Composing Timed Cryptographic Protocols: Foundations and Applications

Karim Eldefrawy *, Ben Terner * [†], and Moti Yung [‡]

$*$ SRI International, $\star$ Confidencial, ‡ Google and Columbia University

Abstract. Time-lock puzzles are unique cryptographic primitives that use computational complexity to keep information secret for some period of time, after which security expires. Unfortunately, current analysis techniques of time-lock primitives provide no sound mechanism to build multi-party cryptographic protocols which use expiring security as a building block. We explain in this paper that all other attempts at this subtle problem lack either composability, a fully consistent analysis, or functionality. The subtle flaws in the existing frameworks reduce to an impossibility by Mahmoody et al., who showed that time-lock puzzles with super-polynomial gaps (between committer and solver) cannot be constructed from random oracles alone; yet still the analyses of algebraic puzzles today treat the solving process as if each step is a generic or random oracle.

This paper presents a new complexity theoretic based framework and new structural theorems to analyze timed primitives with full generality and in composition (which is the central modular protocol design tool). The framework includes a model of security based on fine-grained circuit complexity which we call residual complexity, which accounts for possible leakage on timed primitives as they expire. Our definitions for multi-party computation protocols generalize the literature standards by accounting for fine-grained polynomial circuit depth to model computational hardness which expires in feasible time. Our composition theorems incur degradation of (fine-grained) security as items are composed. In our framework, simulators are given a polynomial "budget" for how much time they spend, and in composition these polynomials interact.

Finally, we demonstrate via a prototypical auction application how to apply our framework and theorems. For the first time, we show that it is possible to prove – in a way that is fully consistent, with falsifiable assumptions – properties of multi-party applications based on leaky, temporarily secure components.

1 Introduction

Time-lock cryptography has been studied since the seminal work of Rivest, Shamir, and Wagner (RSW) [35] more than twenty-five years ago with the purpose of modelling security which expires over time. More recently, inherently sequential functions motivated by large-scale consensus, distributed ledgers, and

[†] Work performed partially while at SRI International.

blockchain applications have also precipitated considerable research in verifiable delay functions [10, 34, 39]. The interest in time-lock primitives has yielded various notions like non-malleable time-lock puzzles [22], non-malleable timed commitments [26], and UC-security [5, 6] of time-lock puzzles in the random oracle model.

Time-lock primitives facilitate distributed applications in which one or more parties promise that a value will be revealed at a later time, without requiring such parties to be honestly participating, or even be online, at the time of reveal. This paradigm enables new techniques and applications for distributed computations. For example, over twenty years ago, Boneh and Naor [11] introduced timed commitments as a way to achieve fairness in secure multi-party computation (MPC). Recently, Wan et al. [38] used time-lock puzzles to construct more efficient broadcast with adaptive security.

1.1 New Foundations for Timed-Release Cryptography

Our motivating concrete, yet prototypical, application for this work is developing a provably secure simultaneous multi-round auction (SMRA) [32] without idealized assumptions in its security proof, i.e., avoiding solely relying on random oracle based analysis. An SMRA proceeds in rounds, in which each round multiple parties may bid on multiple items; at the end of each round, all bids are revealed. We wish to implement such an SMRA via time-lock primitives, e.g., when imposing no requirements on committers after submitting their commitment. We further wish to treat the solution in a unified consistent manner, in the sense that all desired protocol properties should rely on a consistent (rather than self-contradicting) set of cryptographic assumptions.

However, we found that the existing literature on time-lock primitives contains two major deficiencies which impede soundly designing and analyzing a secure protocol without inadvertently admitting subtle inconsistencies in the developed security analysis and proofs.

1st Deficiency: Composition for Leaky Protocols. When composing cryptographic protocols that use time-lock primitives, it is important to model how to use timed primitives *as part of a protocol (i.e., as timed sub-protocols)*, and more generally protocols with timed security. This includes modeling exactly what happens when the time-lock expires. For example, all definitions of timed primitives allow for some advantage of a parallel adversary to learn the puzzle solution before an honest parties, and this is made explicit in the literature on delay functions [9, 10]. During composition, the adversary can use this extra time advantage to attack other components of the composed protocol, i.e., its timed sub-protocols. Although the recent result of Cohen et al.[16] provides a preliminary analysis of the security degradation of composed resource-restricted primitives and is a good step forward, we seek a more general and precise analysis and framework.

2nd Deficiency: Current Inconsistent Analysis. Existing models idealize the security analysis of time-lock puzzles in a way that presents contradictions with established results [29]. As we explain in Section 2.1, the generation of a puzzle is analyzed in an algebraic model (enabling efficient generation of puzzles based on standard hardness assumptions such as in [35]), while the security of the puzzle solving process is analyzed as if a random oracle/generic models. Such an analyzed solving process (relying on random oracles only, whether explicitly or implicitly) cannot achieve the super-polynomial gap between generation time and solving time due to an impossibility by Mahmoody et al.[29]. The "gap" is the difference between the time to generate a time-lock puzzle and the time required to solve it. Current works all incur an exclusive or (XOR) of two basic properties: the "super-poly gap property" and "consistent analysis property." This situation is tolerated in a preliminary stage of presenting a primitive, but it cannot persist forever. We frame our approach with respect to existing idealized models in Table 1. Moreover, we note that the existing idealized analyses of composable time-lock primitives are all in non-falsifiable models [33].

We argue that this state of (idealized and) inconsistent modeling and analysis should be considered an initial and temporary state of investigations, leading to a more refined understanding of realizable constructions and consistent security analysis. Given the contradictions we discuss above, such idealized inconsistent modeling and security analyses should not be regarded as the final word on actual realizable timed cryptographic protocols and constructions. This has been the tradition in designing primitives in cryptography throughout the lifetime of modern cryptography; the profound difficulties posed by timed primitives should not make us shy away from the solid and tested tradition which is at the core of the field, and this is an important starting point of our investigation.

New Foundations. The above two deficiencies which have been overlooked (following the tradition of tackling a hard problem initially with an idealization) lead us to develop an initial new foundation of time-lock primitives in a falsifiable model which is radically different from prior work. Our approach leads to a fine-grained polynomial model of security, in which the adversary learns the solution based on the (assumed and falsifiable) hardness of the underlying computational problem, and can learn the solution before the honest parties following the honest solution algorithm. In our framework, the analysis of the solve algorithm *does not* implicitly treat the iterative process as a sequence of random oracles, each state revealing nothing about its future. Rather, since the secrecy will expire, we model a process which is leaky. Specifically, each state may computationally reveal something about its nearby future states (which, towards the end of the iterative solution reveals something about the committed secret).

New Techniques for MPC. As a product of the framework, we provide new, generalized definitions of multi-party computation, and introduce techniques to enable temporarily-private applications not possible with previous definitions. Our composition theorems, in turn, capture degradation of security when composing timed primitives. This degradation has non-black-box consequences to the design of our auction application.

Protocol	Analytical Model Generate	 Solve	 Super-poly Gap	 Consistent
Arapinis et al. [1]	Idealized (RO)	Idealized (RO)	NO	YES
Baum et al. '20b [5]	Algebraic	Idealized (RO)	YES	NO
Baum et al. '20a [6]	Algebraic	Idealized (RO)	YES	NO
Katz et al. [26]	Algebraic	Idealized (SAGM)	YES	NO
Chvojka, Jager [14]	Algebraic	Idealized (†)	YES	NO
This Work	Algebraic	Complexity Theoretic	YES	YES

Table 1: Properties of Analytical Models for Composable Timed Primitives. "Generate" and "Solve" describe the analytical framework used for the respective TLP algorithms, whether they are in algebraic or idealized models. RO stands for "Random Oracle" and SAGM stands for "Strong Algebraic Group Model." "Superpoly Gap" describes whether the analytical framework admits a superpolynomial gap between generation and solving. The "Consistent" column describes whether the analyses of generation and solving are consistent. (†) Depends on the Strong Sequential Squaring Assumption, but is effectively treated as a Random Oracle (RO) in the security proof.

Our new proof techniques assign a "budget" to the simulator that allows strictly less work than it takes to solve a puzzle. Otherwise, the security reduction for a protocol argues that an adversary can do no worse than a simulator that can explicitly solve a puzzle, losing privacy. When composing protocols π and ρ, the composition is secure only against the depth of π's adversary less the depth of ρ's simulator; this is due to the fact π's adversary can run ρ's simulator in any attack against π. We expound on these techniques and more in Section 3.

A Full Protocol. Only after fully developing the above model and proving the foundational composition theorems are we able to finally present a protocol for an auction. The auction protocol is built by the concurrent composition of many leaky cryptographic primitives. A full analysis therefore requires considering security degradation that occurs in two places: in the primitives themselves and from their composition. In the complexity theoretic security model, timed primitives incur leakage (computationally derived from a state to its future neighbors as explained above) which must be factored in the security analysis. The composition theorems reveal additional security degradation and constraints on the simulator. In order to design a full protocol in a consistent framework, both of these forms of degradation and the corresponding simulation techniques were necessary components of the analysis.

We hope this work and its principles will motivate further research in the direction advocated herein, further adopting timed primitives into the accepted formal tradition of modern cryptography (rather then continuing tackling difficulties of the primitive's instances by essentially accepting the anomaly of claiming security in one model and analyzing performance of the same instance in a model contrary to the former).

1.2 Related Work

We briefly highlight related works in time-delayed and fine-grained cryptography. A more comprehensive discussion of related work is deferred to Appendix D.

Time-lock Puzzles and Composition. The seminal work on time-lock puzzles was produced by Rivest, Shamir, and Wagner (RSW) [35]. Boneh and Naor [11] introduced timed-commitments, which progressed the study of timed primitives for fairness in MPC. Bitansky et al.[8] formally defined time-lock puzzles and constructed them using randomized encodings, and construct weak time-lock puzzles from one-way functions. Baum et al.[5, 6] formalized time-lock puzzles in the UC model [12]. Freitag et al.[22] built publicly verifiable, non-malleable time-lock puzzles, but do not compose with general MPC. Katz et al.[26] constructed non-interactive non-malleable timed-commitments with a proof of forced opening but also do not compose with MPC. They also showed that in a quantitative group model, speeding up squaring is as hard as factoring. For negative results, Mahmoody et al.[29] proved there are no time-lock puzzles depending only on random oracles with more than polynomial time gap. Arapinis et al. [1] constructed UC secure time-lock puzzles in the random oracle model, but they depend only on random oracles and achieve only polynomial gap.

Two notable additional works have addressed the assumptions underlying the repeated squaring problem in idealized models. Rotem and Segev [36] showed that speeding up repeated squaring in a generic ring is equivalent to factoring. van Baarsen and Stevens [2] address multiple hardness assumptions used for timed primitives in generic Abelian hidden-order groups.

Sequential and Delay Functions. There is a growing literature on sequential functions [15] and verifiable delay functions (VDF) [9, 10, 19, 30, 34, 39] that motivate the time difference between the best parallel adversary's solution algorithm and the honest sequential algorithm. Both of [19, 30] showed impossibility results of constructions based on random oracles for *tight VDFs*, which require that a sequential function be evaluatable by a parallel adversary in time no less than $T - T^{\rho}$ for some constant $0 < \rho < 1$. By this definition, T includes both the time to solve *and* to construct a proof; still the point remains that the impossibilities separate the time between honest strategy and parallel adversary.

Resource-Restricted Simulation. Independently and concurrently to our work, Cohen, Garay, and Zikas [16] introduce a composition theorem for the resource-restricted setting which is similar to ours, but less general. Their theorem states that if π is secure against a T-depth bounded adversary in the F-hybrid model and ρ is a secure protocol for F against an αT-depth adversary then the sequential composition of π and ρ is secure against a $(1 - \alpha)T$-depth bounded adversary. They also claim that the simulator for the composed protocol works in the sum of the simulators for the respective protocols. Like [22], they also consider an environment that is (arbitrary) polynomial.

The above theorem is less granular than ours, as we pay close attention to the way that the polynomial sizes of the adversaries interact with the simulators for composed protocols. We also expand the composition result to concurrent settings, and we consider a fine-grained polynomial bounded environment.

1.3 Paper Outline

Since this paper introduces a new model, it contains a longer than usual motivation, discussion of subtleties, definitions, and involved relevant formal issues. Section 2 motivates and introduces our new falsifiable computational model for timed cryptography. Section 3 contains our core contributions, including definitions for timed MPC, composition theorems, and an outline for proving security of our application. In Section 4 we formally present our model of depth-secure computation and introduce residual complexity. In Section 5 we model depth-secure MPC. In Section 6 we formally present our composition theorems. Section 7 explains how to transform a leaky algebraic puzzle into a time-lock puzzle using a single random oracle call, which achieves analysis consistent with the model. Section 8 builds an example application and shows how to apply our composition theorems. Our model is expanded in Appendix A. Our sequential composition theorem's proof is in Appendix B. In Appendix C we relate time-lock puzzles to residual complexity by proving that the residual hardness of time-lock puzzles remains high until the time-lock expires. Appendix D expands the related work in time-lock primitives and granular computational models. Finally, Appendix E contains the full proof of our single-shot auction protocol, which additionally includes definitions of non-malleability.

2 A New Model for Timed Primitives

We first discuss subtleties and inconsistencies in (explicit and implicit/hidden) random oracle/ generic domain based security analysis of timed-primitives. We then propose a complexity theoretic based model for timed primitives with a focus on their composition. Central to the new model is a notion of fine-grained polynomial hardness in which some problems are solvable while related underlying problems remain hard.

2.1 Subtleties and Inconsistencies in Random Oracle Based Analysis for Time-Lock Puzzles

Similar to the classical result of [13] in the random oracle (RO) model and [17, 24] about the Fiat-Shamir transform [21], we argue that the desired properties of realizable time-lock puzzles do not follow from the current analyses.

All current computational puzzles follow the following blueprint:

1. Puzzle **generation** uses a trapdoor to efficiently sample a puzzle.
2. Puzzle **solving** uses a sequential algorithm. The puzzle is parameterized such that the (polynomial) sequential algorithm is faster than any known (super-polynomial) method to recover the trapdoor.

For a puzzle to be useful, it must be *efficient to generate* and *hard to solve*. The trapdoor is therefore required for *utility*, while the hardness of the computational problem is required for time-lock *security*.

However, analysis of these algorithms occurs in *inconsistent models*. For all existing constructions:

- The **generation** algorithm is analyzed **in an algebraic model**.
- The **solution** algorithm is analyzed **as if each step is a random oracle**.

To elucidate the inconsistency, we quote directly from the approach of [5]. In the excerpt, they describe the leakage provided by their time-delay functionality:

> The intermediate states leaked to the adversary by the functionality are not concrete representations of actual intermediate states but generic random labels assigned to states (similarly to the generic group model treatment given to the RSW assumption in [6]). Learning these intermediate generic states does not allow the environment (or the adversary) any advantage in computing the next states before they are revealed by the functionality, as these states are sampled uniformly at random.

In both of these works [5, 6] – as well as any analyses using either the RO or the strong algebraic group model – the proofs explicitly assume that no information is revealed to the adversary at each step of the solving process, and that each intermediate state is sampled uniformly at random. *The analysis therefore turns the algebraic structure into a random oracle with properties identical to the impossibility by Mahmoody et al [29].* It is inconsistent to analyze puzzle generation in an algebraic model – where super-polynomial gaps are believed to be achieved – and solution in a random oracle model – which can only achieve polynomial gaps.

The Strong Algebraic Group Model. In the strong algebraic group model of [2, 26] and the generic ring model of [36], each element is expressed as a function of factors or as an inverse of another element, which gives algebraic structure to the elements that have been seen, but leaks no more about the solution than a random oracle. It therefore incurs the same problems as the previous analysis. As discussed, this is also how [5, 6] analyze their algebraic functionalities.

Other Approaches with Similar Subtleties. Other works do not explicitly model the solving process via a random oracle, but either the modeling implies a random oracle or it overlooks leakage as the puzzle is partially solved. For example, the base time-lock puzzle in the construction of Freitag et al.[22] defer to analysis by Pietrzak [34] that assumes the hardness of repeated squaring. Chvojka and Jager [14] similarly reduce to the Strong Sequential Squaring assumption. These formalizations simply assume it is infeasible to guess the solution of a repeated squaring until the final squaring; either it implicitly treats the process as if the probability of guessing the solution before the end is negligible, or it uses a game-based definition that implies the solution process is essentially a random oracle until the very end. Therefore, these techniques as well are not differentiated in any meaningful way from the analysis of Mahmoody et al. [29]. While idealization is reasonable when analysis involves a hard modeling problem, this cannot be the final step. To establish a sound foundation for the solving of time-lock puzzles, we require *explicit* techniques.

2.2 A Complexity Theoretic Based Framework for Computational Puzzles

There are two approaches to a consistent analysis of time-lock primitives: either entirely in the random oracle model, which can only yield polynomial gaps, or in an algebraic model where intermediate states may leak some information about the final solution. This work is the first to choose the latter. Sequential algebraic problems are treated as if each intermediate state leaks information about nearby intermediate states, until there are no more intermediate states and the algebraic solution begins to be revealed. Claims about the hardness of guessing the final state from a given state are certainly falsifiable by an efficient sampling and guessing algorithm.

As examples of such leakage, consider the following ways that repeated squaring admits leakage. (This is a list for illustration, and should not be considered exhaustive. The examples can also be combined by a savvy adversary.)

- For all intermediate states $k < \sqrt{N}$ because no modular reduction is necessary to compute $k^2 \bmod N$, the next state is leaked in low depth.
- In the non-uniform model where the adversary may run many parallel computations, it is possible to compute forward mapping tables that allow the solver to infer an approximation on the puzzle solution, or look-ahead chains. (Even though the probability of computing a look-ahead chain is small because the adversary can compute only a polynomial number within an exponential space, this technique does provide leakage.)
- For small δ, knowledge of an intermediate state k and its solution $k^2 \bmod N$ leaks partial information about $(k+\delta)^2 \bmod N$.

Fine-Grained Complexity. Recall that in time-lock puzzle specifications ([8, 22] Definition 5) the puzzle solver must be able to recover the secret within time that is polynomial in the puzzle's security parameter. Therefore, the (leaky) iterative solution process occupies a regime of fine-grained polynomial complexity, where (too much) information must not be leaked to an adversary with some polynomial depth d, but all information must be leaked when surpassing a different polynomial depth $d' > d$.

The above guides our work into a model of (timed) cryptography with fine-grained polynomial depth which, as we explain below, brings new challenges in modeling and intricate formal definitions.

Residual Complexity. To formalize the above notion of fine-grained polynomial hardness – in which some problems are solvable while *related underlying problems* remain hard – we introduce our definition of *residual complexity.* Intuitively, residual complexity quantifies the "remaining hardness" of a puzzle that has already been (partially) solved by a parallel adversary of depth d.[1]

Definition 1 (Residual Complexity (Informal)). *A puzzle scheme has residual complexity* (d, r) *if no depth-d adversary can guess the solution of a randomly sampled puzzle with probability more than* r.

The "remaining hardness" of the puzzle after attempting to solve it in d depth is computed by 1-r. Our formalization (Definition 7) is a generalization of a technique in defining the depth-hardness of certain computational problems in [26] and others.

Residual complexity models the entire leakage profile of a puzzle by defining the "leakage" of a puzzle as the decrease in residual complexity of the puzzle between every two levels of depth of the best solving algorithm. We illustrate the difference between "leaky" and idealized residual complexity curves of puzzle schemes in Figure 1. In the figure, the x axis represents time, and the y axis represents the best adversary's probability of guessing the solution. A point (x, y) on the curve represents that the best x-depth adversary guesses the solution with probability y. At the moment of the time-parameter, the puzzle is guaranteed to be solvable with probability 1 by the honest strategy.

The Critical Time. For a sequential function, we quantify the difference in time between when the best (parallel) adversary can guess a puzzle solution (with unacceptable, or non-negligible probability[2]) and the time that the honest parties learn the solution via the scheme's solve algorithm. We name the moment when the adversary learns unacceptable information on the solution the *critical time.*

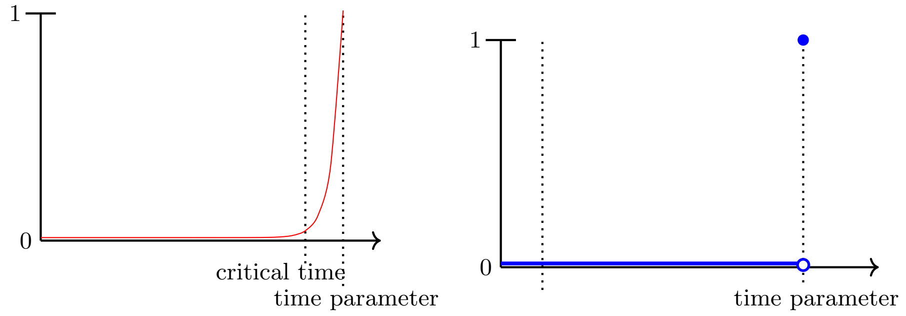
Looking ahead, our simulators will be required to equivocate the solution of the puzzle at the critical time. This is an artifact of the analysis which reflects that puzzles can only be considered secure until the critical time.

2.3 A Random Oracle Compiler For Leaky Puzzles

In Section 7, we present a compiler that applies a random oracle *once* to any algebraic time-lock puzzle. The random oracle is applied only as a last step – as opposed to every intermediate step. This single application of a random oracle serves to "boost" the security of the compiled puzzle until $O(\lambda)$ bits of the algebraic solution have leaked. The "inner" algebraic puzzle is explicitly modeled as a leaky primitive. The analysis is consistent because the security during solving depends explicitly on the leakage of the inner algebraic primitive.

[1] Note that the remaining hardness measures *pseudo-entropy* rather than entropy, as the solution of a timed primitive is always committed at the moment it is generated. (Otherwise the solving algorithm could not be deterministic.)

[2] A negligible function is an asymptotic notion. For each security parameter, the protocol designer can choose a probability that is "unacceptable" for guessing the solution, and designate the depth for which the residual complexity meets this threshold as the critical time. This specifies the moment at which the time-lock is considered to expire.



(a) Example leakage profile for a leaky time-lock puzzle. The residual complexity remains low until almost the hardness expires, which we call the "critical time."

(b) Example leakage profile for an idealized puzzle scheme that perfectly hides its solution until the final step. The residual complexity is a step function.

Fig. 1: Illustration of leakage profiles for a leaky puzzle and an idealized puzzle. The x axis represents time, and the y axis represents the best adversary's probability of guessing the solution. A point (x, y) on the curve represents that the best x-depth adversary guesses the solution with probability y. At the moment of the time-parameter, the puzzle is guaranteed to be solvable with probability 1 by the honest strategy.

3 Technical Overview: Composition and Application

3.1 Simulation Budgets and Depth-Secure MPC

Our treatment of time-based primitives and protocols requires a granular, depth-based definition of secure computation which departs from the standard cryptographic regime of "security up to arbitrary composition within complexity class $\mathbb{P}$," and must account for the exact depths of all involved machines – the adversary, the simulator, and the distinguisher/environment.[3]

Specifically, security should hold with respect to an adversary with depth that is bounded by a fixed polynomial (in comparison to any polynomial in the security parameter). We bound the depth of a distinguisher (or environment) in tandem with the adversary. After surpassing these parameterized depths, it is alright for the information to be revealed.

The simulator must also be restricted to less depth than the puzzle requires to solve. Otherwise, the claim of privacy via reduction is meaningless: an adversary can do no worse *than a simulator that could solve a puzzle outright and learn the solution*. Therefore, the simulator has a granular "depth budget" and it must run in less time than privacy is required to hold. We give the formal definition in Section 5.3 and describe it informally as follows:

[3] To generalize both the works of [22] and [26], our definition states the depth of the environment, but the variable could be either polynomially bounded or unbounded. See [22] for discussion.

Definition 2 (Depth-Secure Multi-Party Computation (Informal)). *A protocol π (d_a, d_s, d_e)-securely implements a functionality F if π's simulator runs in no more than d_s depth, and the distribution of views produced by the simulator is indistinguishable from the distribution of real executions for any d_a-bounded adversary and any d_e-depth bounded distinguisher (the environment).*

Our observation on the simulator's depth budget leads to new questions about whether existing works apply to a fully consistent model such as ours. The simulator in the work by Baum et al. [6] explicitly solves a time-lock puzzle during simulation, and is able to shortcut the solving process only because the simulator is not bound by the global clock functionality. [4] Freitag et al. [22] allow the simulator to explicitly solve puzzles, and artificially constrain the distinguisher by allowing it only to see a function of the solution of modified puzzles, which does not conform with meaningful definitions of a distinguisher *which could run the simulator on its own.* These are very delicate arguments, and while the corresponding constructions are contributions, they fall short of the nuances our fine-grained model brings to light: *since the simulators are of a much different type than usual, nuances regarding the qualitative properties of the proofs using them follow.*

Remark 1 (Comparison to Definitions for Secure Multi-party Computation). Our definition for depth-secure multi-party computation is a strict generalization of the standard simulation-based definitions of MPC [23, 28]. To prove that a protocol is depth-secure, perform the same steps as for a "traditional" MPC proof. In addition, account granularly for the depths of all machines involved.

3.2 Composition of Depth-Secure Protocols

Composing secure timed primitives and protocols introduces additional nuances. For example, consider sequentially composing protocols π and ρ, where π is proven secure in the F-hybrid model against a d_a-depth adversary, and ρ securely implements F against a d'_a adversary. The composition π^ρ is not trivially secure against a $d_a + d'_a$-depth adversary! An adversary against π could use the time during ρ in order to continue attacking π; similarly, an adversary against ρ could use the time *after ρ concludes* and π resumes in order to continue attacking ρ. Similar issues occur in concurrent composition, although they are of the same ilk – in our model, the depth used by an environment to run a "side session" during an attack against π counts towards its depth in the attack.

When composing protocols with timed primitives, the composed simulation must also be shorter than the time that privacy must hold. We also show that the black-box composition is secure only against the *smaller* of the two protocols' distinguishers, and against an adversary that is *smaller* than the first protocol's adversary by the size of the second's simulator.

[4] This observation is more an indictment of bounding time with a global clock functionality than of the simulation technique, since the simulator is not constrained by the functionality and therefore not granularly constrained by depth/time.

Theorem 1 (General Composition (Informal)). *Let π (d_a, d_s, d_e)-securely implement F and let ρ (d'_a, d'_s, d'_e)-securely implement G. The composition of π and ρ is $(d_a - d'_s, d_s + d'_s, \min(d_e, d'_e))$-secure.*

The term $d_a - d'_s$ comes from our simulation technique. Intuitively, if the composition is *not* secure against this depth of adversary, then there exists a d_a-depth adversary that simulates an execution of ρ in parallel to its attack on π and uses the simulation to break π.

The above theorem considers concurrent as well as sequential composition. We additionally prove another relaxed composition theorem for protocols that cannot be proven concurrently composable but may be proven sequentially composable (e.g., if the simulator must be rewound).

Theorem 2 (Sequential Composition (Informal)). *Let π (d_a, d_s, d_e)-securely implement F in the G-Hybrid model and let ρ (d'_a, d'_s, d'_e)-securely implement G. The composition π^ρ $(d_a - d'_s, d_s \cdot d'_s, \min(d_e, d'_e))$-securely implements F.*

The multiplication in the middle term results from considering rewinding.

We present formal versions of these composition theorems for depth-bounded secure computation (Theorems 3 and 4) in Section 6. These analyses are limited to composing depth-secure protocols in a black-box manner, and we do not prove tightness of degradation. There may be better techniques, including those with knowledge of the underlying protocols, that show tighter security preservation under composition.

3.3 Example Application: Simultaneous Multi-Round Auction

The composition theorems allow us to present the first (to our knowledge) timed cryptographic protocol analyzed by composing timed subprotocols. For the application we choose a simultaneous multi-round auction (SMRA) [32]. In a SMRA, many distinct auctions are held simultaneously, and parties may adjust their bids on each item based on the current winners of other items; this allows the auction mechanism to reflect the fact that some bidder may believe that multiple items X and Y are worth more only if they can be bought *together*, and therefore increase its bid for X if it is currently leading the bidding for Y.

Single-Round, Single-Item Auction. We first model a single item, single-round auction in which all parties submit bids via time-lock puzzles. The auction is split into two phases: first a bidding phase concluding with t^{bid} during which parties submit the puzzles containing their bids, and second a solving phase concluding with t^{end} during which they solve the puzzles to extract bids. No puzzles received after t^{bid} are considered in the auction. Importantly, t^{bid} must be set so that the critical time of all submitted puzzles occurs *after* t^{bid}, which implies that the adversary cannot use information about honest parties' bids in order to submit its own.

Additionally, at the end of the round, all parties know all bids, which are also the inputs. Therefore, we must model via *temporary privacy* (Section 5.4) that privacy of all bids must be maintained until the revelation time. In the proof, we withhold the bids from the simulator until t^{bid}, and require the simulator to equivocate after it learns them.

SMRA via Composition. Given a single-round single-item auction protocol, we compose the protocol concurrently with itself in order to achieve a simultaneous single-round auction for multiple items. The security of this simultaneous auction is provided via our composition theorem, but the composition is not black box! Concurrently composed protocols are secure against a smaller adversary, effectively assuming that the adversary can learn information about honest parties' puzzles earlier due to the composition. We therefore must "move back" the assumed critical time for each puzzle, which implies an earlier t^{bid}.

Finally, we trivially compose simultaneous single-round auctions in serial (meaning one begins after the previous concludes, without degrading security) in order to achieve a protocol for a SMRA.

4 Definitions

We denote by $[m]$ the set $\{1, 2, \ldots, m\}$ and $[n_1, n_2]$ the set of all integers between n_1 and n_2. When we write $f = f(\lambda)$, we indicate f is a function of λ. By $\mathsf{poly}(\lambda)$, $\mathsf{polylog}(\lambda)$, and $\mathsf{superpoly}(\lambda)$ we denote any polynomial function, any poly-logarithmic function, and any super-polynomial function of λ, respectively. A function negl is *negligible* if there exists a constant n for which for every polynomial function poly and every $m > n$, $\mathsf{negl}(m) < \frac{1}{\mathsf{poly}(m)}$.

4.1 Interactive Circuits

We adapt a model of computation based on *interactive circuits* [7]. We refer to [7] for the full definition and summarize it here.

An L-round interactive circuit $\mathsf{iC} = \{\mathsf{iC}^\ell\}_{\ell \in [L]}$ with oracle $\mathcal{O}$ is a sequence of L next-step circuits that interacts with $\mathcal{O}$ as follows. In round $r \in [L]$, the next-step circuit iC^r takes as input st^{r-1} and a^{r-1}, where st^{r-1} is the state output by the previous circuit and a^{r-1} is the list of oracle responses. The round-r output is described as $\mathsf{iC}^r(st^{r-1}, a^{r-1}) = (st^r, q^r, o^r)$, where st^r is the state output by the rth circuit, q^r is the set of queries output by the rth circuit, a^r is the list of answers to q^r, and o^r is the output of the rth circuit. The initial inputs st^0 and q^0 are defined to be the 0 bit string, and a^0 is defined to be the circuit's advice string. Or specifically,

$$(st^r, q^r, o^r) = \begin{cases} \mathsf{iC}^r(st^{r-1}, a^{r-1}) & \text{if } \forall k, a_k^{r-1} = \mathcal{O}(q_k^{r-1}) \neq \bot \\ (\bot, \bot, \bot) & \text{otherwise} \end{cases}$$

The transcript is the list of all queries, answers, and outputs $\{q^r, a^r, o^r\}_{r \in [L]}$. The oracle-assisted interface allows interactive circuits to interact concurrently with each other. One can consider two interactive circuits A and B to interact via a configuration in which the queries q_A^r produced by circuit A in round r are the answers a_B^r provided to circuit B in round r, and vice versa.

4.2 Depth-Bounded Computation

Our computational model constrains the length of time that a party may run by constraining the depth of its corresponding circuit. In support of this paradigm, we introduce definitions for circuits that are bounded in both size and depth.

For any circuit C, we denote by $\mathsf{size}(C)$ the size of C, and by $\mathsf{depth}(C)$ the depth of C (indicating parallel time). For an interactive circuit iC, $\mathsf{depth}(iC)$ denotes the sum of the depths of its next-step circuits. We now define depth-bounded circuit ensembles.

Definition 3 (Depth-Bounded Circuits). *For any function $d(\cdot)$, an ensemble of circuits $C = \{C_\lambda\}_{\lambda\in\mathbb{N}}$ is d-depth-bounded if for all λ, $\mathsf{depth}(C_\lambda) \leq d(\lambda)$ and $\mathsf{size}(C_\lambda) \leq \mathsf{poly}(\lambda)$. An interactive circuit $\mathsf{iC} = \{\mathsf{iC}_\ell\}_{\ell\in[L]}$ is D-depth-bounded if $D > \sum_{\ell\in[L]} \mathsf{depth}(\mathsf{iC}_\ell)$ and $\mathsf{poly}(\lambda) \geq \sum_{\ell\in[L]} \mathsf{size}(\mathsf{iC}_\ell)$.*

We next define a notion of depth-bounded computational indistinguishability.

Definition 4 (Depth-Bounded Indistinguishability). *Two ensembles $X = \{X(a,n)\}_{a\in\{0,1\}^*,n\in\mathbb{N}}$ and $Y = \{Y(a,n)\}_{a\in\{0,1\}^*,n\in\mathbb{N}}$ are d-*depth-indistinguishable*, denoted $\overset{d}{\approx}$ if for every d-depth-bounded distinguisher $D = \{D_n\}_{n\in\mathbb{N}}$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that for every $a \in \{0,1\}^*$ and every $n \in \mathbb{N}$*

$$\Pr[D_n(X(a,n)) = 1] - \Pr[D_n(Y(a,n)) = 1] \leq \mathsf{negl}(n)$$

4.3 Time-lock Puzzles

We adapt a definition of puzzles from Bitansky et al. ([8] Definition 3.1).

Definition 5 (Puzzle). *A* puzzle *for solution domain $M = \{M_\lambda\}_\lambda$ is a pair of algorithms $\mathsf{Puz} = (\mathsf{Puz.Gen}, \mathsf{Puz.Solve})$ for which*

- *$Z \leftarrow \mathsf{Puz.Gen}(\tau, \chi)$ is a probabilistic algorithm over difficulty parameter $\tau \in \mathbb{N}$ and solution $\chi \in M_\lambda$, where λ is a security parameter, and outputs puzzle Z.*
- *$\chi \leftarrow \mathsf{Puz.Solve}(Z)$ is a deterministic algorithm that takes as input puzzle Z and outputs solution $\chi \in M_\lambda$.*

subject to the following constraints:

- ***Completeness:** For every security parameter λ, difficulty parameter τ, solution $\chi \in M_\lambda$, and puzzle Z in the support of $\mathsf{Puz.Gen}(\tau, \chi)$, $\mathsf{Puz.Solve}(Z)$ outputs χ.*
- ***Efficiency:***
  - *$Z \leftarrow \mathsf{Puz.Gen}(\tau, \chi)$ can be computed in size $\mathsf{poly}(\log \tau, \lambda)$.*
  - *$\mathsf{Puz.Solve}(Z)$ can be computed in size $\tau \cdot \mathsf{poly}(\lambda)$.*

We continue by adapting the more constrained definition of a *time-lock* puzzle by Bitansky et al. ([8] Definition 3.2).

Definition 6 (Time-lock Puzzle). *A puzzle* $\mathsf{Puz} = (\mathsf{Puz.Gen}, \mathsf{Puz.Solve})$ *is a* time-lock *puzzle for solution domain* $M = \{M_\lambda\}_\lambda$ *with gap* $\varepsilon < 1$ *if there exists a polynomial* $r(\cdot)$ *such that for every polynomial* $t(\cdot) \geq r(\cdot)$ *and every polynomial size,* t^ε*-depth-bounded adversary* $\mathcal{A} = \{\mathcal{A}_\lambda\}_{\lambda \in \mathbb{N}}$*, there exists a negligible function* negl *such that for every* $\lambda \in \mathbb{N}$*, and every pair of solutions* $\chi_0, \chi_1 \in M_\lambda$*:*

$$\Pr[b \leftarrow \mathcal{A}_\lambda(Z) \colon b \leftarrow \{0,1\}, Z \leftarrow \mathsf{Puz.Gen}(t(\lambda), \chi_b)] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

4.4 Residual Complexity and the Critical Time

Residual Complexity. We introduce *a new basic definition of the residual complexity of a puzzle*, which describes the remaining hardness of solving a randomly sampled puzzle after a given amount of solving time. Residual complexity measures the pseudo-entropy [25, 37] of a puzzle solution from the perspective of a computationally bounded solver.

Definition 7 (Residual Complexity). *For a function* $r \colon \mathbb{N} \to [0,1]$*, we say that a puzzle* Puz *with solution domain* $M = \{M_\lambda\}_{\lambda \in \mathbb{N}}$ *has* (d, r) *residual complexity if for every depth* d*-bounded adversary* A_d*, and every* $\lambda \in \mathbb{N}$*:*

$$\Pr[\chi \leftarrow A_d(Y) \colon \chi \leftarrow M_\lambda, Y \leftarrow \mathsf{Puz.Gen}(\tau, \chi)] \leq r(\lambda)$$

When d is implied by context, we refer the residual complexity of a puzzle by the function r. When we consider the residual complexity of a puzzle at a particular depth d, we explicitly write r_d. The "remaining hardness" of the puzzle is $1 - r(\lambda)$.

"The Critical Time" of a Time-lock. In the life of every time-lock puzzle, there is a point at which the adversary has learned "too much" information about the solution (according to the protocol designer), expressed by a threshold residual complexity $r^*(\lambda)$. We call this point the *critical time* (or critical depth) and denote it by t^*. Specifically, $t^* = t^*(\tau, \lambda, r^*)$ depends on the solution time τ of the puzzle, the security parameter λ that tunes the puzzle's difficulty, and a threshold residual complexity $r^* = r^*(\lambda)$ for guessing the solution. $t^*(\tau, \lambda, r^*)$ is the moment at which the leakage of the puzzle exceeds the threshold $r^*(\lambda)$.

Note that because the leakage curve is a representation of hypothesized hardness of a computational problem at varying depths, the critical time represents only a belief by the protocol designer. It is possible to conservatively estimate the critical time (by assuming it occurs earlier for a particular guessing probability) without negatively affecting security.

5 Modeling Secure Multi-Party Computation

This section discusses in detail the modeling issues that arise in our work from composition of timed primitives with other cryptographic computations, including simulating leaky functionalities.

To provide a full treatment of depth-secure multi-party computation, we present two models:

1. A "general" model which adapts the Universal Composability (UC) framework [12] such that all parties (including the environment, trusted third party, and adversary) are modeled as interactive circuits.
2. A "sequential" model, which is useful for proving security of sequential composition of protocols which cannot be proven secure in our more general model, but is otherwise similar, and adapts standard sequential models to our fine-grained treatment.

We then present our definitions for depth-secure computation and theorems – both general and sequential – for how depth-secure protocols compose.

5.1 General Execution Model

In our generalized, UC-like model, we consider an execution in the presence of an *environment* that provides inputs to parties and reads their outputs. The environment is an interactive circuit which directs the execution. It delivers inputs to parties as well as messages that have been sent to them by the adversary. The environment is also responsible for directing query responses between interactive circuits. Each party that receives an input or message from the environment proceeds by evaluating its next-step circuit, after which control is returned to the environment.

The adversary informs the environment which parties it would like to (adaptively) corrupt, and the environment passes the adversary all of the corrupt parties' inputs, the queries they make, and the responses they receive (the latter two are analogous to the messages they send and receive, adapted for our model). The adversary may also inform the environment before the execution which parties it will corrupt from the start; in this case, the environment passes the adversary those parties' inputs and the adversary may choose to replace their inputs by responding to the environment. Only after this exchange, the environment provides inputs to all honest parties. This models that an adversary may select inputs in order to affect a computation.

Synchronization. When all parties that are online in execution evaluate one level of depth of computation at the same rate, we say they proceed *in lockstep.* We use this assumption to prove a stronger version of our composition theorems.

For a full treatment of the execution model, refer to Appendix A.1.

The Ideal/Real Paradigm in the General Model. We next describe our general ideal/real paradigm for granular-depth secure multi-party computation (MPC).

Execution in the Real Model. In the real model, participants execute a protocol π to compute the desired functionality $\mathcal{F}$ without a trusted party. At the end of the execution, honest parties output their protocol outputs. The corrupt parties output nothing. The adversary outputs an arbitrary function of its inputs and the messages that corrupt parties have received. The environment learns every output. The random variable $\mathsf{REAL}_{\pi,\mathcal{A}(z),\mathcal{Z}}(\overline{x})$ denotes the output of the environment in a real execution of π with honest inputs $\overline{x}$, auxiliary input z to $\mathcal{A}$, with environment $\mathcal{Z}$.

Execution in the Ideal Model. In an ideal execution, the parties interact with a trusted party by submitting all of their inputs to the trusted party in the beginning of the execution. The trusted party for a leaky functionality responds to the parties by dividing an execution into *phases* such that at the end of each phase, the parties receive some output.

At the end of an execution, honest parties output whatever they have received from the trusted party. Corrupt parties output nothing, and the adversary outputs an arbitrary function of its input and the messages that corrupt parties have received from the trusted party. The environment learns every output. The random variable $\mathsf{IDEAL}_{\mathcal{F},\mathcal{A}(z),\mathcal{Z}}(\overline{x})$ denotes the output of the environment in an *ideal execution* of functionality $\mathcal{F}$ on honest inputs $\overline{x}$, auxiliary input z to $\mathcal{A}$, with environment $\mathcal{Z}$.

5.2 Sequential Model

Our sequential model is like the general model, except that each protocol execution is considered in isolation, and instead of being directed by the environment, it is directed by the adversary itself. The adversary controls message deliveries and may adaptively corrupt parties throughout an execution. When the adversary delivers a message to a party, it evaluates the party's next step circuit. It is then responsible for forwarding any messages returned in the circuit's queries, as per the oracle-assisted interface explained in Section 4.1. The adversary can additionally adaptively corrupt parties and inject messages, analogously to the exposition in Appendix A.1.

The Real/Ideal Paradigm in the Sequential Model.

Execution in the Real Model. In the real model, the parties execute a protocol π in the presence of an adversary $\mathcal{A}$. The random variable $\mathsf{REAL}_{\pi,\mathcal{A}(z)}(\overline{x})$ denotes the execution transcript on a real execution of π with honest inputs $\overline{x}$ and auxiliary input z to adversary $\mathcal{A}$. The execution transcript includes all of the honest parties' inputs, the messages received by honest parties, and the adversary's output.

Execution in the Ideal Model. As in the general model, in the ideal experiment the honest parties send their inputs to a trusted third party, and the third party delivers the results. In our sequential model, the simulator generates an execution transcript by interacting with the third party on behalf of the honest parties. The random variable $\mathsf{IDEAL}_{\mathcal{F},\mathcal{S}(z)}(\overline{x})$ denotes an execution transcript generated by an adversary $\mathcal{S}$ in an idealized execution of functionality $\mathcal{F}$ on honest inputs $\overline{x}$ and auxiliary input z to $\mathcal{S}$.

5.3 Depth-Bounded Secure Multi-Party Computation

Depth Constraints. For a meaningful definition of secure multi-party computation (MPC) with timed primitives, the computational power of the simulator must be constrained in a manner similar to the adversary's. Otherwise, if the depth of the simulator is substantially more than the adversary, then the simulator could (for example) solve a time-lock puzzle, and use the solution in the simulation. It would be meaningless to argue privacy by claiming that any information the adversary can learn about the honest parties' inputs in a real execution could also be learned by a simulator *which explicitly solves* a time-lock puzzle in order to learn secret information (such as honest parties' inputs).

Our definitions below therefore constrain the depths of both the simulator and the adversary. We also depth-constrain the distinguisher, intuitively because for timed primitives we need only to show security *for some amount of time.*

Definition 8 (Depth-Bounded Secure Computation: General). *Let $d_a = d_a(\lambda)$, $d_s = d_s(\lambda)$, and $d_e = d_e(\lambda)$. Protocol π (d_a, d_s, d_e)-depth securely computes $\mathcal{F}$ if there exists a d_s-depth-bounded $\mathcal{S}$ such that for every real-world d_a-depth-bounded adversary $\mathcal{A}$ and every d_e-depth-bounded environment $\mathcal{Z}$, the following two ensembles are d_e-depth indistinguishable:*

$$\{\mathsf{REAL}_{\pi,\mathcal{A}(z),\mathcal{Z}}(\overline{x})\}_{\overline{x}\in(\{0,1\}^*)^n, z\in\{0,1\}^*}$$

$$\{\mathsf{IDEAL}_{\mathcal{F},\mathcal{S}(z),\mathcal{Z}}(\overline{x})\}_{\overline{x}\in(\{0,1\}^*)^n, z\in\{0,1\}^*}$$

Remark 2. Ours definitions for composition say that a protocol (d_a, d_s, d_e)-securely computes some functionality if there is a d_s-depth bounded universal simulator $\mathcal{S}$ such that for every d_a-depth-bounded adversary, $\mathcal{S}$ produces a distribution of views that is d_e-depth indistinguishable from a real execution. Although we reverse the order of quantifiers for the simulator and adversary in the definition from the standard ordering, most proofs are written by providing a universal simulator that works for any adversary.

The depth of the distinguisher. The constraint on a distinguisher's depth (in this case, the environment; below, the distinguisher) is a significant weakening of the definition compared to those by Goldreich or Lindell's [23, 28], as neither constrains the depth of the distinguisher by a granular polynomial. However, this weakening is sufficient for our setting, since in practice, if a time-locked output will eventually be revealed anyway, we require indistinguishability of the simulation only for the duration of the experiment.

Depth-Secure Computation: Sequential. In the sequential model, as explained above, the execution is directed by the adversary, and the real and ideal experiments should be indistinguishable to a depth-bounded distinguisher who receives a transcript of the execution. (We still use the notation d_e to represent the depth of the distinguisher despite removing the role of the environment; above, d_e is the depth of the distinguishing environment.)

Definition 9 (Depth-Bounded Secure Computation: Sequential). *Let $d_a = d_a(\lambda)$, $d_s = d_s(\lambda)$, and $d_e = d_e(\lambda)$. Protocol π (d_a, d_s, d_e)-depth securely computes $\mathcal{F}$ if there exists a d_s-depth-bounded $\mathcal{S}$ such that for every d_a-depth-bounded real-world adversary $\mathcal{A}$, the following two ensembles are d_e-depth indistinguishable:*

$$\{\mathsf{REAL}_{\pi,\mathcal{A}(z)}(\overline{x})\}_{\overline{x}\in(\{0,1\}^*)^n, z\in\{0,1\}^*}$$

$$\{\mathsf{IDEAL}_{\mathcal{F},\mathcal{S}(z)}(\overline{x})\}_{\overline{x}\in(\{0,1\}^*)^n, z\in\{0,1\}^*}$$

5.4 Simulation for Temporary Privacy

In some applications, sensitive information is revealed during the computation, but must not be revealed before some specific point in time. We call this paradigm *temporary privacy.* For example, consider an accountable computing application, where parties time-lock their inputs and are held accountable to them at a later time. In standard definitions of MPC[23, 28], there is no way to quantify security of such a protocol. These inputs would be output by all parties, making any security reduction trivial: because the simulator would receive all parties' inputs (as one party's output), the standard reduction for proving security would declare that no adversary could learn more information than a simulator *which already knows all of the parties' inputs.*

Therefore, the formalization of temporary privacy requires that the simulator knows no more information during the computation than the adversary. In such a situation, the honest parties' outputs (which contain their inputs) are withheld from the simulator until they are revealed to the adversary. By this restriction, the proven statement is that the adversary can do no worse than a simulator *which knows the same amount of information at each step of the computation.* In Section 7.2, we prove security of such a scheme by allowing the simulator to equivocate the output of a timed puzzle.

6 Composition of Depth-Secure Protocols

We now treat the composition of depth-secure protocols. In the following, we use the notation π^ρ to denote that protocol π calls ρ as a subroutine, as per the convention by Canetti [12]. We use the notation that $\Gamma^{\pi,\rho}$ denotes the concurrent composition of π and ρ.

6.1 General/Concurrent Composition

We now state our general composition theorem, which includes concurrent composition.

Theorem 3 (Composition of Two Depth-Secure Protocols). *Let π (d_a, d_s, d_e)-depth-securely compute functionality F and let ρ (d'_a, d'_s, d'_e)-depth-securely compute functionality G. Then $\Gamma^{\pi,\rho}$ is $(d_a - d'_s, d_s + d'_s, \min(d_e, d'_e))$-secure.*

Proof (Sketch). We define a simulator $\mathcal{S}$ for π^ρ that simply composes the simulators $\mathcal{S}^\pi$ and $\mathcal{S}^\rho$ which exist by assumption. We then perform a reduction that shows if there is an attack against π^ρ, we can isolate an attack against π in the G-hybrid model. The reduction is straightforward, although it must carefully consider the depths of all simulators and adversaries. Given an adversary $\mathcal{A}$ which attacks π^ρ, we define an adversary $\mathcal{B}$ such that $\mathcal{B}$ runs $\mathcal{A}$ as a black box, and $\mathcal{B}$ forwards messages sent by $\mathcal{A}$ to their recipients. The only exception is that $\mathcal{B}$ must simulate an execution of ρ for $\mathcal{A}$ when $\mathcal{A}$ expects ρ to be called.

Proof. First we create a simulator $\mathcal{S}$ for the composition. $\mathcal{S}$ works by invoking the simulators $\mathcal{S}^\pi$ and $\mathcal{S}^\rho$ (for π and ρ, respectively) in parallel. Note that its depth is at most $d_s + d'_s$.

For the sake of the following lemma, we use the notation $\overline{x}$ to denote the honest parties' inputs and z to denote an auxiliary input. Because we consider two separate protocols in concurrent composition, we let $\overline{x} = (\overline{x}_1, \overline{x}_2)$ where $\overline{x}_1$ are for π and $\overline{x}_2$ are for ρ, and similarly we let $z = (z_1, z_2)$ with analogous association.

We now state our main lemma, from which the proof follows.

Lemma 1. *Let $f' = \min(d_e, d'_e)$. For every $d_a - d'_s$-depth adversary $\mathcal{A}$, every $\min(d_e, d'_e)$-depth environment $\mathcal{Z}$, and every $\overline{x} \in (\{0,1\}^{\mathsf{poly}(\lambda)})^n$ and $z \in \{0,1\}^{\mathsf{poly}(\lambda)}$:*

$$\mathsf{REAL}_{\Gamma^{\pi,\rho},\mathcal{A}(z),\mathcal{Z}}(\overline{x}) \overset{f'}{\approx} \mathsf{IDEAL}_{\zeta^{F,G},\mathcal{S}(z),\mathcal{Z}}(\overline{x})$$

Proof. Assume towards contradiction that the above is not true. Then there exist a $(d_a - d'_s)$-depth adversary $\mathcal{A}$, a $\min(d_e, d'_e)$-depth environment $\mathcal{Z}$, and inputs $\overline{x}, z$ for which $(\mathcal{A}, \mathcal{Z})$ distinguishes the two distributions (for any simulator $\mathcal{S}$).

We build an adversary $\mathcal{B}$ and environment $\mathcal{E}$ that distinguish the execution of π from its simulation on honest inputs $\overline{x}$ and advice string z. $\mathcal{E}$ will run $\mathcal{Z}$ as a black box, forwarding messages to $\mathcal{Z}$, sending whatever messages $\mathcal{Z}$ sends, and outputting whatever $\mathcal{Z}$ outputs. $\mathcal{B}$ will use $\mathcal{A}$ and $\mathcal{Z}$ to attack its real-world execution of π, but $\mathcal{B}$ will simulate the concurrent execution of ρ for $\mathcal{A}$ (and $\mathcal{Z}$) *in parallel* to the execution of π. By the assumption that ρ is secure, this will imply that $\mathcal{B}$ and $\mathcal{E}$ use $\mathcal{A}$ and $\mathcal{Z}$ to distinguish π from its simulation, reaching contradiction.

We first introduce notation for an experiment which $\mathcal{B}$ uses to attack π. In this experiment, $\mathcal{B}$ and $\mathcal{E}$ will attack a real execution of π by running $\mathcal{A}$ and $\mathcal{Z}$ as black boxes; when they expect messages from the run of ρ, $\mathcal{B}$ simulates a concurrent execution of ρ using $\mathcal{S}^\rho$. We denote the experiment by $\mathsf{REAL}^{\mathcal{B}}_{\Gamma^{\pi,G},\mathcal{A}(z),\mathcal{Z}}(\overline{x})$. We argue that by the security of ρ, $\mathcal{A}$'s view of this distribution must be indistinguishable from its view of $\mathsf{REAL}_{\Gamma^{\pi,\rho},\mathcal{A}(z),\mathcal{Z}}(\overline{x})$.

Claim 1 *Let $f' = \min(d_e, d'_e)$. For any f'-depth $\mathcal{Z}$, for all $\overline{x} \in (\{0,1\}^{\mathsf{poly}(\lambda)})^n$ and $z \in \{0,1\}^{\mathsf{poly}(\lambda)}$*

$$\mathsf{VIEW}_{\mathcal{A}}(\mathsf{REAL}_{\Gamma^{\pi,\rho},\mathcal{A}(z),\mathcal{Z}}(\overline{x})) \overset{f'}{\approx} \mathsf{VIEW}_{\mathcal{A}}(\mathsf{REAL}^{\mathcal{B}}_{\Gamma^{\pi,G},\mathcal{A}(z),\mathcal{Z}}(\overline{x}))$$

Proof. The difference between the two distributions is that on the right, $\mathcal{B}$ simulates an execution of ρ using the simulator $\mathcal{S}^\rho$ and provides those messages to $\mathcal{A}$ (and $\mathcal{Z}$), and then continues to call $\mathcal{A}$ after the call to $\mathcal{S}^\rho$ using messages from its real execution. By assumption, $\mathcal{A}$ is $(d_a - d'_s)$-depth-bounded and $d_a < d'_e$. Therefore, $\mathcal{A}$ must not be able to distinguish the messages in the real execution of ρ on the left from the simulation on the right. By a similar argument, neither can (any) $\mathcal{Z}$. The claim follows from the additional fact that all other messages in $\mathcal{A}$'s view are distributed indistinguishably in both experiments, since they are both from a real execution of π.

We make another claim that is analogous to the previous, but for the ideal experiment. We claim that $\mathcal{A}$ cannot distinguish between an idealized execution of $\zeta^{F,G}$ in which $\mathcal{S}$ generates $\mathcal{A}$'s view of the execution, and an idealized execution of F in which $\mathcal{B}$ forwards messages generated for it by $\mathcal{S}^\pi$, and in place of the ideal functionality call to G, $\mathcal{B}$ generates a view of the call to ρ (realizing G) by simulating $\mathcal{S}^\rho$, and forwards these messages to $\mathcal{A}$. (The right-hand distribution denoted $\mathsf{IDEAL}^{\mathcal{B}}_{\zeta^{F,G},\mathcal{S}^\pi(z),\mathcal{Z}}(\overline{x})$ represents the ideal world execution of $\mathcal{B}$'s attack on π, in which $\mathcal{B}$ must still simulate the functionality G for $\mathcal{A}$.)

Claim 2 *For all $\overline{x} \in (\{0,1\}^{\mathsf{poly}(\lambda)})^n$ and $z \in \{0,1\}^{\mathsf{poly}(\lambda)}$*

$$\mathsf{VIEW}_{\mathcal{A}}(\mathsf{IDEAL}_{\zeta^{F,G},\mathcal{S}(z),\mathcal{Z}}(\overline{x})) \equiv \mathsf{VIEW}_{\mathcal{A}}(\mathsf{IDEAL}^{\mathcal{B}}_{\zeta^{F,G},\mathcal{S}^\pi(z),\mathcal{Z}}(\overline{x}))$$

Proof. The proof is analogous to the previous. However, in this case, $\mathcal{B}$ perfectly simulates the execution of ρ in comparison to $\mathcal{A}$'s view in the ideal execution of π^ρ, since $\mathcal{B}$ does exactly the same thing that $\mathcal{S}$ does: both run $\mathcal{S}^\rho$. In light of this observation, the claim is mostly notational, since on the left $\mathcal{A}$ receives messages from $\mathcal{S}$, and on the right it receives the same messages, simply forwarded by $\mathcal{B}$ (and generated by $\mathcal{B}$ for the call to G).

Note that $\mathcal{B}$ runs $\mathcal{A}$ and $\mathcal{S}^\rho$ as black boxes, so its depth is $d_a - d'_s + d'_s = d_a$. $\mathcal{E}$'s depth is at most $\min(d_e, d'_e)$ because it is identical to $\mathcal{Z}$.

If there exist $\overline{x}$, z for which $\mathcal{A}$,$\mathcal{Z}$ distinguish $\mathsf{REAL}_{\Gamma^{\pi,\rho},\mathcal{A}(z),\mathcal{Z}}(\overline{x})$ and $\mathsf{IDEAL}_{\zeta^{F,G},\mathcal{S}(z),\mathcal{Z}}(\overline{x})$, then by Claims 1 and 2, $\mathcal{B}$ and $\mathcal{E}$ distinguish $\mathsf{REAL}^{\mathcal{B}}_{\Gamma^{\pi,G},\mathcal{A}(z),\mathcal{Z}}(\overline{x})$ and $\mathsf{IDEAL}^{\mathcal{B}}_{\zeta^{F,G},\mathcal{S}^\pi(z),\mathcal{Z}}(\overline{x})$. The latter two are exactly $\mathcal{B},\mathcal{E}$'s game against π, except that we specified a strategy by which $\mathcal{B}$ simulates a concurrent execution of ρ which it feeds to $\mathcal{A}$ when it runs $\mathcal{A}$. Therefore, we have a contradiction to the security of π, because $(\mathcal{B},\mathcal{E})$ are a (d_a, d_e) adversary and distinguisher for π.

Remark 3 (The Depths d_a and d'_e). For all composition theorems, we require that $d_a < d'_e$. This is a natural choice; in particular if $d_a \geq d'_e$ then the theorem is not meaningful. Specifically, if $d_a \geq d'_e$, then the adversary for the first protocol is deep enough to distinguish an execution of the protocol ρ which is called by it from the callee's simulation; the composition therefore does not have meaningful real-world consequences, since a realistic adversary against the composition implies an adversary for the callee protocol. For all following theorems, we elide the statement of this requirement.

We see from the composition theorem that when composing two depth-secure protocols in order to achieve security against any d_a^*-depth adversary, the composed protocols must be parameterized so that they are secure against stronger adversaries, due to the loss in security that results from composition. Moreover, the composition remains secure only against the smaller of the two distinguishing environments.

6.2 Sequential Composition

In some cases, a protocol cannot be proven concurrently composable, if the simulator needs to be rewound. We therefore provide a "weaker" theorem for the sequential composition of protocols that cannot be proven secure with respect to the general theorem.

Theorem 4 (Sequential Composition of Two Depth-Secure Protocols). *Let π (d_a, d_s, d_e)-depth-securely compute F in the G-hybrid model, and let ρ (d'_a, d'_s, d'_e)-depth-securely compute G. π^ρ $(d_a - d'_s, d_s \cdot d'_s, \min(d_e, d'_e))$-depth-securely computes F.*

The proof is in Appendix B.

Observe that the decrease in simulation budget for the concurrent composition theorem appears to be "better" than the "weaker" sequential theorem because the simulation budget does not deteriorate as much; however, this is attributable to the fact that the simulator for a concurrently composable protocol must already be more efficient than the simulator for the sequential theorem above, as rewinding is not permitted (as in the UC[12]).

We note that the $(d_s \cdot d'_s)$ term in the $(\cdot, d_s \cdot d'_s, \cdot)$-depth security of the composed protocols is too pessimistic in some cases. In the case that the simulator for the calling protocol never needs to rewind over the invocation of the subroutine protocol, we can prove stronger security for the composition. This is in fact a direct fallback to Theorem 3.

Corollary 1 (Optimistic Sequential Composition of Depth-Secure Protocols). *Let π (d_a, d_s, d_e)-depth-securely compute F in the G-hybrid model, and let ρ (d'_a, d'_s, d'_e)-depth-securely compute G. If the simulator for π in the G-hybrid model never rewinds over the point at which G is invoked, then π^ρ $(d_a - d'_s, d_s + d'_s, \min(d_e, d'_e))$-depth-securely computes F.*

Proof (Sketch). This follows immediately from Theorem 3, and in fact when the simulator does not need to be rewound, the protocol is also concurrently composable.

Discussion. Theorem 4 and Corollary 1 give the bounds on the spectrum of "simulation budget depletion" that may occur when composing depth-secure protocols. Specifically, in order to make a meaningful statement about security, the middle term d_s must remain smaller than both of the outer terms d_a and d_e. For a particular composition, the protocol designer may compute the actual security statement by computing the runtime of the composed simulator.

Multi-Composition When composing multiple protocols concurrently, it is sometimes possible to achieve less degradation than by applying Theorem 3 repeatedly. This is the case if the protocols are run simultaneously with each other and parties compute in lockstep.

Corollary 2 (Lockstep Multi-Composition). *Let π (d_a, d_s, d_e)-securely implement $\mathcal{F}$. Let ρ_1, ρ_2, $\ldots$, ρ_n be protocols such that ρ_i $(d_{a_i}, d_{s_i}, d_{e_i})$-securely implements G_i. Then the* lockstep *concurrent composition of π with ρ_1,$\ldots$,ρ_n is $(d_a - \arg\min_i d_{s_i}, d_s + \arg\max_i d_{s_i}, \min(d_e, \arg\min_i d_{e_i})$ secure.*

Proof (Sketch). The proof is a generalization as the proof for Theorem 3, except that the adversary $\mathcal{B}$ runs the simulators for multiple concurrent protocols simultaneously, and importantly $\mathcal{B}$ can run the other simulators in lockstep. This allows $\mathcal{B}$ to run the simulators of all other concurrent protocols with depth at most the largest of the other protocols' simulators.

6.3 Serial Composition

Our application in Section 8 uses an addition form of composition that we call *serial composition.* Protocols π and ρ are serially composed if *after* π ends at time t_1, some universal output is preserved and used as common input to ρ, which begins at time $t_2 > t_1$.

Claim 3 (Serial Composition) *Let π and ρ be protocols composed serially. The composition is secure without degraded security of either protocol.*

Presented without formal proof. The two protocols run independently and the simulators are not dependent on each other.

7 (Correctly) Applying the Random Oracle: Time-Lock Puzzles from (Leaky) Algebraic Puzzles

In Section 7.1 we provide a construction that "boosts" the security of an arbitrary leaky algebraic puzzle to a time-lock puzzle, given that the leaky puzzle does not leak too much. In Section 7.2 we discuss proof techniques using the random oracle that allow simulation of time-lock puzzles, along with the corresponding security degradation.

Both of these results depend on a random oracle. This *does not* make the puzzle's solution algorithm depend on a random oracle (or equivalent analysis) for each intermediate state. Instead, it allows the algebraic trapdoor structure of the puzzle to leak information on each intermediate solution. Therefore, the puzzles fall into a leaky analytical framework that does not fall into an inconsistent analysis. The random oracle is applied only once to an algebraic solution that may have leaked.

7.1 Algebraic Time-Lock Puzzles with a Single Random Oracle

In Figure 2, we recommend a time-lock puzzle compiler that takes an algebraic time-lock puzzle (such as repeated squaring), and applies one step of a random oracle. This compiler preserves the hardness of learning the puzzle solution despite leakage of the algebraic solution, intuitively by making the puzzle solution depend on *every bit* of the algebraic computation rather than partially predictable given a portion of the solution.

It achieves two important objectives:

1. It provides an algebraic time-lock construction with *consistent analysis.*
2. In Lemma 2, we prove that the resulting puzzle is secure (the solution is hidden) for any leaky algebraic puzzle as long as $O(\lambda)$ bits of the algebraic solution are not leaked.

The construction works as follows: Let the puzzle solution be χ. The algebraic puzzle Z is generated with a randomly sampled solution r. Then, $\mathsf{Puz.Gen}$ masks χ with $H(r)$, returning the the pair (Z, γ), where $\gamma = H(r) \oplus \chi$. When solving the puzzle, first retrieve r' via by solving Z and then compute the true solution χ by calling the random oracle $H(r') \oplus \gamma$.

In this way, the simulator can sample time-lock puzzles with random solutions (via the original scheme) from the same distribution as the honest parties, and then equivocate the final solution by programming the random oracle. Note that this technique can also be combined with adjustments to a base TLP scheme for non-malleability [22].

Theorem 5 (Secure Algebraic Puzzle from a Leaky Puzzle in the ROM). *For any algebraic puzzle* Puz*, apply the time-lock puzzle transformation described in Figure 2; namely, as a last step of an algebraic time-lock puzzle, apply one step of a random oracle. Then* $\widetilde{\mathsf{Puz}}$ *is a time-lock puzzle, and its critical time is no sooner than the last moment when* $\mathsf{Puz.Solve}$ *has not not leaked* $O(\lambda)$ *bits of the algebraic solution.*

Proof (Sketch). The proof's core lemma follows:

Lemma 2. *For every intermediate step of* $\mathsf{Puz.Solve}$ *for which* $O(\lambda)$ *bits of the algebraic solution are not leaked, the residual complexity of the corresponding step of* $\widetilde{\mathsf{Puz.Solve}}$ *is* $O(2^{\zeta(\lambda)})$*, where* ζ *is linear in the security parameter* λ*.*

Proof. Consider any depth-d adversary $\mathcal{A}_d$, and let $\mathsf{Adv}^{\mathsf{Puz}}_{\mathcal{A}_d}$ be the probability that $\mathcal{A}_d$ guesses the solution of the puzzle.

If $\gamma = O(\lambda)$ bits of the solution are not leaked to $\mathcal{A}_d$, then $\mathsf{Adv}^{\mathsf{Puz}}_{\mathcal{A}_d} \leq O(\frac{1}{2^\gamma})$. Even if $\mathcal{A}_d$ is permitted polynomially many guesses, then $\mathsf{Adv}^{\mathsf{Puz}}_{\mathcal{A}_d} \leq m \cdot \frac{1}{2^\gamma}$, where $m \in \mathsf{poly}(\lambda)$. It follows that $\mathsf{Adv}^{\mathsf{Puz}}_{\mathcal{A}_d} \leq O(2^{\zeta(\lambda)})$, where ζ is linear in λ. Note that this is negligible in λ.

Given Lemma 2, it is easy to extend the analysis to show that the resulting scheme is a time-lock puzzle. While Puz does not leak $O(\lambda)$ bits of the algebraic solution, the probability of guessing the corresponding solution of the random oracle-assisted construction puzzle remains negligible.

The full proof follows by applying a game-based framework in which the adversary chooses two solutions and provides them to the challenger. Because the challenger samples random solutions for its algebraic puzzles, the distributions of two puzzles provided to the adversary are identical. In order for the adversary to distinguish between the two, it must therefore solve one of the algebraic puzzles. It follows that $\widetilde{\mathsf{Puz}}$ is a time-lock puzzle and the critical time occurs no earlier than when Puz's solution no longer hides at least $O(\lambda)$ bits of the algebraic solution.

TLP from Leaky Algebraic Puzzle

Let Puz be an algebraic puzzle scheme and $H_\lambda \colon M_\lambda \to \{0,1\}^\kappa$ a random oracle, such that $\{0,1\}^\kappa$ is a superset of M. (We leave implicit the re-interpretion from $\{0,1\}^\kappa$ to M_λ as the final step of recovering the solution.[a]) Construct $\widetilde{\mathsf{Puz}}$ for domain M as follows:

[a] When M is all bit-strings of a certain length, this is trivial.

```
$\widetilde{\mathsf{Puz}.\mathsf{Gen}}(\tau, \chi)$
    $r \leftarrow M_\lambda$
    $Z \leftarrow \mathsf{Puz}.\mathsf{Gen}(\tau, r)$
    $\gamma \leftarrow H(r) \oplus \chi$
    $Z' \leftarrow (Z, \gamma)$
    return $Z'$
```

```
$\widetilde{\mathsf{Puz}.\mathsf{Solve}}(Z')$
    parse $Z' = (Z, \gamma)$
    $r' \leftarrow \mathsf{Puz}.\mathsf{Solve}(Z)$
    return $H(r') \oplus \gamma$
```

Fig. 2: Construction of a TLP from a Leaky Algebraic Puzzle

7.2 Simulation Techniques for Temporary Privacy

When a protocol π requires solving a time-lock puzzle and then using the solution of the puzzle, the protocol requires that the solution remain private until the puzzle is solved, but then is no longer private. Because our model (Section 5.4) requires that the simulator does not know information until the adversary learns it, our simulator must equivocate these time-lock puzzles late in the simulation.

We refer to a puzzle scheme that allows the simulator to equivocate at the end of the simulation as *simulation-equivocable.* The construction in Figure 2 makes the compiled scheme simulation-equivocable by programming the random oracle at the moment of equivocation, which is a standard techniques in the random oracle model.

Timed-Advantaged Equivocation In order to program the random oracle to equivocate the result *at the right time*, the simulator needs a small time advantage over the adversary. This means that the simulator must learn the puzzle solution from the ideal functionality (just) before t^*, which is when the adversary is assumed to learn the solution. The protocol using time-lock puzzles must take this time difference into account.

In practice, this degrades the security of the protocol. Because the simulator (and adversary) learns the solution effectively at t^*, the proof only guarantees (temporary) privacy until that point.

8 Simultaneous Multiple Round Auction

As an illustration of our new techniques, we present a protocol for a simultaneous multiple round auction (SMRA) [32]. Before presenting the SMRA protocol, we begin with analysis of a single-round auction that will be used to analyze the simultaneous multi-round version.

8.1 Single Auction Using Time-lock Puzzles

The functionality for a single-round, single-item auction is provided in Figure 3 and the protocol is in Figure 4. For the sake of clarity, we assume that a broadcast can be completed in one time step. Specifically, this means that as long as a party broadcasts its bid *strictly before* time t^{bid}, then it is received by all parties by time t^{bid}. [5]

$\mathcal{F}_{\mathsf{auction}}$

Public Parameters:

- $\mathcal{P} = \{P_1, P_2, \dots, P_n\}$ denotes the parties participating in the auction, where n is the number of parties participating.

The functionality proceeds as follows, with a predetermined time t^{bid}:

- **Bid:** Each party P_i sends a bid (P_i, b_i) to $\mathcal{F}_{\mathsf{auction}}$. If a party does not send a bid before t^{bid}, then the functionality ignores the bid.
- **Reveal:** After all parties submit their round-r bids, $\mathcal{F}_{\mathsf{auction}}$ reveals all bids (including the bidders) to all parties. If a party P_j did not register a bid before t^{bid}, then $\mathcal{F}_{\mathsf{auction}}$ sends $(P_j, \emptyset)$ in place of P_j's bid.

Fig. 3: Single Round Auction Functionality

[5] This assumption can be relaxed by adjusting the protocol for time delays and by enforcing a time at which all parties agree that bids are received, or instead of a time, a certain location on a distributed ledger.

π^{auction}

Public Parameters:

- $\mathcal{P} = \{P_1, P_2, \ldots, P_n\}$ denotes the parties participating in the auction, where n is the number of such parties.

Assignment of Phases: The protocol is divided into phases marked by the following moments in time:

- t^{start} is the starting time
- t^{bid} is the time before which bids must be received
- t^{end} is the time before which puzzles containing bids must be solved

Inputs: Each party P_i has an input b_i
Protocol:

- **Bid:** P_i computes $\mathsf{puz}_i \leftarrow \mathsf{Puz.Gen}(t^{\mathsf{end}} - \hat{t}, b_i)$ and broadcasts (P_i, puz_i) to all parties, where $\hat{t}$ is the time at which P_i will broadcast the puzzle. ($\hat{t}$ must be before t^{bid}.)
- **Solve Bids:** Upon receiving message (P_j, puz_j) – only if it is received before t^{bid} – P_i computes $b_j \leftarrow \mathsf{Puz.Solve}(\mathsf{puz}_j)$. (These computations must be done in parallel.) If a message (P_j, puz_j) is received after t^{bid}, then ignore the message. If $\mathsf{Puz.Solve}(\mathsf{puz}_j)$ has not completed before t^{end}, then ignore the puzzle.
- **Output:** Output $(j, b_j)_{j \in \mathcal{P}}$. Let the j that maximizes b_j be the "winner." If j's puzzle was not received before t^{bid} or solved before t^{end}, then output $(j, 0)$ above.

Fig. 4: Single Auction Protocol

Choosing t^{bid} and t^{end}: In π^{auction}, t^{bid} and t^{end} must be tuned by the leakage curve of the chosen puzzle scheme(s). Specifically, it must be the case that the adversary cannot learn information about any honest party's puzzle before t^{bid} in order to construct a new bid based on its information of honest parties' bids in the same round. This follows by all parties constructing their puzzles such that t^{bid} is before t^*, where the leakage at t^* is set as a function of λ to be negligible.

When composing this protocol concurrently with itself for the multi-item auction in Section 8.2, we will show that parties must additionally tune t^{bid} to account for the degradation incurred by concurrent composition.

By the security implied by the critical time, privacy of the bids and therefore security of the protocol holds against a t^{bid}-size adversary.

Theorem 6 (Security of π^{auction}). *Let* Puz *be an equivocable, non-malleable time-lock puzzle scheme. Then* π^{auction} *$(t^{\mathsf{bid}}, d_s, d_e)$-securely implements* $\mathcal{F}_{\mathsf{auction}}$*, where* $d_s(\lambda) = \mathsf{depth}(\mathsf{Puz.Gen}(\lambda)) + t^{\mathsf{end}}(\lambda) - t^{\mathsf{bid}}(\lambda)$ *and* d_e *is an arbitrary polynomial in the security parameter* λ.

Proof Sketch. In the proof, we assume a rushing adversary which is permitted to see all of the bids by honest parties before it constructs its own bids, but it still must submit those before t^{bid}.

The simulator $\mathcal{S}$ for the protocol switches the time-lock puzzles generated by the honest parties for random puzzles which it can equivocate, as described in Section 7.2. $\mathcal{S}$ can then leak the result of each puzzle according to the puzzle's leakage function. The simulator as above requires only the depth of $\mathsf{Puz.Gen}$ to replace the honest parties' puzzles (in parallel).

The crux of the proof is to show that the adversary cannot distinguish between the puzzles generated by the simulator and those of the honest parties, and additionally that the adversary cannot generate puzzles with bids that depend on the honest parties' puzzles that it receives (even without solving for the honest parties' bids). For this we rely on a definition of *non-malleability*, which we defer to Appendix E.1.

We remark that for a non-malleable puzzle, it is possible to use the construction of Chvojka and Jager [14], and the technique of Section 7.1 for equivocation.

The full proof is deferred to Appendix E. We present the proof with respect to definitions for non-malleability adapted from [22], which are included in Appendix E.1. After the proof we remark on how to adapt it for definitions of CCA-secure timed commitments, as provided by [26, 14].

8.2 Simultaneous Multiple Round Auction

In this section we apply our analysis to build a simultaneous multiple round auction. Figure 6 contains our protocol for a simultaneous multiple round auction, for which the functionality is in Figure 5. We illustrate use of our composition theorems by replacing the **Bid** and **Reveal** steps in each round with a call to $\mathcal{F}_{\mathsf{auction}}$ implemented by π^{auction}.

Adjusting t^{bid} Due to Degradation. The composition and analysis are *not black box* due to degradation incurred by the composition theorem (Theorem 3). Recall that each round of the auction in π^{auction} is secure only against an adversary of depth $d_a = t^{\mathsf{bid}}$. By the composition theorem, each execution becomes secure against an adversary of depth $d_a - d_s$. Therefore, when parameterizing t^{bid} for a single round of π^{SMRA}, t^{bid} should be *decreased* (making the time earlier) for the same solving time and security parameters in order to guarantee security of the protocol. For example, considering the execution of a single round of π^{SMRA}, let t_1^{bid} be the corresponding t^{bid} for a single auction as per π^{auction}. For the composition, t^{bid} used in π^{auction} should be $t_1^{\mathsf{bid}} - d_s$ in order for the statements on which Theorem 6 depends to hold.

Theorem 7 (Security of π^{SMRA}). *Let π^{auction} (d_a, d_s, d_e)-securely implement $\mathcal{F}_{\mathsf{auction}}$. In lockstep execution, π^{SMRA} $(d_a - d_s, 2d_s, d_e)$-securely computes the corresponding round of $\mathcal{F}_{\mathsf{SMRA}}$.*

Proof. One round of π^{SMRA} includes α (lockstep) simultaneous executions of π^{auction}. It follows from Theorem 3 that each round is $(d_a - d_s, \alpha d_s, d_e)$-secure.

$\mathcal{F}_{\mathsf{SMRA}}$

Public Parameters:

- $\mathcal{P} = \{P_1, P_2, \ldots, P_n\}$ denotes the parties participating in the auction, where n is the number of such parties.
- $X = \{x_1, x_2, \ldots, x_\alpha\}$ are the items for auction, where α is the number of items being auctioned.

Inputs: Each party has as input a *preference function* $\sigma\colon [\alpha] \times \mathbb{N}^\alpha \times [n]^\alpha \to \mathbb{N}$

The functionality maintains a table B to track wining bids for which initially $B[1] = B[2] = \ldots = B[\alpha] = 0$, a table T to track winning parties for which initially $T[1] = T[2] = \ldots = T[\alpha] = \emptyset$, and a variable $\mathsf{done} \in \{\mathsf{false}, \mathsf{true}\}$. The functionality proceeds in a series of rounds as follows.

- In each round $r = 1, 2, \ldots$ until the termination condition is met:
  - set $\mathsf{done} = \mathsf{true}$
  - **Bid:** For each auction $a \in \alpha$, each party P_i sends a bid $(P_i, b_{i,a,r})$ to $\mathcal{F}_{\mathsf{SMRA}}$.
  - **Reveal:** After all parties submit their round-r bids, $\mathcal{F}_{\mathsf{SMRA}}$ reveals all round-r bids (including the bidders) to all parties.
  - **Update Max Bids and Current Winners:** For each item $a \in [\alpha]$:
    * Let j^*_a be the j that maximizes $b_{j,a,r}$, and let $\hat{b}_a = B[a]$.
    * Assign $B[a] = \max(B[a], b_{j^*,a,r})$.
    * If $B[a] \neq \hat{b}_a$, then assign $T[a] = j^*_a$ and set $\mathsf{done} = \mathsf{false}$.
  - **Termination Condition:** If $\mathsf{done} = \mathsf{true}$ then terminate the loop and end the auction.
- The functionality sends B, T to all parties.

Fig. 5: Simultaneous Multiple Round Auction Functionality

By instead applying Corollary 2 for a lockstep execution, the degradation of the simulator can be reduced such that the composition is $(d_a - d_s, 2d_s, d_e)$-secure.

Because each round of π^{SMRA} is composed *in serial* (one round concludes before the next round begins), the security analyses of all rounds are independent, as per Claim 3 (in Section 6.3).

The computation of d_e is trivial, as all of the protocols are secure against the same d_e-depth environment, and the minimum is taken as the depth security of the final composition.

References

1. Arapinis, M., Lamprou, N., Zacharias, T.: Astrolabous: A universally composable time-lock encryption scheme. In: ASIACRYPT (2). Lecture Notes in Computer Science, vol. 13091, pp. 398–426. Springer (2021)
2. van Baarsen, A., Stevens, M.: On time-lock cryptographic assumptions in abelian hidden-order groups. In: ASIACRYPT (2). Lecture Notes in Computer Science, vol. 13091, pp. 367–397. Springer (2021)

π^{SMRA}

Public Parameters:

- $\mathcal{P} = \{P_1, P_2, \ldots, P_n\}$ denotes the parties participating in the auction, where n is the number of parties participating.
- $X = \{x_1, x_2, \ldots, x_\alpha\}$ are the items for auction, where α is the number of items being auctioned.

Assignment of Rounds: The protocol proceeds in a series of rounds $r = 1, 2, \ldots$ indefinitely until a termination condition is met. Each round is divided into discrete phases over a period of time, and special points in time are defined within the round as follows.

- t_r^{start} is the starting time of round r
- t_r^{bid} is the time before which bids must be received in round r.
- t_r^{end} is the time before which puzzles sent in round r must be solved

For each round r, $t_{r+1}^{\mathsf{start}} = t_r^{\mathsf{end}} + \delta$, where δ is a small, non-negative amount of time that only serves to separate rounds.
Inputs: Each party has as input a *preference function* $\sigma \colon [\alpha] \times \mathbb{N}^\alpha \times [n]^\alpha \to \mathbb{N}$
Protocol:

- Each party maintains a table B to track wining bids for which initially $B[1] = B[2] = \ldots = B[\alpha] = 0$, and a table T to track winning parties for which initially $T[1] = T[2] = \ldots = T[\alpha] = \emptyset$
- In each round $r = 1, 2, \ldots$, until the termination condition below is met, each party P_i proceeds as follows:
  - set $\mathsf{done} = \mathsf{true}$
  - **Construct Round Bid:** For each item $a \in [\alpha]$, P_i computes its round-r bid $b_{i,a,r} \leftarrow \sigma(a, B, T)$. P_i computes $\mathsf{puz}_{i,a,r} \leftarrow \mathsf{Puz.Gen}(t^{\mathsf{end}} - \hat{t}, b_{i,a,r})$ and broadcasts $(P_i, \mathsf{puz}_{i,a,r})$ to all parties, where $\hat{t}$ is the time at which P_i will broadcast the puzzle. ($\hat{t}$ must be before t_r^{bid}.)
  - **Solve Round Bids:** For each message $(P_j, \mathsf{puz}_{j,a,r})$ received by t_r^{bid}, P_i computes $b_{j,a,r} \leftarrow \mathsf{Puz.Solve}(\mathsf{puz}_{j,a,r})$. If a message $(P_j, \mathsf{puz}_{j,a,r})$ is received after t_r^{bid}, then ignore the message. If $\mathsf{Puz.Solve}(\mathsf{puz}_{j,a,r})$ has not completed before t^{end}, then ignore the puzzle.
  - **Update Max Bids and Current Winners** For each item $a \in [\alpha]$:
    - ∗ Let j_a^* be the j that maximizes $b_{j,a,r}$, and let $\hat{b}_a = B[a]$.
    - ∗ Assign $B[a] = \max(B[a], b_{j^*,a,r})$.
    - ∗ If $B[a] \neq \hat{b}_a$, then assign $T[a] = j_a^*$ and set $\mathsf{done} = \mathsf{false}$.
  - **Termination Condition:** If $\mathsf{done} = \mathsf{true}$ then terminate the loop and end the auction.

Fig. 6: Simultaneous Multiple Round Auction Protocol

3. Ball, M., Rosen, A., Sabin, M., Vasudevan, P.N.: Average-case fine-grained hardness. In: STOC. pp. 483–496. ACM (2017)
4. Ball, M., Rosen, A., Sabin, M., Vasudevan, P.N.: Proofs of work from worst-case assumptions. In: CRYPTO (1). Lecture Notes in Computer Science, vol. 10991, pp. 789–819. Springer (2018)

5. Baum, C., David, B., Dowsley, R., Kishore, R., Nielsen, J.B., Oechsner, S.: CRAFT: composable randomness beacons and output-independent abort MPC from time. In: Public Key Cryptography (1). Lecture Notes in Computer Science, vol. 13940, pp. 439–470. Springer (2023)
6. Baum, C., David, B., Dowsley, R., Nielsen, J.B., Oechsner, S.: TARDIS: A foundation of time-lock puzzles in UC. In: EUROCRYPT (3). Lecture Notes in Computer Science, vol. 12698, pp. 429–459. Springer (2021)
7. Benhamouda, F., Lin, H.: k-round mpc from k-round ot via garbled interactive circuits. Cryptology ePrint Archive, Report 2017/1125 (2017), `https://eprint.iacr.org/2017/1125`
8. Bitansky, N., Goldwasser, S., Jain, A., Paneth, O., Vaikuntanathan, V., Waters, B.: Time-lock puzzles from randomized encodings. In: ITCS-2016. pp. 345–356. ACM (2016)
9. Boneh, D., Bonneau, J., Bünz, B., Fisch, B.: Verifiable delay functions. In: CRYPTO (1). LNCS, vol. 10991, pp. 757–788. Springer (2018)
10. Boneh, D., Bünz, B., Fisch, B.: A survey of two verifiable delay functions. Cryptology ePrint Archive, Report 2018/712 (2018), `https://eprint.iacr.org/2018/712`
11. Boneh, D., Naor, M.: Timed commitments. In: Crypto'00. p. 236–254. LNCS, Springer-Verlag, Berlin, Heidelberg (2000)
12. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067 (2000)
13. Canetti, R., Goldreich, O., Halevi, S.: The random oracle methodology, revisited (preliminary version). In: STOC. pp. 209–218. ACM (1998)
14. Chvojka, P., Jager, T.: Simple, fast, efficient, and tightly-secure non-malleable non-interactive timed commitments. In: Public Key Cryptography (1). Lecture Notes in Computer Science, vol. 13940, pp. 500–529. Springer (2023)
15. Cohen, B., Pietrzak, K.: Simple proofs of sequential work. In: EUROCRYPT (2). Lecture Notes in Computer Science, vol. 10821, pp. 451–467. Springer (2018)
16. Cohen, R., Garay, J.A., Zikas, V.: Completeness theorems for adaptively secure broadcast. In: CRYPTO (1). Lecture Notes in Computer Science, vol. 14081, pp. 3–38. Springer (2023)
17. Dachman-Soled, D., Jain, A., Kalai, Y.T., Lopez-Alt, A.: On the (in)security of the fiat-shamir paradigm, revisited. Cryptology ePrint Archive, Paper 2012/706 (2012), `https://eprint.iacr.org/2012/706`, `https://eprint.iacr.org/2012/706`
18. Degwekar, A., Vaikuntanathan, V., Vasudevan, P.N.: Fine-grained cryptography. In: CRYPTO (3). LNCS, vol. 9816, pp. 533–562. Springer (2016)
19. Döttling, N., Garg, S., Malavolta, G., Vasudevan, P.N.: Tight verifiable delay functions. In: SCN. Lecture Notes in Computer Science, vol. 12238, pp. 65–84. Springer (2020)
20. Egashira, S., Wang, Y., Tanaka, K.: Fine-grained cryptography revisited. In: ASIACRYPT (3). LNCS, vol. 11923, pp. 637–666. Springer (2019)
21. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: CRYPTO. Lecture Notes in Computer Science, vol. 263, pp. 186–194. Springer (1986)
22. Freitag, C., Komargodski, I., Pass, R., Sirkin, N.: Non-malleable time-lock puzzles and applications. In: TCC (3). Lecture Notes in Computer Science, vol. 13044, pp. 447–479. Springer (2021)
23. Goldreich, O.: Foundations of Cryptography: Volume 2, Basic Applications. Cambridge University Press, USA, 1st edn. (2009)

24. Goldwasser, S., Kalai, Y.T.: On the (in)security of the fiat-shamir paradigm. In: FOCS. pp. 102–113. IEEE Computer Society (2003)
25. Håstad, J., Impagliazzo, R., Levin, L.A., Luby, M.: A pseudorandom generator from any one-way function. SIAM J. Comput. **28**(4), 1364–1396 (1999)
26. Katz, J., Loss, J., Xu, J.: On the security of time-lock puzzles and timed commitments. In: TCC (3). LNCS, vol. 12552, pp. 390–413. Springer (2020)
27. LaVigne, R., Lincoln, A., Williams, V.V.: Public-key cryptography in the fine-grained setting. In: CRYPTO (3). Lecture Notes in Computer Science, vol. 11694, pp. 605–635. Springer (2019)
28. Lindell, Y.: How to simulate it - A tutorial on the simulation proof technique. In: Tutorials on the Foundations of Cryptography, pp. 277–346. Springer (2017)
29. Mahmoody, M., Moran, T., Vadhan, S.P.: Time-lock puzzles in the random oracle model. In: CRYPTO. LNCS, vol. 6841, pp. 39–50. Springer (2011)
30. Mahmoody, M., Smith, C., Wu, D.J.: Can verifiable delay functions be based on random oracles? In: ICALP. LIPIcs, vol. 168, pp. 83:1–83:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2020)
31. Malavolta, G., Thyagarajan, S.A.K.: Homomorphic time-lock puzzles and applications. In: CRYPTO (1). LNCS, vol. 11692, pp. 620–649. Springer (2019)
32. Milgrom, P.: Putting auction theory to work: The simultaneous ascending auction. Journal of political economy **108**(2), 245–272 (2000)
33. Naor, M.: On cryptographic assumptions and challenges. In: CRYPTO. Lecture Notes in Computer Science, vol. 2729, pp. 96–109. Springer (2003)
34. Pietrzak, K.: Simple verifiable delay functions. In: ITCS. LIPIcs, vol. 124, pp. 60:1–60:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2019)
35. Rivest, R.L., Shamir, A., Wagner, D.A.: Time-lock puzzles and timed-release crypto. Tech. rep. (1996)
36. Rotem, L., Segev, G.: Generically speeding-up repeated squaring is equivalent to factoring: Sharp thresholds for all generic-ring delay functions. In: CRYPTO (3). Lecture Notes in Computer Science, vol. 12172, pp. 481–509. Springer (2020)
37. Vadhan, S.P., Zheng, C.J.: Characterizing pseudoentropy and simplifying pseudorandom generator constructions. In: STOC. pp. 817–836. ACM (2012)
38. Wan, J., Xiao, H., Devadas, S., Shi, E.: Round-efficient byzantine broadcast under strongly adaptive and majority corruptions. In: TCC (1). Lecture Notes in Computer Science, vol. 12550, pp. 412–456. Springer (2020)
39. Wesolowski, B.: Efficient verifiable delay functions. In: EUROCRYPT (3). Lecture Notes in Computer Science, vol. 11478, pp. 379–407. Springer (2019)

A Model for Depth-Secure MPC

This appendix is an extension of Section 5. Here, we discuss in more detail the execution of the ideal model.

A.1 Execution Model

Our execution model is based on a simpler version of the Universal Composability (UC) framework, modified for our application scenario and depth-bounded computation. In our execution model, all parties (including the environment, trusted third party, and adversary) are modeled as interactive circuits.

The Environment: As in the UC framework, we consider an execution in the presence of an *environment* that provides inputs to parties and reads their outputs. The environment directs the execution by proceeding in rounds. It delivers inputs to parties, activates each party in every round, and delivers messages. The environment controls the time elapse of an execution via the number of protocol rounds it has directed. Importantly, the environment is responsible for directing query responses between interactive circuits. When the environment activates a party, it evaluates one next-step circuit at a time, after which control is returned to the environment. The environment also ensures that the queries made by a party in one round are delivered to the intended oracles (or parties, if oracle queries are used to communicate).

When the adversary is activated, it learns the corrupt parties' inputs, the queries they send, and the responses they receive. In the beginning of the execution, the adversary informs the environment of the identities of the parties it wishes to corrupt. The environment responds with the corrupt parties' inputs, and the adversary may choose new inputs for the corrupt parties based on the provided inputs and its auxiliary information. (This models the fact that inputs for corrupted parties may be adversarially selected, which is in the application scenario of accountable computation.)

As the execution proceeds, the environment activates the adversary after activating other parties, informing the adversary of the queries the corrupt parties make and the responses they receive. The adversary can respond to the environment by making additional queries. (This structure allows the adversary and environment to pass additional messages.) The adversary can also adaptively choose to corrupt additional parties by passing an appropriate query to the environment.

Defining a View: The *view* of any party is defined to be the ordered list of inputs and events it receives from the environment, along with the ordered list of messages it receives from other parties. Formally, we denote the view of party i in an execution of protocol π on inputs $\vec{x}$ and security parameter 1^λ as $\mathsf{View}_i^\pi(\vec{x}, 1^\lambda) = (x_i; r; \vec{m})$, where x_i is party i's input, r is the party's randomness, and $\vec{m}$ is the set of messages that party i receives from other parties and the environment.

A.2 The Ideal/Real Paradigm

Execution in the Ideal Model. We define an ideal model in which parties interact with a trusted third party in an execution that is secure by definition.

Interaction with the Trusted Party In an ideal execution, the parties interact with a trusted party as follows:

1. **Initialization:** The adversary $\mathcal{A}$ receives an auxiliary input z, and may choose to corrupt some parties. It informs $\mathcal{Z}$ of the corruptions.

2. **Inputs:** The environment sends the corrupt parties' inputs to $\mathcal{A}$, which choose new inputs for the corrupted parties based on its auxiliary information and the inputs provided by the environment. It then forwards the new inputs to the environment. All parties then receive inputs from the environment.
3. **Send Inputs to Trusted Party:** Each party sends its input x_i to the trusted party.
4. **Computing Functionalities:** After receiving all inputs, the trusted third party computes the functionality outputs over the provided inputs and saves the outputs.
5. **Phased Output Release:** An execution is divided into *phases* such that at the end of each phase, the parties learn some information from the trusted party. The moment that the trusted party provides the protocol participants with their ith message denotes the end of the ith phase and the beginning of the $i+1$st phase.
6. **Protocol Outputs:** At the end of an execution, honest parties output whatever they have received from the trusted party. Corrupt parties output nothing, and the adversary outputs an arbitrary function of its input, the messages it has received from the environment, and the messages that corrupt parties have received from the trusted party. The environment learns every output.

The random variable $\mathsf{IDEAL}_{\mathcal{F},\mathcal{A}(z),\mathcal{Z}}(\overline{x})$ denotes the output of the environment in an *ideal execution* of functionality $\mathcal{F}$ on honest inputs $\overline{x}$, auxiliary input z to $\mathcal{A}$, with environment $\mathcal{Z}$.

Execution in the Real Model. In the real model, participants execute a protocol π to compute the desired functionality $\mathcal{F}$ without a trusted party. At the end of the execution, honest parties output their protocol outputs. The corrupt parties output nothing. The adversary outputs an arbitrary function of its inputs and the messages that corrupt parties have received.

The random variable $\mathsf{REAL}_{\pi,\mathcal{A}(z),\mathcal{Z}}(\overline{x})$ denotes the output of the environment in a real execution of π with honest inputs $\overline{x}$, auxiliary input z to $\mathcal{A}$, with environment $\mathcal{Z}$. The environment learns every output.

B Sequential Composition of Depth-Secure Protocols: Proof of Theorem 4

In this section, we provide the full proof of Theorem 4, which we restate below for convenience.

Theorem 4 (Sequential Composition of Two Depth-Secure Protocols). *Let π (d_a, d_s, d_e)-depth-securely compute F in the G-hybrid model, and let ρ (d'_a, d'_s, d'_e)-depth-securely compute G. π^ρ $(d_a - d'_s, d_s \cdot d'_s, \min(d_e, d'_e))$-depth-securely computes F.*

Notation. For the proof of Theorem 4, we require notation to describe the distribution of executions in the ideal world for a fixed simulator, fixed distinguisher, and fixed inputs, making explicit the adversary. Let $\mathsf{IDEAL}_{\mathcal{F},\mathcal{S}(z)}(\overline{x})$ denote the distribution of executions of the naive protocol in the ideal world that calls functionality $\mathcal{F}$, with simulator $\mathcal{S}$ and advice string z, on honest inputs $\overline{x}$. (In this experiment, the parties forward their inputs the ideal functionality, and the simulator generates a view for $\mathcal{A}$ that is indistinguishable from the real experiment.)

Proof. The proof will use the simulators $\mathcal{S}^{\pi}$ for π and $\mathcal{S}^{\rho}$ for ρ to construct a new simulator $\mathcal{S}$ for π^{ρ} such that $\mathcal{S}$ is $(d_s \cdot d'_s)$-depth bounded, and for every $(d_a - d'_s)$-depth $\mathcal{A}$, and every $\min(d_e, d'_e)$-depth $\mathcal{Z}$, the distributions $\mathsf{REAL}_{\pi^{\rho},\mathcal{A}(z)}(\overline{x})$ and $\mathsf{IDEAL}_{\mathcal{F},\mathcal{S}(z)}(\overline{x})$ are $\min(d_e, d'_e)$-depth indistinguishable.

The simulator $\mathcal{S}$ works by composing the simulators $\mathcal{S}^{\pi}$ and $\mathcal{S}^{\rho}$. Specifically, to simulate an execution of π^{ρ} up to the point that ρ is called, $\mathcal{S}$ runs $\mathcal{S}^{\pi}$. When ρ is called, $\mathcal{S}$ invokes $\mathcal{S}^{\rho}$. After ρ terminates, $\mathcal{S}$ resumes $\mathcal{S}^{\pi}$.

Claim 4 *$\mathcal{S}$'s depth is bounded by $d_s \cdot d'_s$.*

Proof. The claim follows from the observation that every time $\mathcal{S}^{\pi}$ is rewound, $\mathcal{S}^{\rho}$ must also be rewound the maximum number of times. If $\mathcal{S}^{\pi}$'s running time is at most d_s, then for each rewinding of $\mathcal{S}^{\pi}$, $\mathcal{S}^{\rho}$ must be rewound at most d'_s times. The total run-time of $\mathcal{S}$ is thus $d_s \cdot d'_s$.

We proceed with our main lemma, which completes the proof:

Lemma 3. *For every $(d_a - d'_s)$-depth adversary $\mathcal{A}$, and every $\overline{x} \in (\{0,1\}^{\mathsf{poly}(\lambda)})^n$ and $z \in \{0,1\}^{\mathsf{poly}(\lambda)}$ the distributions $\mathsf{REAL}_{\pi^{\rho},\mathcal{A}(z)}(\overline{x})$ and $\mathsf{IDEAL}_{\mathcal{F},\mathcal{S}(z)}(\overline{x})$ are $\min(d_e, d'_e)$-depth indistinguishable.*

Proof Sketch: If there is an adversary $\mathcal{A}$ and a distinguisher $\mathcal{D}$ that distinguishes the above two distributions, then we create another adversary $\mathcal{B}$ and distinguisher E that isolates an attack against the caller protocol π in the G-hybrid model. $\mathcal{B}$ runs $\mathcal{A}$ as a black box, and when π must call ρ, $\mathcal{B}$ simply simulates an execution of ρ (using $\mathcal{S}^{\rho}$), feeding messages to $\mathcal{A}$ so that $\mathcal{A}$ believes it is running a full execution of π^{ρ}. Similarly, E is provided with the execution transcript generated by $\mathcal{B}$, with the call to ρ in the transcript replaced by the simulated output generated by $\mathcal{B}$. Because the transcript of the simulation of ρ is indistinguishable from a real execution by assumption, this attack must distinguish an execution of π in the real model from its simulation, contradicting the security of π.

Proof. Assume to the contrary that the lemma statement is false. Then there exists a $(d_a - d'_s)$-depth adversary $\mathcal{A}$, a $\min(d_e, d'_e)$-depth distinguisher $\mathcal{D}$, and inputs $\overline{x}, z$ such that the distributions $\mathsf{REAL}_{\pi^{\rho},\mathcal{A}(z)}(\overline{x})$ and $\mathsf{IDEAL}_{\mathcal{F},\mathcal{S}(z)}(\overline{x})$ are $\min(d_e, d'_e)$-depth distinguishable (for any $(d_s \cdot d'_s)$-depth $\mathcal{S}$).

We will show how to use $\mathcal{A}$ for π^{ρ} in order to build an adversary $\mathcal{B}$ to contradict the (d_a, d_s, d_e)-security of π in the G-hybrid model.

In an execution of π in the G-hybrid model, $\mathcal{B}$ works as follows:

1. Until the point at which G is invoked, $\mathcal{B}$ runs $\mathcal{A}$ as a black box, forwarding any messages output by $\mathcal{A}$
2. When G is invoked, $\mathcal{B}$ submits its input y to G and receives some output w. $\mathcal{B}$ runs the simulator $\mathcal{S}^\rho(y, w)$ for ρ, forwarding messages provided by the simulator to $\mathcal{A}$, and forwarding the replies by $\mathcal{A}$ to $\mathcal{S}^\rho$ to continue the simulation.
3. After $\mathcal{S}^\rho$ terminates, $\mathcal{B}$ resumes calling $\mathcal{A}$ as a black box given messages from its execution of π. $\mathcal{B}$ outputs whatever $\mathcal{A}$ outputs.

Claim 5 *$\mathcal{B}$ runs in depth at most d_a.*

Proof. $\mathcal{B}$ runs the adversary $\mathcal{A}$ as a black box, which requires depth at most $d_a - d'_s$. $\mathcal{B}$ also runs the simulator $\mathcal{S}^\rho$, which requires depth at most d'_s. (Recall that we have already counted the depth of rewinding $\mathcal{A}$ during this step towards the depth d'_s.) The sum of the two run-times is $d_a - d'_s + d'_s = d_a$ which concludes the claim.

We proceed to compare the views of the adversary $\mathcal{A}$ when it is running in its own execution, or being called by $\mathcal{B}$. Let $\mathsf{VIEW}_\mathcal{A}(\mathsf{REAL}_{\pi^\rho,\mathcal{A}(z)}(\overline{x}))$ denote the view of $\mathcal{A}$ in a real execution of π^ρ, and let $\mathsf{VIEW}_\mathcal{A}(\mathsf{REAL}_{\pi^G,\mathcal{A}(z)}(\overline{x}))$ denote the view of $\mathcal{A}$ in a real execution of π in the G-hybrid model, in which $\mathcal{B}$ calls $\mathcal{A}$. Similarly, we denote by $\mathsf{VIEW}_\mathcal{A}(\mathsf{IDEAL}^\mathcal{B}_{\mathcal{F},\mathcal{S}(z)}(\overline{x}))$ the view of $\mathcal{A}$ in support of the ideal experiment in which $\mathcal{B}$ calls $\mathcal{A}$, and $\mathcal{S}$ runs both the simulators for π and for ρ; and we denote by $\mathsf{VIEW}_\mathcal{A}(\mathsf{IDEAL}^\mathcal{B}_{\mathcal{F},\mathcal{S}^\pi(z)}(\overline{x}))$ the view of $\mathcal{A}$ in support of the ideal experiment in which $\mathcal{B}$ must call the simulator for ρ.

Claim 6 *Let $f' = \min(d_e, d'_e)$. For all $\overline{x} \in (\{0,1\}^{\mathsf{poly}(\lambda)})^n$ and $z \in \{0,1\}^{\mathsf{poly}(\lambda)}$*

$$\mathsf{VIEW}_\mathcal{A}(\mathsf{REAL}_{\pi^\rho,\mathcal{A}(z)}(\overline{x})) \overset{f'}{\approx} \mathsf{VIEW}_\mathcal{A}(\mathsf{REAL}_{\pi^G,\mathcal{A}(z)}(\overline{x})\})$$

Proof. The difference between the two distributions is that on the right, $\mathcal{B}$ simulates an execution of ρ using the simulator $\mathcal{S}^\rho$ and provides those messages to $\mathcal{A}$, and then continues to call $\mathcal{A}$ after the call to $\mathcal{S}^\rho$ using messages from its real execution. By assumption, $\mathcal{A}$ is $(d_a - d'_s)$-depth-bounded and $d_a < d'_e$. Therefore, $\mathcal{A}$ must not be able to distinguish the messages in the real execution of ρ on the left from the simulation on the right. The claim follows from the additional fact that all other messages in $\mathcal{A}$'s view are distributed identically in both experiments, since they are from the real execution of π.

We make another claim that $\mathcal{A}$ cannot distinguish between an idealized execution of $\mathcal{F}$ in which $\mathcal{S}$ generates its view of the execution and an idealized execution of $\mathcal{F}$ in which $\mathcal{B}$ interacts with $\mathcal{S}^\pi$ in the G-hybrid model, forwarding its messages to $\mathcal{A}$ and when $\mathcal{B}$'s execution of in the G-hybrid model invokes G, $\mathcal{B}$ runs $\mathcal{S}^\rho$ to generate a view for $\mathcal{A}$.

Claim 7 *For all $\overline{x} \in (\{0,1\}^{\mathsf{poly}(\lambda)})^n$ and $z \in \{0,1\}^{\mathsf{poly}(\lambda)}$*

$$\mathsf{VIEW}_\mathcal{A}(\mathsf{IDEAL}^\mathcal{B}_{\mathcal{F},\mathcal{S}(z)}(\overline{x})) \equiv \mathsf{VIEW}_\mathcal{A}(\mathsf{IDEAL}^\mathcal{B}_{\mathcal{F},\mathcal{S}^\pi(z)}(\overline{x}))$$

Proof. The proof is analogous to the previous. However, in this case, $\mathcal{B}$ perfectly simulates the execution of ρ in comparison to $\mathcal{A}$'s view in the ideal execution of π^ρ, since $\mathcal{B}$ does exactly the same thing that $\mathcal{S}$ does: both run $\mathcal{S}^\rho$.

To complete the proof, we describe how the distinguisher E is built from D. E simply runs D as a black box and outputs whatever D outputs.

Next we claim that D's view in support of $\mathsf{REAL}_{\pi^\rho,\mathcal{A}(z)}(\overline{x})$ is $\min(d_e, d'_e)$-depth indistinguishable from its view in support of $\mathsf{REAL}_{\pi,\mathcal{B}(z)}(\overline{x})$ (as forwarded by E). This follows from Claim 6, due to the fact that $\mathcal{A}$'s views in support of the two distributions are $\min(d_e, d'_e)$-depth indistinguishable, and $\mathcal{D}$ sees the transcript of $\mathcal{A}$'s interaction with the real protocol, and $\mathcal{A}$'s outputs must not be distinguishable by the claim.

Similarly, $\mathcal{D}$'s view in support of $\mathsf{IDEAL}^{\mathcal{A}}_{\mathcal{F},\mathcal{S}(z),D}(\overline{x})$ is $\min(d_e, d'_e)$-depth indistinguishable from its view in support of $\mathsf{IDEAL}^{\mathcal{B}}_{\mathcal{F},\mathcal{S}^\pi(z),E}(\overline{x})$ (as forwarded by E). This follows from Claim 7, via the same argument as above.

It follows that if D distinguishes $\mathsf{REAL}_{\pi^\rho,\mathcal{A}(z),D}(\overline{x})$ and $\mathsf{IDEAL}^{\mathcal{A}}_{\mathcal{F},\mathcal{S}(z),D}(\overline{x})$, then E distinguishes $\mathsf{REAL}_{\pi,\mathcal{B}(z),E}(\overline{x})$ and $\mathsf{IDEAL}^{\mathcal{B}}_{\mathcal{F},\mathcal{S}^\pi(z),E}(\overline{x})$. Notice that because $\mathcal{B}$'s depth is bounded by d_a (by Claim 5), and because E's depth is bounded by $\min(d_e, d'_e)$ (by assumption toward contradiction, since E's depth is exactly D's depth), this contradicts the (d_a, d_s, d_e)-depth security of π in the G-hybrid model.

□

C Residual Complexity of a Time-Lock Puzzle

The qualification of a *time-lock* puzzle tells us that for any circuit $\mathcal{A}_d$ attempting to solve a puzzle for which d is much less than the depth required by $\mathsf{Puz.Solve}$, the probability of guessing the solution should be no better than random guessing plus negligible advantage. However, a circuit $\mathcal{A}_d$ whose depth d exceeds t^ε (as enforced in the definition) may have non-negligible advantage in guessing the solution. Therefore, a time-lock puzzle constrains the residual complexity function r of the puzzle to remain small for as long as the time-lock endures. We now formally prove this intuition.

Theorem 8 (Time-Lock Puzzle Implies Small Residual Complexity). *Let* $\mathsf{Puz} = (\mathsf{Puz.Gen}, \mathsf{Puz.Solve})$ *be a time-lock puzzle for solution domain* $M = \{\chi_\lambda\}_\lambda$ *with gap* $\varepsilon < 1$ *for which* $|M_\lambda|$ *is super-polynomial in* λ*. Then there exists a polynomial* $r(\cdot)$ *for which for every polynomial* $t(\cdot) > r(\cdot)$ *and* t^ε*-depth-bounded* $\mathcal{A}_t$*, there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that for every* t^ε*-depth-bounded* $\mathcal{B}$*, and every* $\lambda \in \mathbb{N}$

$$\Pr[\chi \leftarrow \mathcal{B}(Y) \colon \chi \leftarrow M_\lambda, Y \leftarrow \mathsf{Puz.Gen}(\lambda, \chi)] \leq \mathsf{negl}(\lambda)$$

Proof. We prove the lemma by showing that if there exists an adversary $\mathcal{B}_t$ for which $\Pr[\chi \leftarrow \mathcal{B}(Y, z)] > \mathsf{negl}(\lambda)$, then there exists an adversary $\mathcal{A}_t$, infinitely many λ and corresponding solutions $\chi_0, \chi_1 \in M_\lambda$ such that $\mathcal{A}_\lambda$ can win the time-lock challenge with probability more than $\frac{1}{2} + \mathsf{negl}(\lambda)$. For the sake of the proof, let $r > \mathsf{negl}(\lambda)$ be the probability with which $\mathcal{B}$ outputs χ in the above challenge game.

Actually, we will show a result corresponding to a stronger statement. We show that if there exists an adversary $\mathcal{B}$ that wins the above challenge game with non-negligible advantage, then there exists an adversary $\mathcal{A}$ such that for every χ_0 there are many solutions χ_1 such that $\mathcal{A}_\lambda$ can win the time-lock challenge with probability more than $\frac{1}{2} + \mathsf{negl}(\lambda)$. Our time-lock game is slightly weaker than the definition of a time-lock puzzle (and therefore if our time-lock game is broken, the puzzle is not a time-lock puzzle). Rather than quantifying over all χ_0 and χ_1, we allow the adversary $\mathcal{A}$ to choose any $\chi_0, \chi_1 \in \chi_\lambda$ and provide them to a challenger, who samples b and provides $\mathcal{A}$ with a puzzle. $\mathcal{A}$ must guess b' and wins if $b' = b$.

We now explain how $\mathcal{A}$ uses $\mathcal{B}$. Recall that $\mathcal{B}$ is given a randomly sampled puzzle and outputs a guess χ' of the solution. In the time-lock game, $\mathcal{A}$ samples χ_0 and χ_1 at random and must determine which one has been encoded in a challenge puzzle Z. $\mathcal{A}$ forwards Z to $\mathcal{B}$. At the end, $\mathcal{A}$ inspects the guess χ' that $\mathcal{B}$ makes. If χ' is equal to either χ_0 or χ_1, then $\mathcal{A}$ guesses the b for which $\chi_b = \chi'$. If neither χ_0 nor χ_1 is guessed by $\mathcal{B}$, then $\mathcal{A}$ samples b' uniformly at random and outputs b'. Note that the depth of $\mathcal{A}$ is the same as $\mathcal{B}$.

Recall that $\mathcal{B}$ wins its game with probability r, and that by assumption r is non-negligble. We now analyze the probability with which $\mathcal{A}$ wins its game.

Claim. $\Pr[\chi' \in \{\chi_0, \chi_1\}] \geq \Pr[\mathcal{B} \text{ wins}] > \mathsf{negl}(\lambda)$

Proof. Follows immediately from the definition that $\mathcal{B}$ wins when it guesses the solution, and by assumption that $\mathcal{B}$ wins with non-negligible probability.

Claim. $\Pr[\chi' = \chi_{1-b}] = \mathsf{negl}(\lambda)$

Proof. Consider that χ_{1-b} is selected at random by $\mathcal{A}$, and $\mathcal{B}$ has no information about χ_{1-b}. Recall that conditioned on the fact that $\mathcal{B}$ guesses some possible solution with non-negligible probability (the true solution), and let X be the part of the solution space for which $\mathcal{B}$ outputs solutions in X with non-negligible probability. Let Y be the part of the solution space for which $\mathcal{B}$ guesses solutions with negligible probability. We claim that X composes a negligible proportion of the solution space, and that therefore χ_{1-b} is in Y except for negligible probability. The proof proceeds by counting. For all of the points in X, $\mathcal{B}$ must guess each point with probability at least the inverse of some polynomial. It follows that there may only be a polynomial number of points in X. However, there are a super-polynomial number of points in the solution space. Therefore, the probability that χ_{1-b} is in Y is overwhelming. And by definition of Y, the probability that $\mathcal{B}$ guesses χ_{1-b} is negligible.

It follows from the previous claim that conditioned on $\mathcal{B}$ outputting χ_0 or χ_1, $\mathcal{A}$ wins with probability $1 - \mathsf{negl}(\lambda)$.

Claim. $\Pr[\mathcal{A} \text{ wins} \mid \chi' \in \{\chi_0, \chi_1\}] = 1 - \mathsf{negl}(\lambda)$

Proof. The probability that $\mathcal{A}$ wins given that one of the solutions output by $\mathcal{B}$ is divided into cases:

1. $\chi' = \chi_{1-b}$. $\mathcal{A}$ loses
2. $\chi' = \chi_b$. $\mathcal{A}$ wins

By the previous claim, the probability of the first event is $\mathsf{negl}(\lambda)$. In the remaining case, $\mathcal{A}$ wins. Note that because the second case with non-negligible probability, this case dominates, as the other composes a negligible proportion of the event space. It follows that $\mathcal{A}$ wins with probability $1 - \mathsf{negl}(\lambda)$ given $\mathcal{B}$ outputs either χ_0 or χ_1.

We can now conclude the proof:

$$\begin{aligned}
\Pr[\mathcal{A} \text{ wins}] &= \Pr[\mathcal{A} \text{ wins } \mid \chi' \in \{\chi_0, \chi_1\}] \Pr[\chi' \in \{\chi_0, \chi_1\}] \\
&\quad + \Pr[\mathcal{A} \text{ wins} \mid \chi' \notin \{\chi_0, \chi_1\}] \Pr[\chi' \notin \{\chi_0, \chi_1\}] \\
&= (1 - \mathsf{negl}(\lambda)) \Pr[\chi' \in \{\chi_0, \chi_1\}] + \frac{1}{2} \Pr[\chi' \notin \{\chi_0, \chi_1\}]
\end{aligned}$$

Recall that the two events $\chi' \in \{\chi_0, \chi_1\}$ and $\chi' \notin \{\chi_0, \chi_1\}$ are complements. Therefore, if $\Pr[\chi' \in \{\chi_0, \chi_1\}] > \mathsf{negl}(\lambda)$, then $\Pr[\mathcal{A} \text{ wins}] > \frac{1}{2} + \mathsf{negl}(\lambda)$. The proof concludes by the first claim, which states that $\Pr[\chi' \in \{\chi_0, \chi_1\}] \geq \Pr[\mathcal{B} \text{ wins}] > \mathsf{negl}(\lambda)$.

D Extended Related Work

We present additional related work in the areas of fine-grained cryptography and composable timed primitives.

Fine-grained Cryptography: A number of recent works have studied fine-grained cryptographic primitives. Degwekar et al.[18] initiated the study of fine-grained cryptographic primitives that can be built in one complexity class and are secure against adversaries in larger complexity classes. Egashira et al.[20] recently extended their results. Ball et al.[3] and [4] built fine-grained proofs-of-work by using fine-grained worst-case to average-case reductions of hard problems. Lavigne et al.[27] studied the properties necessary to imply fine-grained public key cryptography and presented a fine-grained key exchange protocol.

Homomorphic Time-lock Puzzles: Malavolta and Thyagarajan [31] provided practical homomorphic time-lock puzzles that are either additively homomorphic, multiplicatively homomorphic, or branching programs, but they require indistinguishability obfuscation in order to achieve full homomorphism. They also do not consider composition of their puzzles with other cryptographic primitives.

Time-lock Cryptography and Composition: We now provide a more thorough contrast with the approach of Baum et al.[6, 5].

The model by Baum et al.[6] models a new abstraction of time by allowing the adversary to control ticks of some time-keeping functionality. They define a time-lock functionality that implements the assumption by RSW [35], and provide a protocol that builds a puzzle with respect to this functionality. The functionality implements an idealized version of their assumption which does not leak information until the time-lock expires. In contrast, we model the leakage of puzzles that occurs in the transition from not knowing to knowing the solution. Moreover, in our model time is modeled by depth of computation, and is therefore not controlled by the adversary. Wall-clock time is controlled by the environment which upper bounds the compute depth that may be expended over any period of time. To enforce time-based privacy, we model idealized leaky functionalities that respond to environment-directed time. With respect to our functionality, we show how to simulate an adversary's view as it slowly extracts information from a time-lock puzzle.

The central issue for Baum's approach is a "side-door" attack in which an environment may use cycles from the concurrent execution of a different session in order to solve a TP in given session. Our approach considers this particular attack to be infeasible. All parties in our model are depth-bounded, including the environment. In our model, an environment should be constrained by the same depth requirements among all of its concurrent executions. An environment that expends computational resources in a concurrent session in order to solve a TP must also expend the same depth in the session of a given time-lock protocol; therefore, although the environment may increase its parallel computation to solve a puzzle by invoking concurrent sessions, the depth constraint remains. Therefore, our depth-bounded model specifically excludes this form of attack.

E Security of π^{auction}

In this section we present the full proof of Theorem 6. To set up the proof, we first define notions of non-malleability:

E.1 Non-malleability

We adapt definitions from [22] to our model; for full discussion of the definitions, refer to [22]. We prove our theorem with respect to these definitions; however, we provide a note after the proof of how the same proof can be easily adapted for the use of non-malleable timed commitments, as studied by [26] and [14].

Definition 10 (MIM Adversary).

Let $n_L, n_R, B_{nm}, d_a : \mathbb{N} \to \mathbb{N}$. An (n_L, n_R, B_{nm}, d_a)-Man-in-the-Middle (MIM) adversary is a non-uniform algorithm $\mathcal{A} = \{\mathcal{A}_\lambda\}_{\lambda\in\mathbb{N}}$ satisfying $\mathsf{depth}(\mathcal{A}_\lambda) \leq d_a(\lambda)$ and $\mathsf{size}(\mathcal{A}_\lambda) \in B_{nm} \cdot \mathsf{poly}(\lambda)$ for all $\lambda \in \mathbb{N}$ that receives $n_L(\lambda)$ puzzles on the left and outputs $n_R(\lambda)$ on the right.

Definition 11 (MIM Distribution). *Let $n_L, n_R, B_{nm}, d_a : \mathbb{N} \to \mathbb{N}$. Let $\mathcal{A} = \{\mathcal{A}_\lambda\}_{\lambda\in\mathbb{N}}$ be an (n_L, n_R, B_{nm}, d_a)-MIM adversary. For any $\lambda, \tau \in \mathbb{N}$ and $\vec{s} = (s_1, \ldots, s_{n_L(\lambda)}) \in (\{0,1\}^\lambda)^{n_L(\lambda)}$, we define the distribution*

$$(\tilde{s_1}, \ldots, \tilde{s_{n_R(\lambda)}}) \leftarrow \mathsf{mim}_{\mathcal{A}}(\tau, \vec{s})$$

as follows. $\mathcal{A}_\lambda$ receives puzzles $z_i \leftarrow \mathsf{Gen}(\tau, s_i)$ for all $i \in [n_L(\lambda)]$ and outputs puzzles $(\tilde{z_1}, \ldots, \tilde{z_{n_R(\lambda)}})$. Then for each $i \in [n_R(\lambda)]$ we define

$$\tilde{s_i} = \begin{cases} \perp & \text{if there exists a } j \in [n_L(\lambda)] \text{such that } \tilde{z_i} = z_j \\ \mathsf{Solve}(\tilde{z_i}) & \text{otherwise} \end{cases}$$

Definition 12 (Concurrent Non-malleable). *Let $n_L, n_R, B_{nm} \colon \mathbb{N} \to \mathbb{N}$. A time-lock puzzle is (n_L, n_R, d_a, d_e) concurrent non-malleable against adversaries of size B_{nm} if for all $\lambda \in \mathbb{N}$, and every (n_L, n_R, B_{nm}, d_a)-MIM adversary $\mathcal{A} = \{\mathcal{A}_\lambda\}_{\lambda\in\mathbb{N}}$, the following holds.*

For any non-uniform distinguisher $D = \{D_\lambda\}_\lambda$ with depth at most $d_e(\lambda)$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that for all $\lambda \in \mathbb{N}$, $\vec{s} = (s_1, \ldots, s_{n_L(\lambda)}) \in (\{0,1\}^\lambda)^{n_L(\lambda)}$, and $\tau < d_e(\lambda)$:

$$|\Pr[D(\mathsf{mim}_{\mathcal{A}}(\tau, \vec{s}) = 1] - \Pr[D(\mathsf{mim}_{\mathcal{A}}(\tau, (0^\lambda)^{n_L(\lambda)}) = 1]| \leq \mathsf{negl}(\lambda)$$

E.2 Proof

In the following theorem, we require that $d_a(\lambda) \leq t^{\mathsf{bid}}$. Let n_L denote the number of honest parties in the execution, and let n_R denote the number of parties corrupted by the adversary. We require that the puzzles are (n_L, n_R, d_a, d_e)-non-malleable.

Theorem 6 (Security of π^{auction}). *Let Puz be an equivocable, non-malleable time-lock puzzle scheme. Then π^{auction} $(t^{\mathsf{bid}}, d_s, d_e)$-securely implements $\mathcal{F}_{\mathsf{auction}}$, where $d_s(\lambda) = \mathsf{depth}(\mathsf{Puz.Gen}(\lambda)) + t^{\mathsf{end}}(\lambda) - t^{\mathsf{bid}}(\lambda)$ and d_e is an arbitrary polynomial in the security parameter λ.*

Proof. First note that assuming that t^{bid} is set to be no later than t^* for Puz (for appropriate security parameters), the adversary's probability of learning any honest party's puzzle solution is small (in the security parameter), and sufficiently small (in the security parameter) to disregard in the proof. This follows from a union bound over the analyses of the adversary's ability to attack any individual puzzle. We proceed with the proof under the assumption that the adversary does not learn the solutions of any puzzle provided to it by the honest parties (or by the simulator in place of the honest parties).

For the sake of simplicity, we also assume that all honest parties send their puzzles *at the beginning* of the protocol, and do not wait until almost t^{bid} to submit their puzzles. This simplifies both the behavior of the simulator and the reduction (and is the proper behavior for an honest party, even though puzzles are accepted until t^{bid}). Otherwise, the simulator must be changed so that it provides puzzles to the adversary at different times. The reduction at the end of the proof must also be changed so that the adversary attacking non-malleability of the puzzles is not permitted its full depth to work on some of the puzzles.

In the execution simulated by $\mathcal{S}$, $\mathcal{S}$ replaces the time-lock puzzles by non-corrupt parties with equivocable time-lock puzzles with solutions 0^λ. $\mathcal{S}$ requires $\mathsf{depth}(\mathsf{Puz.Gen})$ depth in order to generate up to n puzzles in parallel. When $\mathcal{S}$ learns the honest parties' puzzle solutions after t^{bid}, it equivocates the solution as discussed in Section 7.2.

Note that the time-lock puzzles used to submit bids are the only messages sent in an execution; therefore, these puzzles constitute the entire view of an execution.

What remains to show is that no distinguisher can distinguish the distributions of puzzles output by the adversary between the cases that the adversary is fed puzzles from the real execution or puzzles generated by the simulator. This serves to show as well that the adversary cannot generate puzzles that *depend* on the honest parties' puzzles, and that moreover the adversary is not a distinguisher for the distribution of puzzles generated by $\mathcal{S}$.

For this we reduce to the *non-malleability* (Definition 12) of the puzzles. Let there be a distinguisher $\mathcal{D}$ that distinguishes the puzzles output by $\mathcal{A}$ in the real execution from the puzzles output by $\mathcal{A}$ in the simulation. Further let $\mathcal{D}$ have depth d_e and $\mathcal{A}$ have depth d_a.

Then there exist adversaries $\mathcal{B}$,$\mathcal{E}$ with depths d_a, d_e, respectively, that breaks (n_L, n_R, d_a, d_e) concurrent non-malleability of the puzzles, where n_L is the number of honest parties and n_R is the number of corrupt parties. The reduction is straightforward. $\mathcal{B}$ simply runs the protocol for $\mathcal{A}$ and provides to $\mathcal{A}$ the puzzles that it receives in $\mathcal{B}$'s own challenge. $\mathcal{E}$ simply runs $\mathcal{D}$ and outputs whatever $\mathcal{D}$ outputs. Note that because the adversary's entire view in an execution of π^{auction} is only the honest party's puzzles, $\mathcal{B}$ does not need to do anything else to simulate an execution for $\mathcal{A}$. Here, the real-world puzzles correspond to when $\mathcal{B}$ is challenged with real puzzles, and the simulation puzzles correspond to when $\mathcal{B}$ is challenged with puzzles of 0^λ.

Now, $\mathcal{A}$'s inputs and outputs are identical to the MIM adversary in the non-malleability game, and $\mathcal{D}$ is the distinguisher. $\mathcal{B}$ and $\mathcal{E}$ simply output what $\mathcal{A}$ and $\mathcal{D}$ output. Therefore, it must be the case that $\mathcal{E}$ distinguishes with the same probability as $\mathcal{D}$. This contradicts (n_L, n_R, d_a, d_e) non-malleability.

Remark 4 (Reducing to CCA Security). The proof could analogously be written with a reduction to CCA security of timed commitments [26, 14], with a similarly structured reduction as the final step. Intuitively, if there is an adversary $\mathcal{A}$ that can distinguish between the simulator's time-lock puzzles and the honest parties' real puzzles, then some adversary $\mathcal{B}$ can use $\mathcal{A}$ to break the CCA security of the puzzles. $\mathcal{B}$ for the CCA game, which is either fed a real puzzle (from the honest parties) or a fake puzzle (from the simulator) simply feeds a puzzle to $\mathcal{A}$ and outputs whatever $\mathcal{A}$ outputs. The proof is slightly more involved because both [26, 14] present non-interactive timed commitments, which have additional algorithms and therefore different oracles, but these can straightforwardly be simulated by $\mathcal{B}$ (because it knows the secrets which are not part of the challenge), and $\mathcal{B}$ forwards the challenges to $\mathcal{A}$. Note that in the CCA games, the adversary is presented a single puzzle rather than a set of puzzles (which are provided in the protocol execution). This difference can be handled by a standard hybrid argument.