Standard Model Time-Lock Puzzles: Defining Security and Constructing via Composition

Karim Eldefrawy*, Sashidhar Jakkamsetti**†, Ben Terner**†,

and Moti Yung***

* SRI International
** University of California, Irvine
*** Google and Columbia University

Abstract. The introduction of time-lock puzzles initiated the study of publicly "sending information into the future." For time-lock puzzles, the underlying security-enabling mechanism is the computational complexity of the operations needed to solve the puzzle, which must be tunable to reveal the solution after a predetermined time, and not before that time. Time-lock puzzles are typically constructed via a commitment to a secret, paired with a reveal algorithm that sequentially iterates a basic function over such commitment. One then shows that short-cutting the iterative process violates cryptographic hardness of an underlying problem.

To date, and for more than twenty-five years, research on time-lock puzzles relied heavily on iteratively applying well-structured algebraic functions. However, despite the tradition of cryptography to reason about primitives in a realistic model with standard hardness assumptions (often after initial idealized assumptions), most analysis of time-lock puzzles to date still relies on cryptography modeled (in an ideal manner) as a random oracle function or a generic group function. Moreover, Mahmoody et al. showed that time-lock puzzles with superpolynomial gap cannot be constructed from random-oracles; yet still, current treatments generally use an algebraic trapdoor to efficiently construct a puzzle with a large time gap, and then apply the inconsistent (with respect to Mahmoody et al.) random-oracle idealizations to analyze the solving process. Finally, little attention has been paid to the nuances of composing multi-party computation with timed puzzles *that are solved as part of the protocol.*

In this work, we initiate a study of time-lock puzzles in a model built upon a realistic (and falsifiable) computational framework. We present a new formal definition of *residual complexity* to characterize a realistic, gradual time-release for time-lock puzzles. We also present a general definition of timed multi-party computation (MPC) and both sequential and concurrent composition theorems for MPC in our model.

1 Introduction

Modern cryptography has expanded its scope much beyond cryptography's traditional use for ensuring secrecy to also address support for numerous interactions and distributed protocols in the form of authentication, signatures, secure (zero-knowledge) validations, distributed secure and resilient multiparty computing (MPC), and more. Another such frontier supporting time management in interactive settings, is the notion of delayed-release cryptography, which incorporates the "time granularity" into cryptographic contexts by exploiting computational properties/limitations to enforce secrecy *for some amount of time* rather than essentially "indefinitely" (in reality, indefinitely is measured as any polynomial in the large enough security parameter). Time-lock puzzles have been studied since the seminal work of Rivest, Shamir, and Wagner (RSW) [33] more than twenty-five years ago. More recently, inherently sequential functions motivated by large scale consensus,

† Work performed partially while at SRI International.

distributed ledgers, and blockchain applications have also precipitated considerable research in verifiable delay functions [11, 32, 37]. The interest in time-lock primitives has yielded various notions like non-malleable time-lock puzzles [21], non-malleable timed commitments [25], and UC-security [5, 6] of time-lock puzzles in the random oracle model.

Time-lock primitives enable new techniques and applications for distributed computations. For example, over twenty years ago, Boneh and Naor [12] introduced timed commitments as a way to achieve fairness in MPC. Recently, Wan et al. [36] used time-lock puzzles to construct more efficient broadcast with adaptive security. Time-lock primitives permit distributed applications in which one party promises that a value will be revealed at a later time, without requiring the party (or other parties) to be honestly participating, or even be online, at the time of reveal. The only resource to both hide and later reveal secrets is the computational hardness inherent in the initial commitment, which is output-guaranteed; specifically, the commitment comes with the promise to reveal the result in the future without any behavioral conditions regarding the parties.

The Inconsistent State of Time-Release Analysis. To a large extent, timed cryptography still treats cryptographic puzzles as essentially a random oracle [1, 5, 6][1]. At a high level, an iterated algebraic computation is modeled such that each iteration gives a random result, and hence the intermediate computations are all random until the determined time when the puzzle is solved. Others similarly cast the solution of a puzzle into a generic group (or ring) model [2, 25, 34] in order to make analogous claims about the information available before the solution time (in generic models the intermediate results are again random and not related to each other). As a common theme, these works arrive at an "instantaneous" revelation model, where the solver only learns the solution in the final step.[2]

However, Mahmoody et al. [28] showed that time-lock puzzles with superpolynomial gap *cannot* be constructed from just random oracles. Their analysis explicitly considers a timed primitive for which each intermediate step is random. Therefore, analyzing *any algebraic* timed primitive in this way is applying an analysis to the solver that has already been shown does not match reality (or even approximate it) if assuming that a puzzle can be generated (via a trapdoor) much faster than it can be solved. Simply, applying random-oracle analysis to the solver is inconsistent with an algebraic puzzle generator, as we know random-oracle analysis does not yield a puzzle with superpolynomial gap. This leaves an open question of modeling the leakage mode of computational puzzles without a random oracle, which is necessary to fully analyze a time-lock puzzle with superpolynomial gap. We expound on this mismatch in Section 3.

New Foundations for Time-Release Cryptography. Traditionally, analysis and proofs of security of cryptographic primitives are based on as realistic and general a model as possible (in computational terms), which permits us to trust that the claims of our model translate to security in the real world. In contrast to the above assumptions, all cryptographic puzzles are defined over well structured algebraic domains, and are based on (concrete) computational assumptions – and any computational puzzle continuously leaks information about the solution of the puzzle as it is being solved. More specifically, each intermediate state reached during the solving process leaks noticeable information

[1] While the authors in [21] do not explicitly treat their base puzzles as random until solved, we argue that their analysis implicitly does so, as they treat the repeated squaring assumption as having "instantaneous" reveal.

[2] In some cases, this is a limitation of the definition of time-lock puzzles only quantifying secrecy until the time when an adversary can guess the solution with non-negligible probability. In generic group models, the solution is explicitly hidden until the last step.

Protocol	Model	Composition	Reveal	Falsifiable?
Arapinis et al. [1]	RO	UC	Instantaneous	No
Baum et al. '20b [5]	RO	UC	Instantaneous	No
Baum et al. '20a [6]	RO	UC	Instantaneous	No
Freitag et al. [21]	P,RO	NM	Instantaneous	No
Katz et al. [25]	Gen	NM	Instantaneous	No
Our Work	RC	S,C	Gradual	Yes

Table 1: State of Research of Composable Time-Lock Primitives. In the "Model" column, RO stands for "Random Oracle," P stands for "Plain," Gen stands for "Generic Algebraic," and RC stands for "Residual Complexity." In the "Composition" column, UC stands for "Universal Composability," NM stands for "Non-malleability", and S,C stands for "Sequential and Concurrent."

about future nearby iterative upcoming states, while revealing only negligible information about far-away states, until at some point there are no more far-away states and the solution begins being revealed from the intermediate results. A realistic model to analyze the security of a timed primitive must capture this "gradual" revelation of the solution.

Our goal in this work is to initiate a foundation of time-release cryptography using a refutable computational model (rather than an idealization), and to introduce falsifiable formal notions of security with which to characterize the time-release. Additionally, foundations of puzzles have to deal with the notion of composability in order to allow puzzles to serve a larger distributed computation setting, *specifically multi-party computation in which time-lock puzzles are solved as part of the protocol.* We frame our work with respect to other recent time-lock research in Table 1.

1.1 A Framework for Computational Puzzles

Below we discuss subtle issues in constructing and analyzing computational puzzles.

Fine-Grained Polynomial Hardness. Current computational puzzles follow the following blueprint:[3]

1. The puzzle generator uses a trapdoor to efficiently sample a puzzle.
2. The solver uses a sequential algorithm to solve the puzzle. The puzzle is parameterized such that the sequential algorithm is faster than any known method to recover the trapdoor.

In this blueprint, the solver must be able to recover the secret within time that is polynomial in the puzzle's security parameter. Therefore, the (leaky) iterative 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 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 probability r_d of guessing the solution of a (randomly sampled) puzzle that has been *partially solved* in d depth.

[3] Recall that by Mahmoody et al.[28], there is no time-lock puzzle based solely on random oracles with superpolynomial time-gap, and therefore in practice puzzles depend on trapdoors.

Our formalization (defined in Definition 4.5 and explained in Section 2.1) is a generalization of a technique in defining the depth-hardness of certain computational problems in [25] and others.

Residual complexity allows us to model *the full lifetime* of a timed primitive's information release, from the moment it is generated until all parties learn the solution. As an illustration, consider the RSW time-lock puzzle construction [33], which depends on the hardness of an iterative squaring mod RSA modulus N, while the underlying trapdoor problem of finding $\Phi(N)$ remains hard. Specifically, a solution χ is encoded via a puzzle (α, t, N) by setting $\chi = \alpha^{2^t} \bmod N$, where t is a hardness parameter that determines how many times the solver must repeatedly square α mod N in order to discover the solution, and $\log(N)$ is a function of the security parameter λ.

For RSW on small N, it is easy to see that revealing such intermediate steps leaks information about the nearby upcoming solutions. For larger moduli, even knowing $x^2 \bmod N$ leaks non-trivial information about $(x \pm \delta)^2 \bmod N$ for small δ (on the order of $\log(N)$).[4] By expanding the modulus to tune the hardness of the problem, the leakage diminishes but does not disappear because the algebraic structure remains.

In realistic computing scenarios and using clever algorithms[5], the solver learns a nontrivial distribution on the puzzle solution before it performs t squarings; we upper-bound the ability of an adversary to learn the solution by hypothesizing a leakage curve for the RSW scheme given a particular security parameter λ and time parameter t. We then quantify the difference in time between when the best adversary can guess a puzzle solution (with non-negligible probability) and the time that the honest parties learn the solution via the scheme's solve algorithm, and can name the moment when the adversary can guess with non-negligible probability as the *critical time*.[6]

Modeling Multi-Party Protocols with Timed Primitives. Equipped with the notion of "hardness up to some depth," we formalize security in the presence of a depth-bounded adversary for protocols with timed hardness. We provide a new model (in Section 5) for depth-bounded secure computation and a framework of security which includes considerations of timed primitives and the subtleties that they incur, including a full treatment of privacy and leakage in fine-grained depth, and the security degradation that results from composing timed-secure primitives.

Leakage and Temporary Privacy. As explained in Section 2, the definition of residual complexity implies a formal treatment of the leakage of a timed primitive, which we expand to a full idealization of a leaky functionality that slowly reveals a hidden value. Our treatment of idealized leakage allows us to capture applications where sensitive information is revealed during the computation, but must not be revealed before some specific point in time. For example, consider an accountable computing application, where parties time-lock their inputs and are held accountable to them at a later time. In traditional definitions of MPC, 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*

[4] This is in contrast to a random oracle analysis, for which the next step of a puzzle is always independent of the current state.

[5] As another example, using parallel processors to compute forward mapping tables for quadratic residues modulo N.

[6] 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.

all of the parties' inputs. However, given our treatment of idealized leakage, we can formally state that no adversary can learn more than a simulator *that learns information at some predetermined pace.*

Simulation and Composition in Fine-Grained Depth. We encounter similar nuances when defining security of multi-party protocols. Our model departs from the standard cryptographic regime of "security up to arbitrary composition within complexity class $\mathbb{P}$," and must account for the depths of all involved machines – the adversary, the simulator, and the distinguisher. Our definition (Definition 5.1) is a generalization of the standard definition in the literature, accounting for fine-grained depths.

As a first consideration for simulating a timed-puzzle, we observe that the simulator must run in less time than the puzzle requires to solve. Otherwise, the claim of privacy via reduction will completely fail: the reduction becomes a claim that an adversary can do no worse *than a simulator that could solve a puzzle outright and learn the solution.*[7] This observation alone leads to new questions about whether existing works apply to a realistic 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. [8] Freitag et al. [21] 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 elegant, 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.

We encounter additional challenges when composing secure timed primitives. For example, sequentially compose 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.

We present two composition theorems for depth-bounded secure computation (Theorems 5.1 and 5.2 in Section 5.4). These theorems include privacy degradation when protocols are composed; specifically, for every additional protocol added to a (concurrent or sequential) composition, the overall scheme is secure against a slightly smaller adversary. Note that our techniques 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.

[7] For phased simulations, discussed in Section 5.5, this becomes that the simulator should run in less depth per phase than the adversary.

[8] 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.

1.2 Paper Outline

Since this paper introduces a new model, it contains a longer than usual motivation, and discussion of subtleties, definitions, and formal issues involved in this work. We build toward a general definition of timed multi-party computation (MPC) and a composition theorem for MPC in our model, which together with the preceding intuitive discussions and the ensuing proofs, validate our definitions and modeling choices. In Section 2 we outline our approach to founding timed-cryptographic-primitives on a computational, falsifiable model, and we describe some of the subtleties in modeling time-release protocols. Section 3 provides a technical explanation of the inconsistency of current time-lock analysis. In Section 4 we present our model of depth-secure computation and introduce residual complexity. In Section 5 we model depth-secure MPC and state our composition theorems. In Section 6 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. Our full composition theorems are in Sections 7 and 8. Section 9 discusses related work in time-lock primitives and granular computational models. Our model is expanded in Appendix A.

2 Technical Overview and Main Results

A realistic model for timed cryptographic primitivies must consider the gradual release of information. In this section, we discuss the subtleties that this implies in our model and fit our definitions into a framework of MPC for which *puzzles are solved as part of a protocol.*

2.1 Residual Complexity: A Falsifiable Leakage Model for Timed Primitives.

To realistically and falsifiably model the "leakage" of a timed primitive, we introduce the notion of *residual complexity.* Intuitively, residual complexity measures the "remaining hardness" of a puzzle that has already been (partially) solved by an adversary of depth d.[9]

Definition 2.1 (Residual Complexity (Informal)). *A puzzle scheme has residual complexity r_d if no depth-d adversary can guess the solution of a randomly sampled puzzle with probability more than r_d.*

By this definition, 1-r_d is the "remaining hardness" of the puzzle after attempting to solve it in d depth. We present the formal definition of residual complexity in Definition 4.5 in Section 4.4.

To show that residual complexity is compatible with (and specifically, a generalization of) existing time-lock definitions, we prove (in Section 6) that a time-lock puzzle by the definition of Bitansky et al. [8] implies small residual complexity until the time-lock expires (as illustrated in Figure 1).

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 provide example curves of puzzle schemes in Figure 1. We note that residual complexity is a more complete characterization of a puzzle than prior definitions, which extend their definitions to characterize the leakage only to the point that an adversary can guess the solution with non-negligible probability, which we refer to as the "critical time."

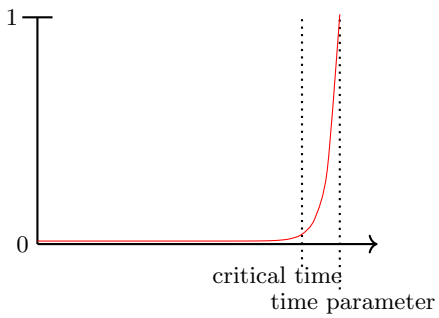
[9] Note that the remaining hardness measures *pseudo-entropy* rather than entropy, as the solution of a timed primitive is always committed at the moment it is generated. (Otherwise the solving algorithm could not be deterministic.)

Falsifiability. Because puzzles always depend on the hardness of their underlying computational assumption, it is difficult for us to compute a puzzle's leakage curve. However, a hypothesized curve can be falsified via an efficient experiment between a challenger and a refuter, where the challenger repeatedly samples puzzles and the refuter must guess solutions.

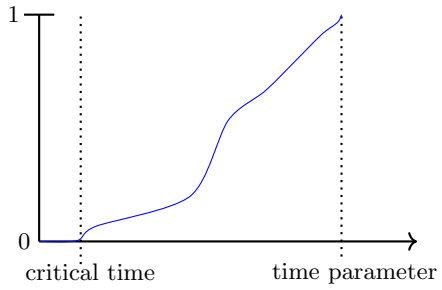
For example, let a challenger claim that for a particular puzzle scheme/size, finding the solution with non-negligible probability requires the solver's state to be at most t steps from the depth of the "honest" solution algorithm, which is given by the time parameter. A refuter asserts that it is able to guess the solution earlier with non-negligible probability p when its local state is at least $t+1$ depth less than the puzzle duration. The challenger and refuter run an interactive falsification procedure in polynomial time (as advocated by Naor [31]):

- The challenger iteratively samples n independent random puzzle instances (e.g., an RSA modulus of a given size and a solution).
- For each puzzle, the refuter runs at most up to $t+1$ steps close to the time parameter, and outputs a guess. For each instance i, if the refuter's guess matches the solution then let $X_i = 1$; otherwise $X_i = 0$.
- Finally, apply the Chernoff bound: $\Pr[|\sum_{i=1}^{n}(X_i/n) - p| > \delta] < 2e^{-2\delta^2 n/2p(1-p)}$

For example if the refuter claims $p = \frac{1}{10}$ then we can choose n large enough (as a polynomial in the security parameter) to show with all but exponentially small error that the frequency of correct guesses is at least $\frac{1}{20}$ (i.e., $\delta = \frac{1}{20}$). At the same time, if the actual probability is negligible, then we will reject the refutation as most likely none of the challenges will be guessed correctly by the refuter, and certainly (with very high probability) only fewer than than $\frac{1}{20}$.[10]



(a) Example leakage profile for a time-lock puzzle.



(b) Example leakage profile for a puzzle scheme that briefly hides its solution.

Fig. 1: Illustration of leakage profiles for two kinds of puzzles. 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.

[10] We use the symmetric Chernoff bound, while knowing that employing the one-sided gives better sample size, an issue we do not address in this abstract.

2.2 Definition of Depth-Secure Protocols

Idealizing Leaky Functionalities. We model the time-release of information by idealizing leaky functionalities which slowly provide information to the parties in a set of *phases* which take the place of time steps. A leaky functionality is parameterized by the leakage curve of the puzzle scheme it idealizes, which determines how much information (as a reduction in pseudo-entropy, as described above) the functionality provides to the adversary at each phase.

Privacy Until Some Time. During simulation of a leaky functionality, we require that the simulator knows no more information at each phase than the adversary, which by definition learns information as the ideal functionality reveals it. As a technical consequence of this requirement, the simulator's input does not include the information which is revealed slowly throughout a computation; the simulator also learns information only as the ideal functionality provides leakage. It follows that in the security reduction, 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.* We can then claim that for time-release computations, privacy of some information holds for a specific amount of time, after which it is revealed and the adversary (respectively simulator) can use it.

Simulation Budgets and Depth-Secure MPC. Our treatment of time-based primitives and protocols requires a granular, depth-based definition of secure computation, where 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). After surpassing some predefined depth, it is alright for the protocol to lose security, as the information is revealed anyway. Similarly, we also bound the depth of a distinguisher (or environment) in tandem with the adversary. However, our definitions must bound more than the depth of the adversary (and distinguisher[11]); they must also bound the depth of the simulator. Otherwise, the security reduction fails where privacy is involved: the security claim becomes that an adversary in the real world can do no worse than a simulator in the ideal world – but if the latter can explicitly solve the puzzle, then no security is proven! 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:

Definition 2.2 (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 remains indistinguishable from the distribution of real executions for any d_a-bounded adversary and any d_e-depth bounded distinguisher (environment).*

2.3 Composition of Depth-Secure Protocols.

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* adversary than the first protocol's adversary by the size of the second's simulator.

Theorem 2.1 (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.*

[11] To generalize between the works of [21] and [25], our definition states the depth of the environment, but the variable could be either polynomially bounded or unbounded. See [21] for discussion.

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 specialized, 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 2.2 (Special 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. These two theorems provide endpoints of a spectrum of budget depletion between the two extremes; when computing depletion for a specific composition, it is possible to reason about the execution time of each simulator. We discuss composition further in Section 5.4.

Applying the Model Concretely When composing primitives in practice, we use the idea of a critical time described above. Consider that an application requires privacy of a primitive until time t. The protocol designer paramaterizes the scheme such that no one refutes (no experiment or mathematical analysis has shown otherwise to date) that the puzzle is unsolvable in t depth. We therefore conclude that the scheme is secure against a t-depth adversary. After depth t, we assume that the adversary can use the solution (earlier than honest parties learn the solution, who must wait for the time parameter).

2.4 Example Application: Simultaneous Multiple Round Auction

Our framework for depth-secure multiparty computation with timed cryptographic primitives enables us to reason about composition of timed primitives in a realistic setting. As an illustration, consider the following variant of a Simultaneous Multiple Round Auction [30] with partial knowledge restriction and forced reveal.[12] As is standard in time-lock literature, we use time-locked bids to guarantee that no party has information about other parties' bids within a round, while allowing for forced reveal.

Stage 1: Every party i submits a bid $b_{1,i}$ in an auction for the first round of bidding, to be opened not before time t_1.

Stage 2: After t_1, all bids for the first stage of the auction are opened. The parties who submitted the top c (constant) bids in Round 1 are chosen to submit stage-2 bids ($b_{2,i}$), to be opened not before t_2.

Winner: The winner of the auction is the party i that maximizes $B = b_{1,i} + b_{2,i}$ constrained such that i's bid in Stage 2 was opened. Party i pays $\$B$.

Observe that this construction requires both *concurrent* and *sequential* composition of timed primitives. Within a stage of bidding, all of the time-lock puzzles require security *in concurrent composition* with each other. For each discrete round of bidding, the concurrently composed bids require sequential composition in order to be analyzed in the framework of a larger protocol.

[12] This variant also has only a fixed number of rounds, but this can be trivially extended as per a true Simultaneous Multiple Round Auction.

Tuning for Timed Security We apply our composition theorems to estimate parameters to tune the security of concurrent primitives *within the first round*. Assume a protocol for submitting and solving an individual puzzle bid which can be set to be (d_a, d_s, d_e)-secure. If the concurrent composition of n puzzles must be secure against an adversary of depth t_1, then each individual puzzle must be tuned such that $d_a = t_1 + (n-1)d_s$. The depth of the composed simulation is at most nd_s. The concurrent composition for this round is then (at least) (t_1, nd_s, d_e)-secure.

To analyze the full example above, including sequential composition between rounds, we analogously apply the composition theorems again, tuning so that the first round is secure for t_1 depth, the second round is secure for $t_2 - t_1$ depth, and the full composition is secure for t_2 depth.

2.5 Leaky Functionalities: Are Random Oracles Required for Composable Timed Primitives?

The work of Baum et al. [6] showed that random oracles are *necessary* to construct UC-secure time-lock puzzles. We argue that by analyzing time-lock puzzles with *leaky* idealizations, it may be possible to avoid this dependence (on a paradigm that we discuss at length in Section 3).[13] The difference in modeling approaches for the idealized primitive can be understood as follows:

- In the model of [6], a timed primitive is given to the simulator by the ideal functionality, and *without further interaction* the simulator must equivocate the output to match whatever the functionality commits.
- In our model, a timed primitive is a promise given to the simulator by the ideal functionality to reveal a value later, and the simulator continuously interacts with the functionality to gradually learn the value until it is fully revealed.

Their proof, which is based off of one by Cleve [16], proceeds by analyzing a two-party coin flipping protocol with guaranteed output delivery. The ideal functionality is defined to provide the output to the adversary in the beginning of the execution, while honest parties receive it at the end. In a real protocol execution, they show there must be a point in the protocol at which one party sends a message which substantially impacts the output of the protocol, and if this party is corrupted so that it does not send the message, the output of the protocol must change. They then argue, based on the fact that an ideal functionality chooses the protocol output at the beginning, that the simulator in the ideal experiment must actually be able to derive the correct output from before this message is sent (intuitively, the simulator must perform some computation which reveals an already-committed output, possibly including solving some puzzle), without any further interaction with the environment or ideal functionality. This, however, leads to a contradiction: the distributions of the output in the real world must differ based on whether the message was sent, but in the ideal world the distributions cannot differ. They use this to conclude that in order to achieve such a functionality, the model must allow the simulator to cheat in a strong way, for example by programming a random oracle.

While appreciating the modeling efforts (avoided in many earlier works), we argue that this model is not the correct way to understand timed primitives (due to subtleties!). Indeed, the committed output must be revealed by the end of the protocol, but if a timed primitive is modeled to *gradually reveal*, then the simulator must indeed continue to interact with the ideal functionality in

[13] Note that we do not implement secure time-lock puzzles in the full UC model as our model is depth-constrained, but use this argument to further our case for a formal treatment of timed primitives in a realistic model.

the ideal world. It then may not need to actively interfere with the output. Hence, we formally describe simulation for leaky timed primitives, in which the ideal functionality continuously interacts with the simulator, in Section 5.5.

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

In this section we discuss the subtle mismatch arising when security of time-lock constructions is proved using random oracle-based analysis. Similar to the classical result of [14] in the random oracle model and [17, 23] about the Fiat-Shamir transform [20], we discuss that security of the realizations of these desired constructions do not follow from the analysis.

Mahmoody et al.'s Result and Popular Idealized Analysis. Mahmoody et al.[28] showed that time-lock puzzles based only on random oracles *cannot* provide super-polynomial gap between generation and solving time. Their analysis explicitly considers time-lock puzzles for which each step in the solving process produces a random result.

This analysis is mirrored by analyses of time-lock puzzle solving in the literature. For example, Baum et al.[5, 6] *explicitly* model an idealized time-lock functionality that provides a uniformly random result at each step of the solving process. This form of analysis provides the next step in the solve algorithm as a random element that provides no more information than the element itself. In the strong algebraic group model of [2, 25] and the generic ring model of [34], 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 so far, but leaks nothing more about the final solution.

Analytical Mismatch. Analyzing a trapdoor-based time-lock construction by modeling the solving process as if a random oracle leads to apparent contradiction. On the one hand, algebraic constructions are believed to realize super-polynomial gaps between generation and solving. On the other hand, Mahmoody's analysis in which each next step is random and independent has been shown to only yield polynomial gaps. The state of current analysis is that puzzles are generated using a trapdoor and then the solving process is treated as a random oracle. These analyses do not match the realization, which should account for the computational difficulty of the solving process.

Implicit Random Oracles. 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.[21] defer to analysis by Pietrzak [32] that assumes the hardness of repeated squaring. But the formalization simply assumes 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. Therefore, these techniques as well are not differentiated in any meaningful way from the analysis of Mahmoody et al. and incur the same analytical mismatch as above.

Modeling Leakage After the Lock Expires. In the examples above, the repeated squaring assumption fails to model the leakage as the solver approaches the final squaring. Moreover, time-lock definitions such as those by [8] or [21] only require that the puzzle remain hard to solve until "close" to the honest solution time fall implicitly into the same trap. For these definitions, the modeling is still

incomplete. The difference between the time when the honest parties arrive at the solution (following the honest solve algorithm), and the hypothesized time when the "gap" expires and the adversary can guess the solution with noticeable probability, is important to modeling a time-lock puzzle utilized during an MPC protocol. In this time, the adversary *can use* the puzzle solution with a head-start over the honest parties, even when honest parties are fully synchronized. We therefore advocate for a complete analysis that *fully models* the leakage of computational puzzles.

4 Definitions

We denote by $[m]$ the sequence $\{1, 2, \ldots, m\}$ 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 4.1 (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 4.2 (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* $X \overset{d}{\approx} Y$ *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 4.3 (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*

- $Z \leftarrow \mathsf{Puz.Gen}(t, \chi)$ *is a probabilistic algorithm over difficulty parameter* $t \in \mathbb{N}$ *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* t*, solution* $\chi \in M_\lambda$*, and puzzle* Z *in the support of* $\mathsf{Puz.Gen}(t, \chi)$*,* $\mathsf{Puz.Solve}(Z)$ *outputs* χ*.*
- ***Efficiency:***
  - $Z \leftarrow \mathsf{Puz.Gen}(t, \chi)$ *can be computed in size* $\mathsf{poly}(\log t, \lambda)$*.*
  - $\mathsf{Puz.Solve}(Z)$ *can be computed in size* $t \cdot \mathsf{poly}(\lambda)$*.*

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

Definition 4.4 (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 Leakage

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 [24, 35] of a puzzle solution from the perspective of a computationally bounded solver.

Definition 4.5 (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* $\lambda \in \mathbb{N}$*:*

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

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. One can consider $1 - r(\lambda)$ to be the remaining hardness of the puzzle.

Leakage Using the definition of residual complexity, we can define the leakage of a puzzle over time. For depths d_1 and d_2 the quantity $r_{d_2} - r_{d_1}$ represent the loss in pseudo-entropy of a puzzle between d_2 and d_1. We call the function describing the loss in pseudo-entropy at each level of depth a *leakage curve*.

For any puzzle scheme, there exists a family of leakage functions indexed by the security parameter and the time parameter, denoted $\mathcal{L} = \{\ell_{\lambda,\tau^*}\}_{\lambda,\tau^*}$, such that each function describes, based on the parameterization, the information a party can extract from the puzzle over time. The time parameter τ^* describes the number of sequential operations required to "solve" the puzzle in the honest case. The security parameter λ tunes the computational difficulty of guessing the solution before τ^* has elapsed. Specifically, λ parameterizes the underlying computational problem which the iterative process solves; for the RSW puzzle, λ describes the size of the modulus used for repeated squaring. In this work, we always consider a specific leakage function ℓ and elide the subscripts from the notation because they are implied by context.

Intuition: The Distribution of Solutions Intuitively, we consider the *leakage* that a party obtains on a puzzle to inform the distribution that the party learns on the puzzle's solution. Any puzzle-solving strategy must imply a distribution on the strategy's "best guess" of a puzzle solution at any point in time. When a party receives a puzzle, the distribution of its best guess has very high pseudo-entropy. As the party learns leakage on the solution, the pseudo-entropy of the distribution of the solution decreases, and the distribution redistributes its mass until eventually all of the mass lies in a single point: the puzzle's solution. We can then understand the leakage of a puzzle to provide a distribution on the solution of the puzzle for every depth d. The residual complexity r_d gives an upper bound on the mass implied at the point of the puzzle solution.

5 Modeling 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 modeling multi-phase functionalities in the ideal/real paradigm and 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 [13] 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.

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 that 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 5.1 (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 5.1. 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 [22, 27], 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.

Definition 5.2 (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 Composition

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 [13]. We use the notation that $\Gamma^{\pi,\rho}$ denotes the concurrent composition of π and ρ.

General/Concurrent Composition We now state our general composition theorem, which includes concurrent composition.

Theorem 5.1 (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. The full proof is deferred to Section 8.

Remark 5.2 (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.

Discussion: Non-malleability. One might think that Theorem 5.1 implies that any depth-secure puzzle is non-malleable because we have shown that the protocols are naively composable. This is not quite true; our result says that a secure protocol remains secure (and non-malleable) only for the *min* granular depth of the concurrent runtimes, and against a smaller adversary. In comparison, nonmalleability definition as in the definition of [21] are secure for *arbitrary polynomial* runtime of the adversary; however, they prove only bounded nonmalleability (is possible), and require tuning parameters based on the number of composed primitives (as we do).

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 5.2 (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.*

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[13]).

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 5.1.

Corollary 5.1 (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 5.1, and in fact when the simulator does not need to be rewound, the protocol is also concurrently composable.

Discussion. Theorem 5.2 and Corollary 5.1 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.

5.5 Simulation for Leaky Functionalities

In the standard definition of secure multi-party computation(MPC) [22, 27], the simulator is given – as input – the adversary's ideal-world outputs, and then it must produce a view for the adversary that is indistinguishable from its view in a real execution. For functionalities where honest parties' inputs are not revealed, privacy is implied by this definition because the simulator must produce such an execution *without access to the honest parties' inputs or outputs.* However, in some leaky

applications the adversary *may* learn the honest parties' inputs, but crucially, the honest parties' inputs are hidden for some period of time.[14] For such applications, the standard MPC definition does not imply privacy of honest parties' inputs up to the point in time that they are revealed because the simulator receives the honest parties' inputs in the beginning.

When we require privacy for some amount of time, we decompose the simulation into a series of phases such that in each phase the simulator learns only the information which is permitted to be revealed at the end of that phase.[15] Specifically, an ideal functionality is parameterized with the leakage function ℓ of the puzzle it emulates. In each phase, the simulator receives from the functionality only the information that the functionality is defined to release during that phase. The leakage is specified by the difference in residual complexity between phases, as described in Section 4.4, which is given by the parameterized leakage function ℓ. This means that in contrast to standard definitions of secure computation, the simulator does not receive all of the adversary's outputs as an input to the computation. This enforces that the simulator does not learn any information before it is supposed to, which implies privacy of the inputs until they are revealed.

6 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 6.1 (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

[14] For example, in accountable computing scenarios.

[15] This is morally similar to the simulation technique of Baum[5], where the simulator forwards next-step queries to the ideal functionality and returns the result. Our simulator automatically learns the leakage for a phase (which may be longer than one step) at the beginning of the phase so that it can simulate the rest.

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

7 Sequential Composition of Depth-Secure Protocols: Proof of Theorem 5.2

In this section, we provide the full proof of Theorem 5.2, which we restate below for convenience.

Theorem 5.2 (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 5.2, 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 1 *$\mathcal{S}$'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 7.1. *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 2 *$\mathcal{B}$ 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 3 *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 4 *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 3, 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 4, 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 2), 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.

□

8 Concurrent Composition of Depth-Secure Protocols: Proof of Theorem 5.1

In this section, we provide the full proof of Theorem 5.1, which we restate below for convenience. Recall the notation that $\Gamma^{\pi,\rho}$ denotes the concurrent composition of π and ρ. Similarly, $\zeta^{F,G}$ is a functionality that concurrently provides functionalities F and G. We also let $\mathsf{VIEW}_{\mathcal{A}}(\cdot)$ denote the view of $\mathcal{A}$ during the enclosed experiment.

Theorem 5.1 (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. 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 8.1. *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 5 *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 6 *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 5 and 6, $\mathcal{B}$ 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.

9 Related Work

We briefly highlight classical works in time-delayed and fine-grained cryptography that we build on, we also review the most related recent work. In spite of the early work, ours is the first (to the best of our knowledge) that considers composition of timed primitives with fine-grained security. Recent work (below) on composability of non-malleable time-lock puzzles and timed-commitments do not provide a full treatment of fine-grained security.

Time-lock Puzzles. The seminal work on time-lock puzzles was produced by Rivest, Shamir, and Wagner (RSW) [33]. Boneh and Naor [12] introduced timed-commitments, which progressed the study of using timed primitives for fairness in MPC. Verifiable delay functions, which are cryptographic primitives that depend on sequential work in order to delay release of information, have also been the focus of several recent research efforts [10, 32, 37]. Bitansky et al.[8] formally defined time-lock puzzles and constructed them using randomized encodings, which in turn depend on indistinguishability obfuscation. They also construct weak time-lock puzzles (with fast *parallel* generation time, although sequentially they may take longer to generate than to solve) from one-way functions. Baum et al.[5, 6] formalized time-lock puzzles in the UC model [13]. Freitag et al.[21] built publicly verifiable, non-malleable time-lock puzzles. Katz et al.[25] recently constructed non-interactive non-malleable timed-commitments that come with a proof that they can be forced open. They additionally showed that in a quantitative group model, speeding up squaring is as hard as factoring. In terms of negative results, Mahmoody et al.[28] proved that in the random oracle model, there are no time-lock puzzles with more than polynomial time gap.

Two additional works of note have addressed the assumptions underlying the repeated squaring problem in idealized models. Rotem and Segev [34] 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.

Time-locked Cryptography and Composition. The recent work by Baum et al.[6] studies composition of time-lock puzzles in the UC model. The recent work by Freitag et al. [21] studies concurrent composition of non-malleable primitives. To our knowledge, these are the only other work that consider composition of time-based primitives. The comparison with Baum is not straightforward because we generalize depth-secure computation and Baum considers concurrent composition of time-lock puzzles in UC; there are common themes which receive different treatment. Baum provides an idealized version of RSW's underlying assumption, while we introduce a generic model which may leak information. Freitag et al. consider depth-bounded adversaries against concurrent composition of time-lock primitives with other time-lock primitives and imply small leakage until a puzzle is solved, but do not consider the more general composition with MPC.

9.1 Techniques for Timed Primitives

We next overview techniques in the literature that are similar to our own for modeling and composing 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.[19] 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.[26] 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 [29] 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 [33], 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. To enforce time-based privacy, we model idealized leaky functionalities that respond to environment-directed time. With respect to our functionality, we discuss 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 time-lock puzzle 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.

Simulation of Time-Lock Puzzles in Phases. The work of Chvojka et al.[15] builds time-release encryptions and sequential time-lock puzzles, and uses a phased simulation technique to argue the security of their puzzles. They define a sequential time-lock puzzles to be one where an intermediate solution is considered to be the starting point of the next puzzle in the sequence. However, the intermediate values provided to their simulators in the arguments of security treat a different issue than arises in our proof. When they consider the simulatability of a sequential puzzle, they

must show how to simulate an intermediate step without computing the previous steps explicitly. Their technique is to provide a function of the previous steps as advice to the simulator for an intermediate step, but they cannot provide the true solutions for previous sequential steps as input to their simulator.

Our simulation in phases treats an entirely different issue; we consider the ability of an uninterrupted simulator to successfully simulate a prefix of the execution ending in a specific phase, given only the information it may learn by the end of that phase. In our case, all of the intermediate values of a prefix of the execution are available to the simulator; in their case, the intermediate values from a prefix of the sequential puzzle solution are not available.

9.2 Comparison with Other Definitions

We now provide a deeper treatment comparing our model with popular approaches for defining time-lock puzzles in the existing literature. In both cases, the provided treatment is insufficient for completely modeling the use of such a primitive in composition with other protocols.

Bitansky's Time-lock Definition. Bitansky et al. [8] formalized the notion that up to a certain point in time before the puzzle is solved, the solver's distribution on the puzzle solution retains very high pseudo-entropy. We reproduce their notion in Definition 4.4 in Section 4.3. Intuitively, for any puzzle with polynomial running time t and any polynomial time solver running in time up to t^{ε}, where $\varepsilon < 1$ is called the *gap parameter*, the solver gains only negligible advantage in guessing the solution of a challenge puzzle. The "few" described above is then cleverly guaranteed by this definition to be no sooner than the solver has run in the time-gapped t^{ε} time. This definition, however, does not consider what happens *after* the solver exceeds the time-gapped running time. (We remark that the notion of a time-lock puzzle by [21] has a similar feel, where the solver runs within some polynomially smaller time than the puzzle has been tuned for, and also does not describe what happens after the time-lock puzzle expires but before the honest parties solve the puzzle.)

Blum-Blum Shub. As a more concrete and illustrative example of what happens when a time-based primitive reaches the end of its guarantees, we recall the generalized Blum-Blum-Shub (BBS) assumption by Boneh and Naor [12].

Definition 9.1 ($(n, n', \delta, \varepsilon)$-Generalized BBS Assumption [12]). *For $g \in \mathbb{Z}$ and a positive integer $k > n'$, let $W_{g,k} = \langle g^2, g^4, g^{16}, \ldots, g^{2^{2^i}}, \ldots, g^{2^{2^k}} \rangle$. Then for any integer $n' < k < n$ and any $\delta * 2^k$-depth-bounded $\mathcal{A}$,*

$$| \ \Pr[\mathcal{A}(N, g, k, W_{g,k} \bmod N, g^{2^{2^{k+1}}}) \ = \ 1] \ - \ \Pr[\mathcal{A}(N, g, k, W_{g,k} \bmod N, R^2) \ = \ 1] \ | \leq \ \varepsilon$$

where the probability is taken over the random choice of an n-bit RSA modulus $N = p_1 p_2$, where p_1 and p_2 are equal primes satisfying $p_1 = p_2 = 3 \mod 4$, and element $g \in \mathbb{Z}_N$, and $R \in \mathbb{Z}_N$.

The assumption was first introduced for the design of a pseudo-random generator [9], and then elegantly generalized [12] to develop timed commitments (with a trapdoor). As noted in [12], the assumption states that given the sequence of repeated squares $W_{g,k}$ of some generator g, the $k+1$st element in the sequence $g^{2^{2^{k+1}}}$ is indistinguishable from a random quadratic residue, for any party whose running time is much less than 2^k. This suffices for showing the pseudo-randomness of the BBS generator.

However, for timed protocols in which any party solves a puzzle as part of the protocol, eventually the depth of the puzzle solver *must* approach the sequence length 2^k *by definition*. At this point, the guarantee of the BBS assumption breaks down! Indeed, as the solver approaches the duration of the time-lock – even before it finally learns the solution – the distribution of the solver's "best guess" on the solution becomes more refined over time.

References

1. Arapinis, M., Lamprou, N., Zacharias, T.: Astrolabous: A universally composable time-lock encryption scheme. Cryptology ePrint Archive, Report 2021/1246 (2021), https://ia.cr/2021/1246
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)
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., Nielsen, J.B., Oechsner, S.: Craft: Composable randomness beacons and output-independent abort mpc from time. Cryptology ePrint Archive, Report 2020/784 (2020), https://eprint.iacr.org/2020/784
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. Blum, L., Blum, M., Shub, M.: Comparison of two pseudo-random number generators. In: Crypto82. pp. 61–78. Plenum (1982)
10. Boneh, D., Bonneau, J., Bünz, B., Fisch, B.: Verifiable delay functions. In: CRYPTO (1). LNCS, vol. 10991, pp. 757–788. Springer (2018)
11. 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
12. Boneh, D., Naor, M.: Timed commitments. In: Crypto'00. p. 236–254. LNCS, Springer-Verlag, Berlin, Heidelberg (2000)
13. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067 (2000)
14. Canetti, R., Goldreich, O., Halevi, S.: The random oracle methodology, revisited (preliminary version). In: STOC. pp. 209–218. ACM (1998)
15. Chvojka, P., Jager, T., Slamanig, D., Striecks, C.: Versatile and sustainable timed-release encryption and sequential time-lock puzzles. Cryptology ePrint Archive, Report 2020/739 (2020), https://ia.cr/2020/739
16. Cleve, R.: Limits on the security of coin flips when half the processors are faulty (extended abstract). In: STOC. pp. 364–369. ACM (1986)
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. Egashira, S., Wang, Y., Tanaka, K.: Fine-grained cryptography revisited. In: ASIACRYPT (3). LNCS, vol. 11923, pp. 637–666. Springer (2019)
20. 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)
21. Freitag, C., Komargodski, I., Pass, R., Sirkin, N.: Non-malleable time-lock puzzles and applications. Cryptology ePrint Archive, Report 2020/779 (2020), https://eprint.iacr.org/2020/779
22. Goldreich, O.: Foundations of Cryptography: Volume 2, Basic Applications. Cambridge University Press, USA, 1st edn. (2009)

23. Goldwasser, S., Kalai, Y.T.: On the (in)security of the fiat-shamir paradigm. In: FOCS. pp. 102–113. IEEE Computer Society (2003)
24. 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)
25. 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)
26. 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)
27. 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)
28. 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)
29. Malavolta, G., Thyagarajan, S.A.K.: Homomorphic time-lock puzzles and applications. In: CRYPTO (1). LNCS, vol. 11692, pp. 620–649. Springer (2019)
30. Milgrom, P.: Putting auction theory to work: The simultaneous ascending auction. Journal of political economy **108**(2), 245–272 (2000)
31. Naor, M.: On cryptographic assumptions and challenges. In: CRYPTO. Lecture Notes in Computer Science, vol. 2729, pp. 96–109. Springer (2003)
32. Pietrzak, K.: Simple verifiable delay functions. In: ITCS. LIPIcs, vol. 124, pp. 60:1–60:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2019)
33. Rivest, R.L., Shamir, A., Wagner, D.A.: Time-lock puzzles and timed-release crypto. Tech. rep. (1996)
34. 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)
35. Vadhan, S.P., Zheng, C.J.: Characterizing pseudoentropy and simplifying pseudorandom generator constructions. In: STOC. pp. 817–836. ACM (2012)
36. 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)
37. 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 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. In our model, 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.