, 2)$ from the PRF key K. It then prepares first message $\mathsf{ot}_1 = \mathsf{OT}_1(x_2; r_1)$ of OT w.r.t. its input while using randomness r_1. Further, it computes $c = x_2 \oplus r_2$. Next, it computes the first message of CDS for the statement $\mathsf{stmt} = (\mathsf{ot}_1, \mathsf{com}_1, \mathsf{com}_2, c) \in \mathcal{L}_{\mathsf{cds}}$ using witness $w = (K, x_2, r)$, that is, $\mathsf{cds}_1 \leftarrow \mathsf{CDS}_1(\mathsf{stmt}, w)$ It also prepares the second message zk_2 of ZKAoK acting as V. It sends $m_2 = (\mathsf{com}_2, c, \mathsf{ot}_1, \mathsf{cds}_1, \mathsf{zk}_2)$.
3. On receiving m_2, P_3 runs $((\vec{l_1}, \ldots, \vec{l_n}), \hat{C}) \leftarrow\$ \mathsf{Garble}(C)$ where $\vec{l_i} = (l_i^0, l_i^1)$ are the two wire-labels for i-th input wire, and C is a garbled circuit for a circuit that computes $f(x_1, \cdot)$. It then prepares $\mathsf{ot}_2 \leftarrow\$ \mathsf{OT}_2(\mathsf{ot}_1, \vec{l})$, and computes $\mathsf{cds}_2 \leftarrow\$ \mathsf{CDS}_2(\mathsf{stmt}, \hat{C})$. Finally, it computes the third message of zk_3 for the statement $(c_1, \mathsf{ot}_1, \mathsf{ot}_2, \mathsf{cds}_1, \mathsf{cds}_2, \mathsf{stmt}) \in \mathcal{L}_{\mathsf{zk}}$. It sends $m_3 = (\mathsf{ot}_2, \mathsf{cds}_2, \mathsf{zk}_3)$
4. **Output Computation:** P_2 learns wire-labels $\{l_i^{b_i}\}_{i \in [n]}$ where b_i is the i-th bit of x_2 via OT, learns the CDS secret $\hat{C}$ from $(\mathsf{cds}_1, \mathsf{cds}_2)$, and then evaluates the garbled circuit $\hat{C}$ using these labels to learn the output y.

Theorem 2. *Assuming the building blocks with appropriate security as described in Section 7.1, there exists a two-party protocol $\mathit{\Pi}$ that securely computes any 2-party functionality f with F_3-time-traveling simulation.*

7.4 Security Proof

We prove Theorem 2 in two parts. First, we consider the case that P_2 is corrupted, and then discuss the case that P_1 is corrupted.

Case 1: P_2 is corrupted. Let A be the adversary that corrupts P_2. We describe a F_3-time-traveling simulator Sim_2 next. Let st_1 and st_2 be the state of P_1 and P_2 respectively. Sim_2 is given $(\mathsf{st}_1, \mathsf{st}_2)$ as inputs and st_{F_3} as auxiliary input where $\mathsf{isvalidchain}(\mathsf{st}_1, \mathsf{st}_{F_3}) = 1$ and $|\mathsf{st}_{F_3}| - |\mathsf{st}_1| \geq F_3$. .

Simulator Sim_2*:* Let st_{F_1} be a prefix of st_{F_3} such that $|\mathsf{st}_{F_1}| - |\mathsf{st}_1| = F_1$. It runs the F_3-time-traveling simulator to simulate the 3-round ZK argument. In parallel, it uses the F_1-time-traveling extractor for $\langle C, R \rangle$ to extract from P_2. Let the extracted value be K. If $K = \perp$, then Sim_2 aborts. Otherwise, it extracts from P_2 in two ways:

1. First computes $x'_2 = s \oplus F(K, 2)$ where s is the string sent by P_2 in the second round.
2. Next, it recovers x''_2 from the transcript of OT using $F(K, 1)$ as the randomness.

If $x'_2 \neq x''_2$ then set $\hat{C}$ to $\perp$. Otherwise, query the ideal functionality on x'_2 to get response y. Then the rest of the simulation proceeds as follows: It runs the simulator for the Garbled Circuit to generate a simulated circuit along with wire labels. Specifically, $(\hat{C}, \{l_i\}_{i \in [n]}) \leftarrow\$ \mathsf{Sim}_{gc}(1^\lambda, \phi(C), y)$ where $\phi(C)$ is topology of the circuit C that computes $f(x_1, \cdot)$. It computes the second message of OT using labels $\{\vec{l_i}\}_{i \in [n]}$ where $\vec{l_i} = (l_i, l_i)$. Finally the CDS messages are honestly computed using $\hat{C}$ as the secret. Finally, all queries by P_2 (or A) to the $\mathcal{G}_{\mathsf{ledger}}$ are forwarded by Sim_2 to its own oracle.

Lemma 4. *The joint view of P_1 and A in the real world is computationally indistinguishable from view of Sim_2 and A in the ideal world, even to distingusihers having access to $\mathcal{G}_{\mathsf{ledger}}$.*

Proof. The proof follows via a sequence of hybrids.

Hybrid H_0: This hybrid corresponds to the real world. The output of this hybrid is the joint view of P_1 and A.

Hybrid $H_1(\mathsf{st}_{F_3})$: This hybrid is identical to H_0 except we run the F_3-time-traveling simulator for 3-round ZK argument using st_{F_3} as advice. Note that the simulator for $\langle P, V \rangle$ requires st_{F_3} only in the third message, and in particular, the first message identical to P.

Claim 3 *Hybrids H_0 and H_1 are computationally indistinguishable assuming the F_3-time-traveling ZK of $\langle P, V \rangle$.*

Hybrid $H_2(\mathsf{st}_{F_3})$: This hybrid is identical to H_1 except we use the prefix st_{F_1} of st_{F_3} as advice to the time-traveling extractor for $(\mathsf{Com}, \mathsf{Rec})$ to compute the first message. In particular, P_1 acts as Rec in H_1 but in H_2 it acts as the time-traveling extractor of $(\mathsf{Com}, \mathsf{Rec})$. After receiving the second message from P_2, the hybrid runs the extractor of $(\mathsf{Com}, \mathsf{Rec})$ to extract a value from P_2. Let the extracted value be K. If $K \neq \perp$ then define x'_2 and x''_2 as defined by Sim_2.

Claim 4 *Hybrids H_1 and H_2 are computationally indistinguishable assuming the F_3-time-traveling indistinguishability of transcripts produced by F_1-time-traveling extractor of $(\mathsf{Com}, \mathsf{Rec})$.*

Proof. For every distinguisher D_{2pc} that distinguishes H_1 and H_2, we build a cheating committer C^* and a F_3-time-traveling distinguisher D that together distinguish the transcripts produced by F_1-time-traveling extractor of $(\mathsf{Com}, \mathsf{Rec})$ from an interaction of $\langle C^*, \mathsf{Rec}\rangle$.

C^* on input com_1, internals simulates an interaction with P_2. Upon receiving message $m_2 = (\mathsf{com}_2, c, \mathsf{ot}_1, \mathsf{cds}_1, \mathsf{zk}_2)$ it outputs com_2. Now, the distinguisher D on input st_{F_3} can complete the interaction with P_2. Note that, when com_1 is sampled by Rec (C^*, D) simulate H_1, and when com_1 is sampled by the extractor of $(\mathsf{Com}, \mathsf{Rec})$, (C^*, D) simulate H_2. The claim follows.

Claim 5 *If $x_2' \neq x_2''$ in H_2, then assuming the correctness of extractor of* $(\mathsf{Com}, \mathsf{Rec})$ *and uniqueness property of OT transcripts,*[13] *then the CDS statement* stmt *in H_2 is such that* $\mathsf{stmt} \notin \mathcal{L}_{\mathsf{cds}}$.

Proof. Let us assume that $\mathsf{stmt} \in \mathcal{L}_{\mathsf{cds}}$. This means that $(\mathsf{com}_1, \mathsf{com}_2)$ is well-formed commitment, and furthermore com_2 commits to a PRF key k and uses $F(k, 1)$ as the randomness to compute ot_2. By the over-extractability of the extractor for $(\mathsf{Com}, \mathsf{Rec})$, we have that it correctly extracts the value k. Let x_2' be the value recovered from ot_2 using randomness $F(k, 1)$. Finally, let $x_2'' = c \oplus F(k, 2)$ where c is the string that P_2 sends in the second round. Since $\mathsf{stmt} \in \mathcal{L}_{\mathsf{cds}}$, we can then infer that $x_2' = x_2''$, which contradicts the hypothesis.

Hybrid $H_3(\mathsf{st}_{F_3})$: This hybrid is identical to H_2 except that the non-interactive commitment sent in the first round. In particular, in this hybrid c_1 computed as a commitment to 0 (whereas in H_2 it was a commitment to x_1).

Claim 6 *Assuming the hiding of* SBCom*, the output of H_2 and H_3 are indistinguishable.*

Further, we also have that if $x_2' \neq x_2''$ in H_3 then the CDS statement is false. The following claim follows from almost the same argument as done in the proof of Claim 5

Claim 7 *If $x_2' \neq x_2''$ in H_3, the CDS statement* stmt *is such that* $\mathsf{stmt} \notin \mathcal{L}_{\mathsf{cds}}$.

Hybrid $H_4(\mathsf{st}_{F_3})$: This hybrid is identical to H_3, except that the second message of CDS is computed differently. In particular, whenever $x_2' \neq x_2''$, we compute cds_2 using the secret $\perp$. The indistinguishability follows from (a) if $x_2' \neq x_2''$ then the cds statement is false, (b) for false statements, cds_2 doesn't reveal the secret to even malicious receivers.

Claim 8 *Assuming the security of* CDS*, the output of H_3 and H_4 are indistinguishable.*

Hybrid $H_5(\mathsf{st}_{F_3})$: This hybrid is identical to H_4, except that the second message of OT is computed differently. In particular, let $(\hat{C}, \{\vec{l_i}\}_{i\in[n]})$ be the output of Garble. Then, whenever, $x_2' = x_2''$, we compute ot_2 using labels corresponding to x_2'. In particular, define $\vec{k_i} = (l_i^{b_i}, l_i^{b_i})$ where b_i is the i-th bit of x_2' and $\vec{l_i} = (l_i^0, l_i^1)$. Then, we compute ot_2 as $\mathsf{OT}_2(\mathsf{ot}_2, \{\vec{k_i}\}_{i\in[n]})$. Note that in H_4, ot_2 was computed as $\mathsf{OT}_2(\mathsf{ot}_1, \{\vec{l_i}\}_{i\in[n]})$.

Claim 9 *Assuming OT is secure against malicious senders, the output of H_4 and H_5 are indistinguishable.*

[13] OT_1 not only binds the Sender's choice bit but also the randomness used by the Sender. This is true for the two-round OT by Naor-Pinkas [39]

Hybrid $H_6(\mathsf{st}_{F_3})$*:* This hybrid is identical to H_5, except that the garbled circuit used as the secret for CDS statement is simulated. In particular, whenever $x'_2 = x''_2$, we query the ideal functionality on x'_2 to get back y and then compute $(\hat{C}, (k_1, \ldots, k_n))$ using the simulator for garbled circuit Sim_{gc} on input y.

Claim 10 *Assuming the security of* $(\mathsf{Garble}, \mathsf{Eval})$*, the output of* H_5 *and* H_6 *are indistinguishable.*

Finally, to conclude the proof, we observe that H_6 is identical to the ideal world where A interacts with Sim_2.

Case 2: P_1 is corrupted. Let A be the adversary that corrupts P_1. We describe a F_2-time-traveling simulator Sim_1 next. Let st_1 and st_2 be the state of P_1 and P_2 respectively. Sim_2 is given $(\mathsf{st}_1, \mathsf{st}_2)$ as inputs and st_{F_2} as auxiliary input where $\mathsf{isvalidchain}(\mathsf{st}_2, \mathsf{st}_{F_2}) = 1$ and $|\mathsf{st}_{F_2}| - |\mathsf{st}_1| \geq F_2$.

Simulator Sim_1*:* It runs the knowledge-extractor for $\langle P, V \rangle$ to extract the witness from P_1 while computing the rest of messages sent by P_2 w.r.t. a fixed input, that is, $x_2 = 0$. In particular, it samples a uniformly random PRF key K and commits to it inside com_2, computes ot_2 w.r.t. input $x_2 = 0$ and uniform randomness, samples a uniformly random string c, computes cds_1 with witness $\perp$.

Lemma 5. *The joint view of* P_2 *and* A *in the real world is computationally indistinguishable from view of* Sim_1 *and* A *in the ideal world, even to distinguishers having access to* $\mathcal{G}_{\mathsf{ledger}}$.

Proof. The proof follows via a sequence of hybrids.

Hybrid H_0*:* This hybrid corresponds to the real world. The output of this hybrid is the joint view of P_2 and A along with the output of P_2.

Hybrid $H_1(\mathsf{st}_{F_2})$*:* This hybrid is identical to H_0 except that we run the F_2-time-traveling knowledge extractor of $\langle P, V \rangle$ using st_{F_2} as advice. Note that the knowledge extractor is straightline extractor. Let w' be the witness output by P_1, further, let x'_1 be the input of P_1 implicit in w'. P_2 computes its output as $f(x'_1, x_2)$.

Claim 11 *Assuming the correctness of* F_2*-knowledge-extractor of* $\langle P, V \rangle$*, the outputs of* H_0 *and* H_1 *are computationally indistinguishable.*

Hybrid $H_2(\mathsf{st}_{F_3})$*:* This hybrid is identical to H_1 except that the first message cds_1 is computed using the witness $\perp$. In particular, note that in H_1, it was computed using the correct witness as described in the protocol.

Claim 12 *Assuming the security of CDS against malicious senders, the output of* H_2 *and* H_1 *are computationally indistinguishable.*

Hybrid $H_3(\mathsf{st}_{F_2})$*:* This hybrid is identical to H_2 except that P_2 commits to an independent PRF key K' inside com_2. In particular, note that in H_2, com_2 was a commitment to a PRF key K which derived the randomness for computing ot_1 and c.

Claim 13 *Assuming the hiding of* $(\mathsf{Com}, \mathsf{Rec})$ *against* (R^*, D^*) *where* R^* *does* t *operations and* D^* *is* F_2*-time-traveling distinguisher, the output of* H_2 *and* H_3 *are computationally indistinguishable.*

Hybrid $H_4(\mathsf{st}_{F_2})$*:* This hybrid is identical to H_3 except that we use uniform randomness to compute ot_1 and c. In particular, note that in H_3 the randomness were sampled using $F(K, 0)$ and $F(K, 1)$.

Claim 14 *Assuming the PRF security of F, the output of H_3 and H_4 are computationally indistingusihable.*

Hybrid $H_5(\mathsf{st}_{F_2})$*:* This hybrid is identical to H_4 except that we compute ot_1 differently. In particular, we compute ot_1 as $\mathsf{OT}_1(0)$ whereas in H_4, ot_1 was computed as $\mathsf{OT}_1(x_2)$.

Claim 15 *Assuming the security of OT against malicious senders, the output of H_4 and H_5 are computationally indistingusihable.*

Finally, the conclude the proof, we observe that H_5 is identical to the ideal world where A interacts with Sim_1.

Acknowledgements. We thank the anonymous reviewers of ITCS 2022 for their helpful comments. The authors were supported by the NSF award 1916939, DARPA SIEVE program, a gift from Ripple, a DoE NETL award, a JP Morgan Faculty Fellowship, a PNC center for financial services innovation award, and a Cylab seed funding award.

References

1. Arash Afshar, Payman Mohassel, Benny Pinkas, and Ben Riva. Non-interactive secure computation based on cut-and-choose. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 387–404. Springer, Heidelberg, May 2014. doi:10.1007/978-3-642-55220-5_22.
2. Prabhanjan Ananth and Abhishek Jain. On secure two-party computation in three rounds. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part I*, volume 10677 of *LNCS*, pages 612–644. Springer, Heidelberg, November 2017. doi:10.1007/978-3-319-70500-2_21.
3. Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and Lukasz Mazurek. Secure multiparty computations on bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 443–458. IEEE Computer Society Press, May 2014. doi:10.1109/SP.2014.35.
4. Christian Badertscher, Peter Gazi, Aggelos Kiayias, Alexander Russell, and Vassilis Zikas. Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 913–930. ACM Press, October 2018. doi:10.1145/3243734.3243848.
5. 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. Springer, Heidelberg, August 2017. doi:10.1007/978-3-319-63688-7_11.
6. Saikrishna Badrinarayanan, Sanjam Garg, Yuval Ishai, Amit Sahai, and Akshay Wadia. Two-message witness indistinguishability and secure computation in the plain model from new assumptions. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part III*, volume 10626 of *LNCS*, pages 275–303. Springer, Heidelberg, December 2017. doi:10.1007/978-3-319-70700-6_10.
7. Saikrishna Badrinarayanan, Vipul Goyal, Abhishek Jain, Dakshita Khurana, and Amit Sahai. Round optimal concurrent MPC via strong simulation. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part I*, volume 10677 of *LNCS*, pages 743–775. Springer, Heidelberg, November 2017. doi:10.1007/978-3-319-70500-2_25.
8. Boaz Barak and Amit Sahai. How to play almost any mental game over the net - concurrent composition via super-polynomial simulation. In *46th FOCS*, pages 543–552. IEEE Computer Society Press, October 2005. doi:10.1109/SFCS.2005.43.
9. 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. Springer, Heidelberg, May 1997. doi:10.1007/3-540-69053-0_20.

10. Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 459–474, 2014. doi:10.1109/SP.2014.36.
11. 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. Springer, Heidelberg, August 2014. doi:10.1007/978-3-662-44381-1_24.
12. Nir Bitansky, Yael Tauman Kalai, and Omer Paneth. Multi-collision resistance: a paradigm for keyless hash functions. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *50th ACM STOC*, pages 671–684. ACM Press, June 2018. doi:10.1145/3188745.3188870.
13. Nir Bitansky, Dakshita Khurana, and Omer Paneth. Weak zero-knowledge beyond the black-box barrier. In Moses Charikar and Edith Cohen, editors, *51st ACM STOC*, pages 1091–1102. ACM Press, June 2019. doi:10.1145/3313276.3316382.
14. Nir Bitansky and Omri Shmueli. Post-quantum zero knowledge in constant rounds. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *52nd ACM STOC*, pages 269–279. ACM Press, June 2020. doi:10.1145/3357713.3384324.
15. G. R. Blakley. Safeguarding cryptographic keys. *Proceedings of AFIPS 1979 National Computer Conference*, 48:313–317, 1979.
16. 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. IEEE Computer Society Press, October 2010. doi:10.1109/FOCS.2010.86.
17. Arka Rai Choudhuri, Vipul Goyal, and Abhishek Jain. Founding secure computation on blockchains. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 351–380. Springer, Heidelberg, May 2019. doi:10.1007/978-3-030-17656-3_13.
18. 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 2017*, pages 719–728. ACM Press, October / November 2017. doi:10.1145/3133956.3134092.
19. Phil Daian, Rafael Pass, and Elaine Shi. Snow white: Robustly reconfigurable consensus and applications to provably secure proof of stake. In Ian Goldberg and Tyler Moore, editors, *Financial Cryptography and Data Security*, pages 23–41, Cham, 2019. Springer International Publishing.
20. Cynthia Dwork and Moni Naor. Zaps and their applications. In *41st FOCS*, pages 283–293. IEEE Computer Society Press, November 2000. doi:10.1109/SFCS.2000.892117.
21. U. Feige, D. Lapidot, and A. Shamir. Multiple noninteractive zero knowledge proofs under general assumptions. *SIAM J. Comput.*, 29:1–28, 1999.
22. Uriel Feige and Adi Shamir. Witness indistinguishable and witness hiding protocols. In *22nd ACM STOC*, pages 416–426. ACM Press, May 1990. doi:10.1145/100216.100272.
23. 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. Springer, Heidelberg, April 2015. doi:10.1007/978-3-662-46803-6_10.
24. 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. Springer, Heidelberg, August 2017. doi:10.1007/978-3-319-63688-7_10.
25. Juan A. Garay, Aggelos Kiayias, Nikos Leonardos, and Giorgos Panagiotakos. Bootstrapping the blockchain, with applications to consensus and fast PKI setup. In Michel Abdalla and Ricardo Dahab, editors, *PKC 2018, Part II*, volume 10770 of *LNCS*, pages 465–495. Springer, Heidelberg, March 2018. doi:10.1007/978-3-319-76581-5_16.
26. Yael Gertner, Yuval Ishai, Eyal Kushilevitz, and Tal Malkin. Protecting data privacy in private information retrieval schemes. In *30th ACM STOC*, pages 151–160. ACM Press, May 1998. doi:10.1145/276698.276723.
27. Oded Goldreich and Hugo Krawczyk. On the composition of zero-knowledge proof systems. *SIAM J. Comput.*, 25(1):169–192, February 1996. doi:10.1137/S0097539791220688.
28. Oded Goldreich, Silvio Micali, and Avi Wigderson. How to prove all NP-statements in zero-knowledge, and a methodology of cryptographic protocol design. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 171–185. Springer, Heidelberg, August 1987. doi:10.1007/3-540-47721-7_11.
29. Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989.
30. 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. Springer, Heidelberg, November 2017. doi:10.1007/978-3-319-70500-2_18.

31. Vipul Goyal and Silas Richelson. Non-malleable commitments using goldreich-levin list decoding. Cryptology ePrint Archive, Report 2019/1195, 2019. https://eprint.iacr.org/2019/1195.
32. Omer Horvitz and Jonathan Katz. Universally-composable two-party computation in two rounds. In Alfred Menezes, editor, *CRYPTO 2007*, volume 4622 of *LNCS*, pages 111–129. Springer, Heidelberg, August 2007. doi:10.1007/978-3-540-74143-5_7.
33. Jonathan Katz and Rafail Ostrovsky. Round-optimal secure two-party computation. In Matthew Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 335–354. Springer, Heidelberg, August 2004. doi:10.1007/978-3-540-28628-8_21.
34. Aggelos Kiayias and Giorgos Panagiotakos. Speed-security tradeoffs in blockchain protocols. Cryptology ePrint Archive, Report 2015/1019, 2015. https://eprint.iacr.org/2015/1019.
35. 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. Springer, Heidelberg, May 2016. doi:10.1007/978-3-662-49896-5_25.
36. Huijia Lin and Rafael Pass. Constant-round non-malleable commitments from any one-way function. In Lance Fortnow and Salil P. Vadhan, editors, *43rd ACM STOC*, pages 705–714. ACM Press, June 2011. doi:10.1145/1993636.1993730.
37. Jia Liu, Tibor Jager, Saqib A. Kakvi, and Bogdan Warinschi. How to build time-lock encryption. 86(11), 2018. doi:10.1007/s10623-018-0461-x.
38. Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Decentralized Business Review*, page 21260, 2008.
39. Moni Naor and Benny Pinkas. Oblivious transfer and polynomial evaluation. In *31st ACM STOC*, pages 245–254. ACM Press, May 1999. doi:10.1145/301250.301312.
40. R. Pass and E. Shi. Hybrid consensus: Efficient consensus in the permissionless model. In *DISC*, 2017.
41. Rafael Pass. Simulation in quasi-polynomial time, and its application to protocol composition. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 160–176. Springer, Heidelberg, May 2003. doi:10.1007/3-540-39200-9_10.
42. 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. Springer, Heidelberg, April / May 2017. doi:10.1007/978-3-319-56614-6_22.
43. Rafael Pass and Elaine Shi. FruitChains: A fair blockchain. In Elad Michael Schiller and Alexander A. Schwarzmann, editors, *36th ACM PODC*, pages 315–324. ACM, July 2017. doi:10.1145/3087801.3087809.
44. Rafael Pass and Elaine Shi. The sleepy model of consensus. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part II*, volume 10625 of *LNCS*, pages 380–409. Springer, Heidelberg, December 2017. doi:10.1007/978-3-319-70697-9_14.
45. R. L. Rivest, A. Shamir, and D. A. Wagner. Time-lock puzzles and timed-release crypto. Technical report, USA, 1996.
46. Adi Shamir. How to share a secret. *Communications of the Association for Computing Machinery*, 22(11):612–613, November 1979.
47. Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. 2014.

A Standard Definitions

The following definitions are all in the plain model.

A.1 Argument Systems

Witness indistinguishability was introduced by Feige and Shamir [22] as a natural weakening of zero-knowledge. Informally, an argument system is witness indistinguishable if no adversarial verifier can tell the difference between the prover using a witness w_1 and using another witness w_2.

Definition 14 (Witness Indistinguishability). *A proof/argument system (P, V) for NP language L with relation R_L is witness indistinguishable if for all PPT adversaries V^*, all auxiliary inputs z, all statements $x \in L$, and all witnesses w_0, w_1 such that $R_L(x, w_0) = R_L(x, w_1) = 1$, the following holds:*

$$\{\mathsf{view}_{V^*}(P(1^\lambda, x, w_0), V_T^*(1^\lambda, x, z))\} \approx_c \{\mathsf{view}_{V^*}(P(1^\lambda, x, w_0), V_T^*(1^\lambda, x))\}$$

An argument of knowledge intuitively allows a prover to show they "know" a witness for the statement being proved. This is formalized by the existence of an extractor which extracts the witness from the prover. We recall the definition described by [36].

Definition 15 (Arguments of Knowledge). *A proof/argument system (P, V) for NP language L with relation R_L is a proof/argument of knowledge if there exists a PPT extractor Ext such that for all adversarial provers P^*, all auxiliary inputs z, and all statements $x \in \{0,1\}^\lambda$, the following holds:*

$$\Pr[\langle P^*(z), V\rangle(1^\lambda, x) = 1] \leq \Pr[\mathsf{Ext}^{P^*(1^\lambda, x, z)(x)} \in R_L(x)] + \mathsf{negl}(\lambda)$$

A.2 Conditional Disclosure Of Secrets

A Conditional Disclosure of Secrets protocol [26] allows a sender S to reveal a secret s to a receiver R conditioned on R having a witness w for a shared NP statement x. If x is not in the language, then R should receive no information about s. Furthermore, in any case S should receive no information about R's input to the protocol.

Definition 16 (Conditional Disclosure Of Secrets). *Conditional Disclosure of Secrets protocol is associated with an NP language L with relation R_L. Both S and R hold the same instance x. Additionally, S holds a secret s and R holds a string $w \in \{0,1\}^*$. At the end of the protocol, R outputs s', which we denote by $s' \leftarrow \langle S(1^\lambda, x, s), R(1^\lambda, x, w)\rangle$. We require the following properties:*

Correctness: *If $(x, w) \in R_L$, then it holds with probability 1 that $s \leftarrow \langle S(1^\lambda, x, s), R(1^\lambda, x, w)\rangle$.*

Soundness: *If $x \notin L$, then for any adversarial receiver R^*, any $s_0, s_1 \in \{0,1\}^\lambda$, and for any auxiliary input z, it holds that*

$$\{\mathsf{view}_{R^*}(S(1^\lambda, x, s_0), R^*(1^\lambda, x, w, z))\} \approx_c \{\mathsf{view}_{R^*}(S(1^\lambda, x, s_1), R^*(1^\lambda, x, w, z))\}$$

Receiver Privacy: *For any adversarial sender S^*, any strings w_0, w_1, and for any auxiliary input z, it holds that*

$$\{\mathsf{view}_{S^*}(S^*(1^\lambda, x, z), R(1^\lambda, x, w_0))\} \approx_c \{\mathsf{view}_{S^*}(S^*(1^\lambda, x, z), R(1^\lambda, x, w_1))\}$$

For a two-round CDS protocol, the receiver sends the first message $\mathsf{cds}_1 \leftarrow \mathsf{CDS}(1^\lambda, x, w)$ and the sender sends the second message $\mathsf{cds}_2 \leftarrow \mathsf{CDS}(1^\lambda, x, \mathsf{cds}_1, s)$.

A.3 Public Key Integrated Encryption-Signature Scheme

A public key integrated encryption-signature scheme works as both an IND-CPA secure public key encryption scheme and an unforgeable public key signature scheme. We recall the definition from [30].

Definition 17 (Public Key Integrated Encryption-Signature Scheme). *A public key integrated encryption-signature scheme for encryption message space $\mathcal{M}_1$ and signing message space $\mathcal{M}_2$ consists of the following polynomial time algorithms:*

$\mathsf{Setup}(1^\lambda)$*: The setup algorithm takes as input the security parameter λ. An outputs a master public-secret key pair $(\mathsf{mpk}, \mathsf{msk})$.*

$\mathsf{Enc}(\mathsf{mpk}, m)$: *The encryption algorithm takes as input master public key* mpk *and a message* m*, and outputs a ciphertext* c*.*

$\mathsf{Dec}(\mathsf{msk}, c)$: *The decryption algorithm takes as input master secret key* msk *and a ciphertext* c*, and outputs a message* m*.*

$\mathsf{Sign}(\mathsf{msk}, m)$: *The signing algorithm takes as input master secret key* msk *and a message* m*, and outputs a signature* σ*.*

$\mathsf{Vf}(\mathsf{mpk}, m, \sigma)$: *The verification algorithm takes as input master public key* mpk*, a message* m*, and a signature* σ*, and outputs a bit.*

It must also satisfy the following properties:

Correctness: *For all* $\lambda, m_1 \in \mathcal{M}_1, m_2 \in \mathcal{M}_2$ *and* $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda)$ *we have that* $\mathsf{Dec}(\mathsf{msk}, \mathsf{Enc}(\mathsf{mpk}, m_1)) = m_1$ *and* $\mathsf{Vf}(\mathsf{mpk}, m_{2,\mathsf{Sign}(\mathsf{msk}, m_2)} = 1)$*.*

Integrated Security: *For every PPT adversary* $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2)$*, for all* $\lambda \in \mathbb{N}$ *the following holds:*

$$\left| \Pr\left[\mathcal{A}_1(c, \mathsf{st}) = b \middle| \begin{array}{c} (\mathsf{mpk}, \mathsf{msk}) \leftarrow\$\, \mathsf{Setup}(1^\lambda); b \leftarrow\$\, \{0,1\} \\ (m_0, m_1, \mathsf{st}) \leftarrow \mathcal{A}_0^{\mathsf{Sign}(\mathsf{msk},\cdot)}(\mathsf{mpk}); c \leftarrow \mathsf{Enc}(\mathsf{mpk}, m_b) \end{array} \right] - \frac{1}{2} \right| \le \mathsf{negl}(\lambda)$$

and

$$\Pr\left[\mathsf{Vf}(\mathsf{msk}, m^*, \sigma^*) = 1 \middle| \begin{array}{c} (\mathsf{mpk}, \mathsf{msk}) \leftarrow\$\, \mathsf{Setup}(1^\lambda) \\ (m^*, \sigma^*) \leftarrow \mathcal{A}_2^{\mathsf{Sign}(\mathsf{msk},\cdot)}(\mathsf{mpk}) \end{array} \right] \le \mathsf{negl}(\lambda)$$

where $\mathcal{A}_2$ *must never have queries* m^* *to the signing oracle.*

A.4 Threshold Secret Sharing

Threshold secret sharing [15,46] allows the secret to be split into a number of shares such that it can be reconstructed using a number of shares at least as big as the threshold, but any collection containing less shares than the threshold reveals nothing about the secret.

Definition 18 (Threshold Secret Sharing). *Threshold secret sharing scheme consists of a pair of probabilistic algorithms* $(\mathsf{Share}, \mathsf{Recon})$ *with the following syntax.*

$\mathsf{Share}(s, n, k)$: *On input a secret* s*, an integer* s*, and a threshold* k*, the sharing algorithm outputs* n *shares* $(\mathsf{sh}_i)_{i=1,\dots,n}$*.*

$\mathsf{Recon}(I, \{\mathsf{sh}_i\}_{i\in I})$: *On input a set* I *and the corresponding shares, the reconstruction algorithm outputs the secret* s' *if* $|I| \ge k$ *and* $\perp$ *otherwise.*

We require the following properties:

Correctness: *For all secrets* s *and sets* $I \subseteq [n]$ *such that* $|I| \ge k$*, it holds that* $\mathsf{Recon}(I, \{\mathsf{sh}_i\}_{i\in I} | (\mathsf{sh}_i)_{i=1,\dots,n} \leftarrow \mathsf{Share}(s, n, k)) = s$*.*

Secrecy: *For all secrets* s_0, s_1 *and sets* $I \subseteq [n]$ *such that* $|I| < k$*, it holds that*

$$\{\{\mathsf{sh}_i\}_{i\in I} : (\mathsf{sh}_i)_{i=1,\dots,n} \leftarrow \mathsf{Share}(s_0, n, k)\} \approx_s \{\{\mathsf{sh}_i\}_{i\in I} : (\mathsf{sh}_i)_{i=1,\dots,n} \leftarrow \mathsf{Share}(s_1, n, k)\}$$

A.5 Semi-Malicious Secure Two-Party Computation

A semi-malicious adversary follows the protocol, but may choose their input and randomness arbitrarily at any time during the execution. After each message, they output the input and randomness which result in that message to a side tape.

Semi-malicious secure two-party computation is defined using the real/ideal world paradigm.

Ideal World The ideal world contains P_1, P_2, and a trusted third party. At most one of P_1 and P_2 are controlled by the adversary. The ideal world execution proceeds as follows:

1. **Input Distribution:** P_1 and P_2 receive their respective inputs x_1 and x_2 from the environment.
2. **Inputs to Trusted Third Party:** P_1 and P_2 send their inputs to the trusted third party. An honest party always sends the input they received from the environment. The corrupted party may send any input of their choice.
3. **Trusted Third Party Answers P_2:** The trusted third party sends $f(x'_1, x'_2)$ to P_2, where x'_1 and x'_2 are the inputs it received in step 2.
4. **Output:** If P_2 is honest, it outputs $f(x'_1, x'_2)$ as received from the trusted third party. If P_1 is honest, it outputs $\perp$. The adversarial party always outputs its entire view.

We define $\mathsf{Ideal}_{f,A}(x_1, x_2)$ to be the joint distribution over the outputs of the adversary A and the honest party according to the above ideal execution.

Real World Let Π be a two-party protocol computing f. In the real process, both parties execute the protocol Π. As in the ideal process, they receive inputs from the environment. At most one of P_1 and P_2 are controlled by the adversary. After each message from the adversary, it outputs the input and randomness which result in that message to a side tape. The honest party outputs according to the Π specification, while the adversary outputs its entire view.

We define $\mathsf{Real}_{\Pi,A}(x_1, x_2)$ to be the joint distribution over the outputs of the adversary A and the honest party according to the above real execution.

Definition 19 (Semi-Malicious Secure Two-Party Computation). *Let Π be a two party protocol computing the two party functionality f. We say Π is a semi-malicious two-party computation protocol for f if there is a stateful PPT algorithm* Sim *such that for all semi-malicious A, all auxiliary inputs z, and all inputs x_1, x_2 the following holds:*

$$\{\mathsf{Real}_{\Pi,A}(x_1, x_2)\} \approx_c \{\mathsf{Ideal}_{f,\mathsf{Sim}}(x_1, x_2)\}$$