Non-Malleable Time-Lock Puzzles and Applications

Naomi Ephraim[*] Cody Freitag[†] Ilan Komargodski[‡] Rafael Pass[§]

Abstract

Time-lock puzzles are a mechanism for sending messages "to the future", by allowing a sender to quickly generate a puzzle with an underlying message that remains hidden until a receiver spends a moderately large amount of time solving it.

We introduce and construct a variant of a time-lock puzzle which is *non-malleable*. A non-malleable time-lock puzzle guarantees, roughly, that it is impossible to "maul" a puzzle into one for a related message without solving it. The security of this construction relies on the existence of any (plain) time-lock puzzle and it is proven secure in the auxiliary-input random oracle model. We show that our construction satisfies bounded concurrency and prove that it is impossible to obtain full concurrency. We additionally introduce a more general non-malleability notion, termed *functional* non-malleability, which protects against tampering attacks that affect a specific function of the related messages. We show that in many (useful) cases, our construction satisfies *fully* concurrent functional non-malleability.

We use our (functional) non-malleable time-lock puzzles to give efficient multi-party protocols for desirable tasks such as coin flipping and auctions. Our protocols are (1) *fair*, meaning that no malicious party can influence the output, (2) *optimistically efficient*, meaning that if all parties are honest, then the protocol terminates within two message rounds, and (3) *publicly verifiable*, meaning that from the transcript of the protocol anyone can quickly infer the outcome, without the need to perform a long computation phase. Our protocols support an unbounded number of participants and require no adversary-independent trusted setup. Our protocol is the first protocol that satisfies all of the above properties *under any assumption*. Security is proven based on the repeated squaring assumption and in the auxiliary-input random oracle model. Along the way, we introduce a publicly verifiable notion of time-lock puzzles which is of independent interest. This notion allows the solver of the puzzle to compute the solution together with a proof which can be quickly verified by anyone—even if the puzzle is maliciously crafted.

[*]Cornell Tech, `nephraim@cs.cornell.edu`

[†]Cornell Tech, `cfreitag@cs.cornell.edu`

[‡]NTT Research, `ilan.komargodski@ntt-research.com`

[§]Cornell Tech, `rafael@cs.cornell.edu`

Contents

1 Introduction

Time-lock puzzles (TLPs), introduced by Rivest, Shamir, and Wagner [RSW96], are a cryptographic mechanism for committing to a message, where a sender can (quickly) generate a puzzle with a solution that remains hidden until the receiver spends a moderately large amount of time solving it (even in the presence of parallel processors). Rivest et al. [RSW96] gave a very efficient construction of TLPs where security relies on the repeated squaring assumption. This assumption postulates, roughly, that it is impossible to significantly speed up repeated modular exponentiations in a group of unknown order, even when using many parallel processors. This construction and assumption have proven extremely useful in various (and sometimes unexpected) applications [BN00, LPS17, Pie19, Wes19, EFKP19, MT19, DKP20], some of which have already been implemented and deployed in existing systems.

Non-malleability. In a Man-In-the-Middle (MIM) attack, an eavesdropper tries to actively maul intermediate messages to compromise the integrity of the underlying values. To address such attacks, Dolev, Dwork and Naor [DDN91] introduced the general concept of non-malleability in the context of cryptographic commitments. Roughly speaking, non-malleable commitments are an extension of plain cryptographic commitments (that guarantee binding and hiding) with the additional property that no adversary can maul a commitment for a given value into a commitment to a "related" value. As this is a fundamental concept with many applications, there has been a tremendous amount of research on this topic [Bar02, PR05b, PR05a, LPV08, PPV08, LP09, PW10, Wee10, Goy11, LP11, GLOV12, GPR16, COSV16, COSV17, Khu17, LPS17, KS17, KK19].

Non-malleable TLPs and applications. To date, non-malleability has never been considered in the context of TLPs (or other timed primitives). Indeed, the construction of TLPs of [RSW96] *is malleable.*[1] This fact actually has negative consequences in various settings where TLPs could be useful. For instance, consider a scenario where n parties perform an auction by posting bids on a public bulletin board. To implement this fairly, a natural approach is to use a commit-and-reveal style protocol, where each party commits to its bid on the board, and once all bids are posted each party publishes its opening. Clearly, one has to use non-malleable commitments to guarantee that bids are independent (otherwise, a malicious party can potentially bid for the maximal other bid plus 1). However, non-malleability is not enough since there is a fairness issue: a malicious party may refuse to open after seeing all other bids and so other parties will never know what the unopened bid was.

Using non-malleable TLPs to "commit" to the bids solves this problem. Indeed, the puzzle of a party who refuses to reveal its bid can be recovered after some moderately large amount of time by all honest parties. This style of protocol can also be used for fair multi-party collective coin flipping where n parties wish to agree on a common unbiased coin. There, each party encodes a random bit via a TLP and all parties will eventually agree on the parity of those bits.[2] This gives a highly desirable collective coin flipping protocol with an important property that we refer to as *optimistic efficiency*: when all parties are honest and publish their "openings" immediately after

[1]The puzzle of [RSW96] for a message s and difficulty T is a tuple $(g, N, T, s \oplus g^{2^T} \bmod N)$, where N is an RSA group modulus and g is a random element from $\mathbb{Z}_N$. The puzzle is trivially malleable since the message is one-time padded.

[2]In the context of coin flipping, the fairness issue mentioned above where a malicious party prematurely aborts can bias the output [Cle86]. Boneh and Naor [BN00] used timed primitives and interaction to circumvent the issue in the two-party case but we care about the multi-party case and prefer to avoid interaction as much as possible.

seeing all puzzles, the protocol terminates and all parties agree on an unbiased bit. As we will see, no other known protocol for this (highly important) task has this property.

1.1 Our Results

We next give our main results. In Section 1.1.1, we give our generic non-malleable time-lock puzzle construction based on any TLP, and we discuss the various notions of non-malleability that we consider in this setting. In Section 1.1.2, we show how to make our construction satisfy a strong public verifiability property using a specific time-lock puzzle based on repeated squaring. Finally, in Section 1.1.3, we discuss the applications of our construction for fair multi-party protocols.

1.1.1 Non-Malleable Time-Lock Puzzles

Our first result is a construction of a non-malleable TLP. We emphasize that, as explained above, this primitive is not only natural on its own right, but also has important applications to the design of secure protocols for various basic tasks. Our construction relies on the existence of any given TLP [RSW96, BGJ+16] and is proven secure in the (auxiliary-input) random oracle model [Unr07]. Our construction achieves non-malleability even in a bounded *concurrent* setting [PR08], where a man-in-the-middle (MIM) attacker participates as a receiver in n_{left} of concurrent sessions "on the left" and as a sender in n_{right} concurrent sessions "on the right". In the left executions, it receives n_{left} TLPs with underlying values $s_1, \ldots, s_{n_{\mathsf{left}}}$, and attempts to generate puzzles in the right executions with underlying solutions $\tilde{s}_1, \ldots, \tilde{s}_{n_{\mathsf{right}}}$ that are somehow non-trivially related to $s_1, \ldots, s_{n_{\mathsf{left}}}$. We refer to this as $(n_{\mathsf{left}}, n_{\mathsf{right}})$-concurrency.

Theorem 1.1 (Informal; See Theorem 4.2 and Corollary 4.3)**.** *For every $n_{\mathsf{left}}, n_{\mathsf{right}}, L \in \mathrm{poly}(\lambda)$, assuming that there is a TLP (supporting 1-bit messages) that is secure for attackers of size $2^{n_{\mathsf{right}} \cdot L} \cdot \mathrm{poly}(\lambda)$, there exists an $(n_{\mathsf{left}}, n_{\mathsf{right}})$-*concurrent non-malleable *TLP supporting messages of length L. The scheme is proven secure in the auxiliary-input random oracle model.*

In terms of security, our reduction is *depth preserving*: if the given TLP is secure against attackers of depth $T(\lambda)/\alpha(\lambda)$, where $\alpha(\cdot)$ is a fixed polynomial independent of T denoting the advantage of an attacker, then the resulting non-malleable TLP is secure against attackers of depth $T(\lambda)/\alpha'(\lambda)$ for another fixed polynomial $\alpha'(\cdot)$. In particular, the dependence on T in hardness is preserved. Additionally, note that if $n_{\mathsf{right}} \cdot L \in O(\log \lambda)$, then the underlying TLP only needs to be polynomially secure.

Instantiating the TLP with the construction of [RSW96], our scheme is extremely efficient: encoding a message requires a single invocation of a random oracle and few (modular) exponentiations. Additionally, our construction is very simple to describe: to generate a puzzle for a solution s with opening r, we sample a puzzle for (s, r) using randomness which itself depends (via the random oracle) on s and r. Nevertheless, the proof of security turns out to be somewhat tricky and non-trivial; see Section 2 for details. The efficiency and simplicity of our construction stand in contrast to a previous construction [DKP20] whose main feature is that it is in the plain model (without any form of setup), albeit they rely on a variety of assumptions (including keyless hash functions and NIWI) and their construction is not practical.

We prove that our scheme is non-malleable against all polynomial-size attackers that cannot solve the puzzles (and this is inherent as the latter ones can easily maul any puzzle). We even allow the attacker's description to depend arbitrarily on the random oracle. We formalize this notion by showing that our TLP is non-malleable in the *auxiliary-input* random oracle model, a model that was introduced by Unruh [Unr07] (see also [CDGS18]) in order to capture preprocessing attacks, where a non-uniform attacker obtains an advice string that depends arbitrarily on the random oracle. Thus, in a sense, our construction does *not* require any form of attacker-independent setup.

We emphasize that our protocol only achieves *bounded* concurrency, where the number of instances the attacker participates in is a priori bounded (and the scheme may depend on this bound). We show that the stronger notion of *full* concurrency, which does not place such limitations and is achievable in all other standard settings of non-malleability, is actually impossible to achieve for TLPs. Therefore, our result is best possible in this sense.

Theorem 1.2 (Informal; See Theorem 4.17)**.** *There is no fully concurrent non-malleable TLP (even in the random oracle model).*

In a nutshell, the impossibility from Theorem 1.2 is proven by the following generic MIM attack. Given a puzzle z, if the number of "sessions" the attacker can participate in at least as large as $|z|$, they can essentially generate $|z|$ puzzles encoding *the bits of* z. Since the distinguisher of the MIM game (which is now given those bits) can run in arbitrary polynomial time, it can simply solve the original puzzle and recover the original solution in full. This attack is circumvented in the bounded concurrency setting (Theorem 1.1) by setting the length of the puzzle to be longer than the concurrency bound. Specifically, to support n concurrent puzzles on the right, we can set the message length to $\lambda \cdot n$, which is what results in exponential security loss $2^{\lambda \cdot n}$ as discussed above.

Functional non-malleability. We note that the attack on fully concurrent non-malleable time-lock puzzles crucially relies on the fact that the *distinguisher* in the MIM game can solve the underlying puzzles. However, it is easy to see that if the distinguisher is restricted to bounded depth, this attack fails. One could define a *weaker* notion of non-malleability where the MIM distinguisher is depth-bounded, but this results in a weaker security guarantee. In particular, we show in Appendix A that there exists a natural TLP construction that satisfies this (weaker) definition yet has a valid mauling attack.[3]

In light of this observation, we introduce a new definition of non-malleability that *generalizes* the standard notion considered in Theorem 1.1. We call the notion *functional* non-malleability and, as the name suggests, the security notion is parameterized by a class of functions $\mathcal{F}$. Denote by L the bit-length of the messages we want to support and by n the number of sessions that the MIM attacker participates in on the right. We think of $f \in \mathcal{F}$ as some *bounded depth* function of the form $f\colon (\{0,1\}^L)^n \to \{0,1\}^m$, which is the target function of the input messages that the MIM adversary is trying to bias. Formally, we require that the distinguisher in the MIM game cannot distinguish the function f applied to the values underlying the puzzles the MIM attacker outputs. When $\mathcal{F}$ includes all identity functions (which are bounded depth and have output length $m = n \cdot L$), functional non-malleability implies the standard definition of concurrent non-malleability (as the distinguisher just gets all the messages from the n mauled puzzles).

Naturally, it makes sense to ask what guarantees can we get if we a priori restrict f, say in its output length, without limiting the number of sessions n. This turns out to particularly useful when the application at hand only requires non-malleability against a specific form of tampering functions (this indeed will be the case for us below). Concretely, let $\mathcal{F}_m$ be the class of all functions whose output length is at most m bits and which can be computed in depth polynomial in the security parameter λ and in $\log(n \cdot L)$ (using the notation given above). Then, we have the following result.

Theorem 1.3 (Informal; See Theorem 4.2)**.** *Assuming that there exists a TLP, then for every* $m \in \mathrm{poly}(\lambda)$ *there exists a* fully concurrent functional non-malleable *TLP for the class of functions*

[3] As we discuss in the Section 1.3, concurrent works allow the distinguisher to be bounded depth.

$\mathcal{F}_m$. *The scheme is proven secure in the auxiliary-input random oracle model assuming the given TLP is secure for all attackers of size at most* $2^m \cdot \mathrm{poly}(\lambda)$.

The above construction satisfies the *depth-preserving hardness* property in the same way as the construction from Theorem 1.1. Further, note that as long as $m \in O(\log \lambda)$, we only require standard polynomial hardness from the given TLP. We remark that Theorem 1.3 will turn out to be instrumental for our applications we discuss below. We also believe that the abstraction of functional non-malleability is important on its own right and view it as an independent contribution.

1.1.2 Publicly Verifiable Time-Lock Puzzles

In addition to non-malleability, we construct TLPs that also have a public verifiability property: after a party solves the puzzle, they can publish the underlying solution together with a proof which can be later used by anyone to *quickly* verify the correctness of the solution. We emphasize that this must hold even if the solver determines that the puzzle has no valid solution. We believe this primitive is of independent interest.

We build our non-malleable, publicly verifiable TLP assuming a very weak form of (partially) trusted setup. The setup of our TLP consists of a set of many public parameters where we only assume that at least one of them was generated honestly. We call this model the All-But-One-string (ABO-string) model.[4] We design this to fit into our multi-party protocol application (see Theorem 1.5 below) in such a way where the parties themselves will generate this setup in the puzzle generation phase. Indeed, as we discuss below, publicly verifiable TLPs in the ABO-string model will imply coin flipping *without setup*.

Theorem 1.4 (Informal; See Corollary 5.10)**.** *Assuming the repeated squaring assumption, there exists a publicly verifiable non-malleable TLP in the ABO-string model. The construction is proven secure in the auxiliary-input random oracle model.*

Our construction is depth-preserving and requires the same message-length dependent security as in the constructions Theorem 1.1 and Theorem 1.3, depending on the type of non-malleability desired for the resulting TLP.

To construct our publicly verifiable TLP, we use a *strong trapdoor VDF* (formalized in Definition 5.3). A trapdoor VDF (not necessarily a strong one) has the property that the generator of the puzzle has a trapdoor which allows for quickly solving it. Known VDFs, e.g., [Wes19, Pie19], can be instantiated to satisfy this property.

While it may seem a priori immediate that any trapdoor VDF implies a publicly verifiable TLP, there is a significant challenge with carrying out this approach assuming only a weak form of (partially) trusted setup. Specifically, the trapdoor VDFs based on known constructions all rely on honestly sampled public parameters. Naively, if we allow the TLP generator to sample the public parameters on their own, it would allow them to give false proofs regarding the opening to their puzzle.

This attack can be mitigated with a *strong* trapdoor VDF: a trapdoor VDF with the additional property that *for every* well-formed public parameters, VDF soundness holds. We show that (somewhat surprisingly) Pietrzak's VDF [Pie19] (which is based on the repeated squaring assumption) can be instantiated to satisfy this property using the group of signed quadratic residues QR_N^+ where N is a product of safe primes.[5]

[4] Our ABO-string model is a variant of the multi-string model of Groth and Ostrovsky [GO14], where it is assumed that a majority of the public parameters are honestly generated.

[5] For this, we assume that sampling uniformly random safe primes can be done in efficiently; this is a pretty common assumption, see [VZGS13] for more details.

Lastly, we remark that standard VDF soundness only holds if the challenge for the proof phase is generated honestly. In Pietrzak's protocol, this is accomplished using a random oracle. However, since we are in the auxiliary input ROM, intuitively the attacker might "know" an accepting proof of a false statement. This is where we use the ABO-string model—we seed the random oracle using the partially trusted common random string, which, as we show, suffices for soundness.

1.1.3 Fair Multi-Party Auctions and Coin Flipping

As we mentioned above, an appealing application of non-malleable TLPs is for tasks such as fair multi-party auctions or coin flipping. Our protocols (for both tasks) are extremely efficient and consist of just two phases: first each party "commits" to their bid/randomness using some puzzle, and then after all puzzles are made public, each party publishes its solution. If some party refuses to open their puzzle, a force-opening phase is performed.

In what follows, we focus on the task of fair multi-party coin flipping, which is a core building block in recent proof-of-stake blockchain designs; see below. The application to auctions follows in a similar manner. It is convenient to consider our protocol in a setting where there is a public bulletin board. Any party can publish a puzzle to the bulletin board during the commit phase and then publish its solution after some pre-specified amount of time has elapsed. Our protocol requires no form of *adversary-independent* trusted setup (not even a common random string). We further emphasize the following highly desirable properties we achieve:

- **Optimistic Efficiency:** If all participating parties are honest, then the protocol terminates within two message rounds (without the need to wait the pre-specified amount of time for the second phase), and all parties can efficiently verify the output of the protocol.

- **Fairness:** No malicious party can bias the output of the protocol. Namely, as long as there is at least one honest participating party, the output will be a (nearly) uniformly random value.

- **Public Verifiability:** In the case that any participating party is dishonest and does not publish their solution, any party can break the puzzle in a moderate amount of time and provide a publicly verifiable proof of the solution. We even require that an honest party can prove that a published puzzle has *no* valid solution.

We note that optimistic efficiency is a typical feature of multi-party protocols using the "commit-and-reveal" paradigm, which we employ using time-lock puzzles as the analog of commitments. However, to obtain fairness and public verifiability, we make use of the additional properties of our time-lock puzzles: concurrent functional non-malleability and public verifiability. As described above, these can be achieved simultaneously relative to a random oracle (which the attacker may depend on arbitrarily) in the ABO-string model, assuming the repeated squaring assumption in this model.

Theorem 1.5 (Informal; See Theorem 6.1)**.** *Assuming the repeated squaring assumption, there exist multi-party coin flipping and auction protocols that satisfy optimistic efficiency, fairness, and public verifiability. The protocols support an unbounded number of participants and require no adversary-independent trusted setup. Security is proven in the auxiliary-input random oracle.*

We emphasize that our protocol supports an a priori unbounded number of participants. This may seems strange in light of our impossibility from Theorem 1.2. We bypass this lower bound (as mentioned above) by observing that for most natural applications (including coin flipping and auctions), the notion of functional non-malleability from Theorem 1.3 suffices. The key insight is that we only need indistinguishability with respect to specific depth-bounded functions with a priori bounded output lengths (e.g., parity for coin flipping, or taking the maximum for auctions). Since the output length in both cases is known, we can actually support *full* concurrency which translates into having an unbounded number of participants. Additionally, when the protocol's output length is short, we note that we do not need complexity leveraging and can base our protocol on standard polynomial hardness.

We can view our protocol both as a fully non-interactive protocol where all participants solve all submitted puzzles, or as a two-round protocol where participants open their puzzles after some specified time. The two-round version allows for optimisitic efficiency, but we believe the non-interactive protocol may still be of interest. For auctions, we note that our protocols are the first multi-party protocols *under any assumption* that satisfy fairness against malicious adversaries and requires no adversary-independent setup—using the timed commitments of [BN00] works only in the two-party setting and additionally relies on trusted setup, and using the homomorphic time-lock puzzles of [MT19] does not satisfy fairness in the presence of malicious adversaries. For coin flipping, our two-round protocol is the first multi-party protocol that is fair against malicious adversaries while satisfying optimisitic efficiency. Next, we provide a more in depth comparison of our non-interactive coin flipping protocol with existing solutions.

Non-interactive coin flipping. In the non-interactive setting, Boneh et al. [BBBF18] proposed a VDF-based protocol. Specifically, each party publishes a random string r_i and then the agreed upon coin is defined by running a VDF on the seed $H(r_1\| \dots \|r_n)$, where H is a random oracle. As the VDF must be evaluated to obtain the output, this type of protocol does not satisfy optimistic efficiency. Nevertheless, the VDF-based protocol has the advantage that only a single slow computation needs to be computed, whereas our non-interactive protocol requires n such computations for n participants (which can be done in parallel). Malavolta and Thyagarajan [MT19] address this inefficiency in the context of time-lock puzzles (which do allow for the option of optimistic efficiency) by constructing *homomorphic* time-lock puzzles, where many separate puzzles can be combined into a single puzzle to be solved. However, their TLP scheme is malleable and so cannot be directly used to obtain a fair protocol against malicious adversaries.[6] We note, however, that because our protocol satisfies public verifiability, only a single honest party needs to solve each puzzle, and this computation can easily be delegated to an external server.

The VDF-based scheme of [BBBF18] can be based on repeated squaring in a group of unknown order based on the publicly verifiable proofs of [Wes19, Pie19]. In this setting, the protocols can either be instantiated using RSA groups that require attacker-independent trusted setup, or based on class groups that rely only on a common random string. As we do in this work, the common random string can be implemented in the ABO-string model using a random oracle (which the attacker may depend on arbitrarily). Therefore, when restricting our attention to protocols without attacker-independent setup, the previous VDF-based protocols are based on less standard assumptions on class groups, whereas our protocol can be instantiated from more standard assumptions on RSA groups.

Privacy. Let us remark that the protocols that we described guarantee fairness but not privacy. The latter, however, can be obtained in specific applications by composing our protocols with existing privacy-preserving tools such as Anonize [HMPS15].

[6] It is possible to make this protocol maliciously secure using concurrent non-malleable zero-knowledge proofs [BPS06, OPV10, LPTV10, LP11], proving that each party acted honestly, but this (1) makes the construction significantly less efficient, and (2) requires either trusted setup and additional hardness assumptions, or additional rounds of interaction.

1.2 Related Work

Non-malleable TLPs. The study of non-malleability in the context of TLPs is new to this work and as we mentioned it has exciting applications. Such a puzzle was implicitly constructed in [DKP20] (presented there as a non-malleable code for bounded polynomial depth attackers). That construction is in the plain model, without any setup assumptions, however it relies on a variety of assumptions and it should be viewed as a feasibility result. Our approach, on the other hand, is more oriented towards practical constructions and therefore we are willing to assume some very weak form of setup (i.e., AI-ROM). Additionally, our construction is somewhat reminiscent to the Fujisaki-Okamoto (FO) transformation [FO13] used to generically transform any CPA-secure public-key encryption scheme into a CCA-secure one using a random oracle. Nevertheless, while conceptually the transformation is similar, since we are dealing with concurrent non-malleability in the bounded polynomial depth setting, our actual proof turns out to be quite challenging.

Timed commitments. Boneh and Naor [BN00] introduced timed commitments, which can be viewed as a publicly verifiable and interactive TLP. They additionally require that the puzzle (which is an interactive commitment) convinces the receiver that if they brute-force the solution, they will succeed. Because of this additional property, their commitment scheme is interactive and relies on a less standard assumption called the generalized Blum-Blum-Shub assumption. Their scheme is additionally malleable.

Fair coin flipping in blockchains. Generating unbiased bits is one of the largest bottlenecks in modern proof-of-stake crypto-currency designs [BPS16, DPS19, DGKR18]. Recall that in a proof-of-stake blockchains, the idea is, very roughly speaking, to enforce "one vote per unit of stake". This is usually implemented by choosing random small committees at every epoch and letting that committee decide on the next block. The main question is how to obtain "pure" randomness so that the chosen committee is really "random".

One option is to use the hash of an old-enough block as the randomness. Unfortunately, it is known that the hash of a block is not completely unbiased: an attacker can essentially fully control about logarithmically many of its bits. In existing systems, this is mitigated by "blowing up" parameters to compensate for the (small yet meaningful) advantage the attacker has, making those systems much less efficient. Using a mechanism that generates unbiased bits, we could make proof-of-stake crypto-currencies much more efficient.

1.3 Concurrent Work

Several related papers [BDD$^+$20b, KLX20, BDD$^+$20a] have been developed concurrently and independently to this work. The works of Baum et al. [BDD$^+$20b, BDD$^+$20a] formalize and construct various (publicly verifiable) time-based primitives, including TLPs, under the Universal Composability (UC) framework [Can01]. Katz et al. [KLX20] (among other results, less related to ours) introduce and construct non-malleable non-interactive timed-commitments. While the notions that are introduced and studied are related, the results are all incomparable as each paper has a somewhat different motivation which leads to different definitions and results.

Definitions. Let us start by comparing definitions. Katz et al. consider a CCA-style definition and Baum et al. consider a UC-style definition, both adapted to the depth-bounded setting. In the classical setting of unbounded polynomial-time attackers, these definitions are usually stronger than "only" non-malleability, but this is not generally true in the depth-bounded setting.

In more detail, they consider a depth-bounded version of CCA/UC-style security, where the attacker/environment (who is also the distinguisher) is bounded to run in time less than the hardness of the timed primitive. We, on the other hand, allow the distinguisher of the MIM game to be unbounded (while only the attacker is bounded). We believe this is an important distinction and we provide more insights into the differences between the bounded and unbounded distinguisher settings in Appendix A. Specifically, we show that non-malleability with a depth-bounded distinguisher is (essentially) equivalent to our definition of functional non-malleability with output length 1. (In particular, our construction of Theorem 1.3 immediately gives a concurrent non-malleable time-lock puzzle against depth-bounded distinguishers assuming only polynomially secure TLPs in the auxiliary-input random oracle model.) Next, we give a construction separating the definitions of non-malleability with an unbounded vs. depth-bounded distinguisher, showing that non-malleability in the bounded distinguisher setting gives a strictly weaker security guarantee. We also give a discussion comparing these definitions to different settings of functional non-malleability in Appendix B.

Comparison with [KLX20]. Recall that timed commitments [BN00] (ignoring non-malleability for now) allow one to commit to a message m in such a way that the commitment hides m up to some time T, yet the verifier can be sure that it can be force opened to *some* value after roughly T time. In contrast, plain TLPs are not necessarily guaranteed to contain valid messages. In this context, our notion of publicly-verifiable TLPs is in between these two notions: we treat puzzles without a solution as invalid (say encoding $\perp$) but we additionally provide a way to publicly verify that this is the case after it has been solved. Nevertheless, we note that the construction of Katz et al. constructions does not imply a TLP since their commitment procedure takes T time (while TLP generation should take time essentially independent of T).

Regarding assumptions, their construction is proven secure in the algebraic group model [FKL18] and relies on trusted setup, while ours is proven secure in the (auxiliary-input) random oracle model and hence requires no adversary-independent trusted setup. Lastly, both constructions rely on repeated squaring as the source of depth-hardness, and theirs additionally makes use of general-purpose NIZK (which requires setup).

Comparison with [BDD$^+$20b, BDD$^+$20a]. The construction of a UC-secure TLP given in [BDD$^+$20b] relies on a programmable random oracle, whereas our construction relies on a non-programmable (auxiliary-input) random oracle. In fact, they prove that their (incomparable) notion of UC security cannot be achieved in the non-programmable random oracle model. In a follow-up work [BDD$^+$20a], they show that their time-lock puzzle construction satisfies a notion of public verifiability. However, they achieve public verifiability only for honestly generated puzzles, that is, one can prove that a puzzle has a solution s, but cannot prove that a puzzle has no solution. In our terminology, we refer to this as one-sided public verifiability (see Definition 5.2). In contrast, our construction achieves full verifiability. This property is crucial for our efficient coin flipping protocol since it allows only one honest party to (attempt to) solve any invalid puzzle. With only one-sided public verifiability, every participant would need to solve all invalid puzzles, and the output of the coin-flip can only be efficiently verified (in time less than T) in the case that all puzzles are honestly generated.

2 Technical Overview

In Section 2.1, we give an overview of our non-malleable time-lock puzzle construction and its proof of security. Then in Section 2.2, we overview our construction public verifiable (and non-malleable) time-lock puzzles from repeated squaring. Finally in Section 2.3, we discuss how our non-malleable and publicly verifiable time-lock puzzle construction can be used for fair multi-party coin flipping with various desirable properties.

We start by recalling the definition of TLPs, as necessary to give an overview of our techniques. A TLP consists of two algorithms $(\mathsf{Gen}, \mathsf{Sol})$. Gen is a probabilistic procedure that takes as input an embedded solution s and a time parameter t, and outputs a puzzle z. Sol is a deterministic procedure that on input a puzzle z for time bound t, outputs a solution in depth (or parallel time) roughly t. We note that TLPs can be thought of as a fine-grained analogue to commitments where "hardness" of the puzzle means that the puzzles are hiding against distinguishers of depth less than t. On the other hand, hiding *can be broken* in depth t (using Sol). Additionally, we require that Sol always finds the correct underlying solution s for a puzzle z. This corresponds to perfect biding in the language of commitments.

2.1 Non-Malleability for Time-Lock Puzzles

Non-malleability, at a high level, requires that any man-in-the-middle (MIM) attacker $\mathcal{A}$ that receives a puzzle z that has an underlying solution s cannot output a different puzzle $\widetilde{z}$ for a related value $\widetilde{s}$. Formally, we consider the (inefficient) distribution $\mathsf{mim}_{\mathcal{A}}(t, s)$ that computes $\widetilde{z} = \mathcal{A}(\mathsf{Gen}(t, s))$ and outputs the value $\widetilde{s}$ underlying $\widetilde{z}$. However, if $z = \widetilde{z}$, then $\widetilde{s} = \bot$ (since simply forwarding the commitment does not count as a valid mauling attack), and if there is no valid value underlying $\widetilde{z}$, then $\widetilde{s}$ is also $\bot$. Then, non-malleability requires that for any solution s and MIM attacker with depth much less than t (so it cannot break hiding), the distribution for value $\widetilde{s}$ given by $\mathsf{mim}_{\mathcal{A}}(t, s)$ is statistically indistinguishable from the distribution $\mathsf{mim}_{\mathcal{A}}(t, 0)$ for an unrelated value, 0. We emphasize that indistinguishability should hold even against *unbounded* distinguishers that, in particular, can run Sol.[7]

Our construction. Our construction relies on any time-lock puzzle TLP and a common random oracle $\mathcal{O}$. We now describe our non-malleable TLP, which we denote nmTLP. In order to generate a puzzle for a solution s that can be broken in time t, $\mathsf{nmTLP.Gen}$ uses randomness r and feeds $s\|r$ into the random oracle to get a string r_{tlp}. It then uses $\mathsf{TLP.Gen}$ to create a puzzle with difficulty t for $s\|r$ using randomness r_{tlp}. That is,

$$\mathsf{nmTLP.Gen}(t, s; r) := \mathsf{TLP.Gen}(t, s\|r; \mathcal{O}(s\|r)).$$

Note that in order to solve the puzzle output by $\mathsf{nmTLP.Gen}$, it suffices to just solve the puzzle generated using $\mathsf{TLP.Gen}$, which takes time t. In other words, $\mathsf{nmTLP.Sol}(t, z)$ simply computes $s\|r = \mathsf{TLP.Sol}(t, z)$ and outputs s.

Hardness. To show the hardness of nmTLP relative to a random oracle, we rely on the hardness of TLP in the plain model, against attackers of depth much less than t. At a high level, we show that breaking the hardness of nmTLP requires either guessing the randomness r used to generate the randomness $r_{\mathsf{tlp}} = \mathcal{O}(s\|r)$ for the underlying puzzle, or directly breaking the hardness of TLP, both

[7] We could also naturally consider a computational notion of non-malleability that requires only computational indistinguishability. Still, an arbitrary polynomial time attacker could also run the Sol algorithm.

of which are infeasible for bounded attackers. To formalize this, we consider any depth-bounded distinguisher $\mathcal{D}^{\mathcal{O}}$, who receives as input a nmTLP puzzle z corresponding to solution s_0 or s_1 and distinguishes the two cases with noticeable probability. By construction, z actually corresponds to a TLP puzzle for $s_0||r_0$ or $s_1||r_1$, so we would like to use $\mathcal{D}$ to construct a distinguisher against the hardness of TLP.

We first note that if $\mathcal{D}$ never makes a query to $\mathcal{O}$ containing the randomness r_b underlying z, then we can simulate $\mathcal{O}$ by lazily sampling it in the plain model, and hence use $\mathcal{D}$ as a distinguisher for the hardness of TLP. If $\mathcal{D}$ does make a query containing r_b, then with overwhelming probability it must have received a puzzle corresponding to $s_b||r_b$ (since in this case, r_{1-b} is independent of $\mathcal{D}$ and its input z). Moreover, all of its queries up until that point have uniformly random answers independent of z, so we can simulate them as well, up until receiving this query. Therefore, in both cases, we can carry out this attack in the plain model and rely on the hardness of TLP.

Non-malleability. To show non-malleability of nmTLP, we want to argue that any depth-bounded man-in-the-middle (MIM) attacker $\mathcal{A}$ cannot maul a puzzle z for s (received on the left) to a puzzle $\widetilde{z}$ (output on the right) for a related value $\widetilde{s} \neq s$. At a high level, whenever $\mathcal{A}$ changes the underlying value s to $\widetilde{s}$, then the output of the random oracle on $\widetilde{s}$ is now uniformly random and independent of z. Indeed, we show that for any fixed puzzle $\widetilde{z}$ and a value $\widetilde{s}$, a randomly generated puzzle for $\widetilde{s}$ will not be equal to $\widetilde{z}$ with high probability (otherwise we show how to break the hardness of TLP). So, intuitively, the only way to generate a *valid* puzzle $\widetilde{z}$ for $\widetilde{s}$ is to "know" the underlying value $\widetilde{s}$, but hardness intuitively implies that no depth-bounded adversary can "know" s.

We formalize this intuition by a hybrid argument to show that the MIM distribution $\widetilde{s} \leftarrow \mathsf{mim}_{\mathcal{A}}(t, s)$ is indistinguishable from $\mathsf{mim}_{\mathcal{A}}(t, 0)$. At a high level, we first replace the inefficient distribution $\mathsf{mim}_{\mathcal{A}}(t, s)$ by a low-depth circuit $\mathcal{B}$. At this point, we want to use the hiding property to indistinguishably swap the puzzle to 0, so the hybrid is now unrelated to s. We describe the key ideas for these hybrids below.

For the first hybrid, the key insight is that we can compute $\mathsf{mim}_{\mathcal{A}}(t, s)$ *in low depth* using an algorithm $\mathcal{B}$ by simply looking at the oracle queries made by $\mathcal{A}$. In this sense, we are relying on the extractability property of random oracles to say that $\mathcal{A}$ must know any valid value $\widetilde{s}$ it generates a puzzle for. Specifically, let $\widetilde{z}$ be the output of $\mathcal{A}$. For every query $(s_i\|r_i)$ that $\mathcal{A}$ makes to $\mathcal{O}$, $\mathcal{B}$ outputs s_i if $\widetilde{z} = \mathsf{nmTLP}(t, s_i\|r_i; \mathcal{O}(s_i\|r_i))$. If there are no such queries, $\mathcal{B}$ outputs $\perp$. $\mathcal{B}$ requires depth comparable to the depth of $\mathcal{A}$ since all of these checks can be done in parallel. Furthermore, the output of $\mathcal{B}$ is indistinguishable from the true output given the above observation that $\mathcal{A}$ cannot output a valid puzzle for a value it doesn't query.

For the next hybrid, we would like to indistinguishably replace the underlying puzzle for s with a puzzle for 0, which would suffice to show non-malleability. Because $\mathcal{B}$ is low-depth, it seems that we should be able to use the hiding property of nmTLP to say that the output of $\mathcal{B}$ does not depend on the underlying value s. Specifically, we want to conclude that if the output of $\mathcal{B}$ (who outputs many bits) is statistically far when the underlying value is s versus 0, then there exists a distinguisher (who outputs a single bit) that can distinguish puzzles for s and 0. Towards this claim, we show how to "flatten" any (possibly unbounded) distinguisher $\mathcal{D}$ who distinguishes between the output of $\mathcal{B}$ in the case where the underlying value is s versus 0. Specifically, we encode the truth table of $\mathcal{D}$ as a low-depth distinguishing circuit of size roughly $2^{|s|}$ to make this reduction go through. As a result, we need to rely on a sub-exponentially security of the underlying TLP when $|s| = \lambda$. Namely, the underlying TLP cannot be broken by sub-exponential sized circuits with depth much less than t. However, when $|s| \in O(\log \lambda)$, we only need to rely on *polynomial* security of the underlying TLP.

Impossibility of fully concurrent non-malleability. Ideally, we would like to achieve fully concurrent non-malleability, meaning that any MIM attacker that receives any polynomial n number of puzzles on the left cannot maul them to n puzzles for related values. However, we show that this is impossible to achieve.

Consider an arbitrary TLP for a polynomial time bound t. We construct a MIM attacker $\mathcal{A}$ that receives only a single puzzle z on the left with solution s where the length of z is L. Then, $\mathcal{A}$ can split z into L bits and output a puzzle on the right for each bit *of the puzzle* z. Then, the values underlying the puzzles output by $\mathcal{A}$ when viewed together yield z, which is related to the value s! More formally, there exists a polynomial time distinguisher that solves the puzzle z in polynomial time t and can distinguish $\mathcal{A}$'s output in the case when it receives a puzzle for s or an unrelated value, say 0.

This implies that for any n which is greater than the size of a puzzle, the TLP cannot be non-malleable against MIM attackers who output at most n puzzles on the right. At a high level, the impossibility follows from the fact that hardness does not hold against arbitrary polynomial-time distinguishers (which usually *is* the case for hiding of standard commitments).

Despite this impossibility, we show that we actually *can* achieve concurrent non-malleability against a *specific class of distinguishers* in the non-malleability game. We refer to this notion as concurrent *functional* non-malleability.

Achieving concurrent functional non-malleability. In many applications, we only need a form of non-malleability to hold with respect to certain classes of functions. For example, in our application to coin flipping, we only need that a puzzle z with solution s cannot be mauled to a set of puzzles $\widetilde{z_1}, \ldots, \widetilde{z_n}$ with underlying values $\widetilde{s_1}, \ldots, \widetilde{s_n}$ such that $\bigoplus_{i\in[n]} \widetilde{s_i}$ "depends on" s. With this in mind, we define a concurrent functional non-malleability with respect to a class of functions $\mathcal{F}$. We say that a TLP satisfies *functional* non-malleability for a class $\mathcal{F}$ if the output of $f(\mathsf{mim}_{\mathcal{A}}(t, s))$ is indistinguishable from $f(\mathsf{mim}_{\mathcal{A}}(t, 0))$ for any $f \in \mathcal{F}$, which also naturally generalizes to the concurrent setting. We note that functional non-malleability for a class $\mathcal{F}$ actually implies standard non-malleability whenever the class $\mathcal{F}$ contains the identity function, so functional non-malleability generalizes the standard notion of non-malleability.

Going back to the proof of standard (non-concurrent) non-malleability for our construction nmTLP, we observe that the security we need for the underlying time-lock puzzle we use depends on $2^{|s|}$ where $|s|$ is the size of the puzzle solutions. Specifically, given any distinguisher in the non-malleability that had input of size $|s|$, we were able to construct a distinguisher for hardness of size $2^{|s|}$. In fact, this exact same proof works in the context of concurrent functional non-malleability for functions f that have *low depth* and bounded output length m. We require f to be low depth so the reduction constitutes a valid attack against hardness, and then we only require security proportional to 2^m!

We briefly discuss how our nmTLP construction works for concurrent functional non-malleability for the class $\mathcal{F}_m$ of function with low depth and output length m. Specifically, for every m, we define a scheme nmTLP_m assuming that TLP is secure against attackers of size roughly 2^m. Because TLP requires security against 2^m size attackers, our construction nmTLP_m also only achieves security against 2^m size attackers. As such, our $\mathsf{nmTLP.Gen}$ algorithm needs to use at least $m + \lambda$ bits of randomness (otherwise an attacker could cycle through all choices of randomness to break security). Recall that $\mathsf{nmTLP}_m.\mathsf{Gen}$ with randomness r outputs a puzzle using $\mathsf{TLP.Gen}$ with solution $s\|r$. As a result, if we want to support solutions of size $|s|$ in nmTLP_m, we need our underlying TLP so support solutions of size at least $|s| + m + \lambda$. By correctness, this implies that our schemes outputs puzzles of size roughly $O(|s| + m + \lambda)$.

Bounded concurrent non-malleability. Our construction of time-lock puzzles for concurrent functional non-malleability can also be seen as a construction for bounded concurrent (plain) non-malleability. Specifically, consider the case where the MIM attacker outputs at most n puzzles on the right. We can think of this as functional non-malleability where the low depth function is simply identity on $n \cdot |s|$ bits. From the above discussion, this implies a protocol assuming a TLP with security against size $2^{n \cdot |s|}$ attackers, with puzzles of size roughly $O(n \cdot |s| + \lambda)$.

Security in the auxiliary-input random oracle model. Finally, we remark that all of our constructions and all formal proofs in the main body of the paper are in the auxiliary-input random oracle model (AI-ROM) introduced by Unruh [Unr07]. In this model, the non-uniform attacker is allowed to depend arbitrarily on the random oracle, so there is no attacker-independent non-uniform advice. At a high level, we use the result from [Unr07] (restated in Lemma 3.6) to conclude that the view of any bounded-size MIM attacker $\mathcal{A}$ with oracle access to $\mathcal{O}$ (where $\mathcal{A}$ may depend arbitrarily on $\mathcal{O}$) is indistinguishable the view of $\mathcal{A}$ with access to a "lazily sampled" oracle $\mathcal{P}$ that is fixed at a set of points F (which depend on $\mathcal{A}$). Formally, in the non-malleability analysis, we switch to an intermediate hybrid where the MIM attacker has access to a partially fixed, lazily sampled oracle $\mathcal{P}$. Then, because the MIM attacker $\mathcal{A}$ must maul honestly generated puzzles that have high entropy, we show that it is necessary for $\mathcal{A}$ to query the oracle $\mathcal{P}$ outside the fixed set of points F. From this, we carefully show that a similar analysis follows as discussed above for the ROM.

2.2 Publicly Verifiable Time-Lock Puzzles

We observe that the non-malleable time-lock puzzle construction nmTLP we described above has a very natural—yet incomplete—public verifiability property. Solving a puzzle yields both the solution s and the randomness r use to generate that puzzle. As such, anyone who solves a valid puzzle can send the opening r to another party, and convince them that s is the unique valid solution to the puzzle. However, we emphasize that this only works for *valid* puzzles and solutions.

Consider the following problematic scenario for our nmTLP construction. Suppose a party "commits" to a value via a puzzle z and refuses to open the commitment. As we said before, if z is a valid puzzle, any party can can solve the puzzle, get the solution s and an opening r that proves that s is the unique solution. What if the puzzle corresponds to *no solution*? We refer to this scenario by saying that the puzzle corresponds to the solution $\perp$. In this case (by definition), there is no solution s and opening r for any such that $z = \mathsf{Gen}(t, s; r)$. Anyone who solve the invalid puzzle—which requires a lot of computational power—*will* be able to conclude that the puzzle is malformed, but they will not be able to convince anyone else that this is the case. Ideally, we would have a time-lock puzzle where Sol additionally outputs a publicly verifiable proof π that the solution it computes is correct, even if the solution may be $\perp$! We refer to such a time-lock puzzle as a *publicly verifiable* time-lock puzzle. We next discuss the definition and our construction of publicly verifiable time-lock puzzles.

Defining public verifiability. More formally, a publicly verifiable time-lock puzzle consists of algorithms $(\mathsf{Gen}, \mathsf{Sol}, \mathsf{Verify})$. As with normal time-lock puzzles, $\mathsf{Gen}(t, s)$ outputs a puzzle z. The algorithm $\mathsf{Sol}(t, z)$ outputs the solution s as well as a proof π that it computed s correctly. Finally $\mathsf{Verify}(t, z, (s, \pi))$ checks that s is indeed the correct solution for the puzzle z (corresponding to $\mathsf{Sol}(t, z)$), using the proof π. In addition to $(\mathsf{Gen}, \mathsf{Sol})$ being a valid time-lock puzzle, we require that Sol and Verify constitute a sound non-interactive argument. In fact, we require a very strong notion of soundness. We need it to be the case that even for maliciously chosen puzzles that have no solution, the time-lock puzzle is still sound—even against the adversary that generated the malformed puzzle. In other words, we require that no attacker can compute a puzzle z, a value s', and a proof π' such that $\mathsf{Verify}(t, z, (s', \pi'))$ accepts yet s' is not the value s computed by $\mathsf{Sol}(t, z)$, which may be $\perp$.

Ideally, we would want a publicly verifiable time-lock puzzle that requires *no setup*. We instead consider a weak form of setup which we refer to as the All-But-One-string (ABO-string) model. In this model, Sol and Verify additionally take as input a string $\mathsf{mcrs} = (\mathsf{crs}_1, \ldots, \mathsf{crs}_n) \in (\{0,1\}^\lambda)^n$, and we require that soundness holds as long as one of the values of crs_i is sampled uniformly (without necessarily knowing which one); this is why we refer to it as the all-but-one string model. We note that in multi-party protocols, the ABO-string model is realistic as each participant $i \in [n]$ can post a value for crs_i. Then, we require soundness to hold as long as one participant is honest, which is a reasonable assumption in this multi-party setting.

Constructing publicly verifiable time-lock puzzles. Our construction of a publicly verifiable time-lock puzzle follows the blueprint of Rivest, Shamir, and Wagner [RSW96] for constructing time-lock puzzles from repeated squaring. Namely, we use the output of a sequential function (repeated squaring in a suitable group) essentially as one-time pad to mask the value underlying the time-lock puzzle. As in [RSW96], we require that the sequential function has a trapdoor so that puzzles can be generated efficiently. Unlike [RSW96], we additionally require that the sequential function is publicly verifiable to enable publicly verifiability for the time-lock puzzle. Finally, we apply the non-malleability transformation described above to achieve full public verifiability. In what follows, we describe each of these steps in more detail.

For the underlying sequential function, we use what we call a *strong trapdoor verifiable delay function* (VDF). A VDF (introduced by Boneh et al. [BBBF18]) is a publicly verifiable sequential function that can be computed in time t but not much faster, even with lots of parallelism. A trapdoor VDF (formalized by Wesolowski [Wes19]) additionally has a trapdoor for quick evaluation. We require a trapdoor VDF in the ABO-string model that satisfies additional properties required by our application. While the properties we define—and achieve—are heavily tailored towards our application, we believe some of the techniques may be of independent interest. More specifically, a strong trapdoor VDF comes with a Sample algorithm to generate inputs for an evaluation algorithm Eval. We emphasize that, even in the ABO-string model, Sample is independent of any form of setup. Previous definitions of VDFs require the proof to be sound with probability over an *honestly sampled* input. In contrast, we require that the proof is sound for any maliciously chosen input that is in the support of the Sample algorithm. We note that this property is satisfied by a variant of Pietrzak's VDF [Pie19] based on repeated squaring. At a high level, this is because Pietrzak's VDF is sound (at least in the random oracle model) for any group of unknown order where no adversary can find a group of low order (see e.g.[BBF18] for further discussion), so by using any RSA group with no low order elements (as in [Pie19]), the proof is sound even if the group is maliciously chosen (yet still a valid RSA group), which gives the strong property we need. We note that the proof of soundness for our strong trapdoor VDF in the ABO-string and auxiliary-input random oracle model follows by a similar argument to that of [Pie19] in the (plain) random oracle model after applying Unruh's Lemma [Unr07] (stated in Lemma 3.6).

Next, we construct what we refer to as a *one-sided* publicly verifiable time-lock puzzle in the ABO-string model by using the strong trapdoor VDF in the RSW-style construction described above. By one-sided, we mean that completeness and soundness hold only for puzzles in the support of Gen (again, we emphasize that this is in contrast to a randomly sampled puzzle). Then, our full construction applies our non-malleability transformation to a one-sided publicly verifiable time-lock puzzle. We already argued that the non-malleability transformation provides a form of public verifiability for puzzles z in the support of Gen. Namely, anyone can prove to another party that a valid puzzle z has a solution s, but the proof may not be sound when trying to prove that a puzzle has no solution. However, we next show that if the underlying puzzle satisfies one-sided public verifiability, then the resulting (NM) PV TLP is sound for any $z \in \{0,1\}^*$ (possibly not in the support of Gen).

Proof of full public verifiability. Let $(\mathsf{Gen}, \mathsf{Sol}, \mathsf{Verify})$ be the TLP resulting from applying our non-malleability transformation to a one-sided PV TLP $(\mathsf{Gen}_{\mathsf{tlp}}, \mathsf{Sol}_{\mathsf{tlp}}, \mathsf{Verify}_{\mathsf{tlp}})$. Consider any puzzle $z \in \{0,1\}^*$. If z is in the support of Gen, we want to ensure that no one can prove that $s' = \bot$ is a valid solution. At the same time, if z is not in the support of Gen, we want to ensure that no one can prove that $s' \neq \bot$ is a valid solution.

When we run $\mathsf{Sol}(t, z)$, we first run $\mathsf{Sol}_{\mathsf{tlp}}(t, z)$ and get a solution $s_{\mathsf{tlp}} = \hat{s} \| \hat{r}$ with a proof π_{tlp}. If $\hat{r}$ is a valid opening for the proposed solution $\hat{s}$, then Sol can simply output the solution $s = \hat{s}$ and the proof $\pi = \hat{r}$. If $\hat{r}$ is not a valid opening for $\hat{s}$, Sol must output $\bot$ and a proof π that this is the case. We set $\pi = (s_{\mathsf{tlp}}, \pi_{\mathsf{tlp}})$, which intuitively gives anyone else a way to "shortcut" the computation of $\mathsf{Sol}_{\mathsf{tlp}}$.

Now suppose that an adversary tries to falsely convince you that a puzzle z with no solution has a solution $s' \neq \bot$ using a proof $\pi' = r'$. To do so, it must be the case that r' is a valid opening for s' with respect to Gen. But if that were the case, then z *would have a solution*, in contradiction.

On the other hand, suppose that an adversary tries to falsely convince you that a puzzle z with solution s has no solution, i.e. $s' = \bot$, using a proof $\pi' = (s'_{\mathsf{tlp}}, \pi'_{\mathsf{tlp}})$. Since z has a solution, it means that z is in the support of $\mathsf{Gen}_{\mathsf{tlp}}$. By one-sided public verifiability, this means that π'_{tlp} is a valid proof that $s'_{\mathsf{tlp}} = \hat{s} \| \hat{r}$ is the correct solution to z with respect to $\mathsf{Gen}_{\mathsf{tlp}}$. So if $\hat{r}$ is not a valid opening for $\hat{s}$ with respect to Gen, we know the adversary must be lying. In other words, the only way the adversary can cheat is by cheating in the underlying one-sided PV TLP on a puzzle z in the support of $\mathsf{Gen}_{\mathsf{tlp}}$.

Discussion of our non-malleable PV TLP. We note that the publicly verifiable time-lock puzzle we described above can be made to satisfy the same non-malleability guarantees as we discuss in Section 2.1 (as we construct it using the same transformation but with a specific underlying time-lock puzzle). Thus, assuming the repeated squaring assumption, we get a publicly verifiable time-lock puzzle that satisfies concurrent function non-malleability for any class of low depth functions $\mathcal{F}_m$ with output length m. Our construction is in the ABO-string model, and we prove security in the auxiliary-input random oracle model (which is needed for soundness of the strong trapdoor VDF in the ABO-string model in addition to the non-malleability transformation). This model is reasonable for our practical applications to multi-party protocols, as we will see below.

We note that our explicit repeated squaring assumption states that repeated squaring in RSA groups for n-bit integers cannot be sped up even by size 2^m adversaries. The repeated squaring assumption is closely related to the assumption on factoring (which has recently been formalized in different generic models by the works of [RS20, KLX20]). The current best known algorithms for factoring run in time at least $2^{n^{1/3}}$. In the case where $m \in O(\log \lambda)$, for example, we only require that polynomial-size attackers cannot speed up repeated squaring, which is a relatively mild assumption. In the case where m is larger, say $m = \lambda$, then we need to choose n to be at least

λ^3 (based on known algorithms for factoring). This gives an example of the various trade-offs we get for the security and efficiency of our construction depending on the class of low depth functions $\mathcal{F}_m$ that we want non-malleability for.

2.3 Fair Multi-Party Protocols

We will focus on coin flipping for concreteness, and note that for auctions the ideas are similar. At a high level, the coin flipping protocol is very simple. Each party chooses a random bit and publishes a time-lock puzzle that encodes the chosen bit. After all puzzles are published, each party open their puzzle by revealing the bit that it used as well as the randomness used to generate the puzzle. Any puzzle that is not opened can be "solved" after a moderately large amount of time t. Once all puzzles have been opened, the agreed upon bit (i.e., the output of the protocol) is the XOR of all revealed bits. The above protocol template is appealing because it naturally satisfies optimistic efficiency: if all parties are honest and open their puzzles, the protocol terminates immediately. When using time-lock puzzles which are both non-malleable (as discussed in Section 2.1) and publicly verifiable (as discussed in Section 2.2), we achieve the following highly desirable properties:

- **Fairness:** No malicious party can bias the output of the protocol.

  This crucially relies on non-malleability for the underlying time-lock puzzle. For a protocol with n participants, we need the time-lock puzzle to satisfy n-concurrent non-malleability. This guarantees that as long as one party is honest, the output of the protocol will be (at least statistically close to) a uniformly random bit.

- **Unbounded participants:** Anyone can participate in the protocol.

  This property might come at a surprise since we show fully concurrent non-malleability is impossible to achieve. However, we emphasize that our time-lock puzzle achieves fully concurrent *functional* non-malleability for the XOR function. This allows us to deal with any a priori unbounded number of participants, which is important in many decentralized and distributed settings.

- **Public verifiability:** Only one party needs to solve each unopened puzzle, and can provide a publicly verifiable proof that it solved it correctly.

  This follows immediately by the public verifiability property we achieve for the underlying time-lock puzzle. Without this property, any unopened puzzles may need to be solved by every party that want to know the output of the protocol, which is prohibitively expensive. However, public verifiability instead opens up the application to *any* party, not even involved in the protocol. Furthermore, this work can even be delegated to an external server since trust is guaranteed by the attached proof.

We note that our non-malleable and publicly verifiable time-lock puzzle is defined in the All-But-One-string (ABO-string) model, which is required for public verifiability. To implement this model, we have each participant i publish a fresh random string $\mathsf{crs}_i \leftarrow \{0,1\}^\lambda$ in addition to its puzzle z_i. Then, whenever some party tries to solve (or verify) a puzzle, it puts all of the random strings together as a multi-common random string $\mathsf{mcrs} = (\mathsf{crs}_1, \ldots, \mathsf{crs}_n)$ from all n participants, and uses this for the publicly verifiable proof. As long as a single party is honest and publishes a random string crs_i independent of all other participants, then the publicly verifiable proof system will be sound.

3 Preliminaries

We denote by $x \leftarrow X$ the process of sampling a value x from the distribution X. For a set $\mathcal{X}$, we denote by $x \leftarrow \mathcal{X}$ the process of sampling a value x from the uniform distribution on $\mathcal{X}$. We let $\mathrm{Supp}(X)$ denote the support of the distribution X. For an integer $n \in \mathbb{N}$ we denote by $[n]$ the set $\{1, 2, \ldots, n\}$. We use PPT as an acronym for *probabilistic polynomial time.* A function $\mathsf{negl} \colon \mathbb{N} \to \mathbb{R}$ is *negligible* if it is asymptotically smaller than any inverse-polynomial function, namely, for every constant $c > 0$ there exists an integer N_c such that $\mathsf{negl}(\lambda) \leq \lambda^{-c}$ for all $\lambda > N_c$.

A non-uniform algorithm $\mathcal{A} = \{\mathcal{A}_\lambda\}_{\lambda \in \mathbb{N}}$ is a sequence of circuits for all $\lambda \in \mathbb{N}$. We assume that $\mathcal{A}_\lambda$ always implicitly receives 1^λ as its first input. We define $\mathsf{size}(\mathcal{A}_\lambda)$ to be the size of the circuit (corresponding to total time) and $\mathsf{depth}(\mathcal{A}_\lambda)$ to be the depth of the circuit (corresponding to parallel time).

For $a, b \in \mathbb{N}$, we let RF_a^b denote the set of all functions $f \colon \{0,1\}^a \to \{0,1\}^b$. A *partial assignment* to $\{0,1\}^a$ is a function $F \colon S \to \{0,1\}^b$ where $S \subseteq \{0,1\}^a$. We let $\mathsf{RF}_a^b[F]$ denote the set of functions f consistent with F, namely functions $f \in \mathsf{RF}_a^b$ where $f(x) = F(x)$ for all $x \in S$.

3.1 Time-Lock Puzzles

We first define the standard definition of time-lock puzzles without any additional properties.

Definition 3.1. *Let $B \colon \mathbb{N} \to \mathbb{N}$. A B-*secure time-lock puzzle (TLP) *is a tuple* $(\mathsf{Gen}, \mathsf{Sol})$ *with the following syntax:*

- $z \leftarrow \mathsf{Gen}(1^\lambda, t, s)$**:** *A PPT algorithm that on input a security parameter $\lambda \in \mathbb{N}$, a difficulty parameter $t \in \mathbb{N}$, and a solution $s \in \{0,1\}^\lambda$, outputs a puzzle $z \in \{0,1\}^*$.*
- $s = \mathsf{Sol}(1^\lambda, t, z)$**:** *A deterministic algorithm that on input a security parameter $\lambda \in \mathbb{N}$, a difficulty parameter $t \in \mathbb{N}$, and a puzzle $z \in \{0,1\}^*$, outputs a solution $s \in (\{0,1\}^\lambda \cup \{\bot\})$.*

We require $(\mathsf{Gen}, \mathsf{Sol})$ *to satisfy the following properties.*

- **Correctness:** *For every $\lambda, t \in \mathbb{N}$, solution $s \in \{0,1\}^\lambda$, and $z \in \mathrm{Supp}\big(\mathsf{Gen}(1^\lambda, t, s)\big)$, it holds that $\mathsf{Sol}(1^\lambda, t, z) = s$.*
- **Efficiency:** *There exist a polynomial p such that for all $\lambda, t \in \mathbb{N}$, $\mathsf{Sol}(1^\lambda, t, \cdot)$ is computable in time $t \cdot p(\lambda, \log t)$.*
- B**-Hardness:** *There exists a positive polynomial function α such that for all functions T and non-uniform distinguishers $\mathcal{A} = \{\mathcal{A}_\lambda\}_{\lambda \in \mathbb{N}}$ satisfying $\alpha(\lambda) \leq T(\lambda) \in \mathrm{poly}(\lambda, B(\lambda))$, $\mathsf{size}(\mathcal{A}_\lambda) \in \mathrm{poly}(\lambda, B(\lambda))$, and $\mathsf{depth}(\mathcal{A}_\lambda) \leq T(\lambda)/\alpha(\lambda)$ for all $\lambda \in \mathbb{N}$, there exists a negligible function negl such that for all $\lambda \in \mathbb{N}$, and $s, s' \in \{0,1\}^\lambda$,*

$$\left| \Pr\left[\mathcal{A}_\lambda(\mathsf{Gen}(1^\lambda, T(\lambda), s)) = 1\right] - \Pr\left[\mathcal{A}_\lambda(\mathsf{Gen}(1^\lambda, T(\lambda), s')) = 1\right] \right| \leq \mathsf{negl}(\lambda),$$

  where the probabilities are over the randomness of Gen *and* $\mathcal{A}_\lambda$.

In the above definition, we assume for simplicity that the solutions s are λ-bits long. We can naturally generalize this to consider the case where solutions have some specified length $L(\lambda)$.

Comparison with the definition of [BGJ$^+$16]. We discuss the definition of B-hardness in comparison with the definition given by Bitansky et al. [BGJ$^+$16]. First, they consider only polynomial B, whereas we expand this notion to possibly allow for possibly super-polynomial functions B. Second, their definition only requires that the depth of $\mathcal{A}$ is bounded by T^ϵ for a constant $\epsilon \in (0,1)$. In other words, the adversary has an advantage which depends on T over the running time of the honest evaluator. We instead have a stronger requirement where adversary's advantage α only depends on λ and is independent of T. We remark that one could relax our definition to theirs by allowing α to be a function of T and λ.

3.2 Non-Malleable Time-Lock Puzzles

To formalize non-malleability in the context of time-lock puzzles, we introduce a Man-In-the-Middle (MIM) adversary. Because time-lock puzzles are designed to be broken in some depth t, we restrict our MIM adversary to have at most depth $t/\alpha(\lambda)$ for a function α denoting the advantage of the adversary. Furthermore, we allow for concurrent MIM adversaries that possibly interact with many senders and receivers at the same time.

Definition 3.2 (MIM Adversaries)**.** *Let* $n_L, n_R, B, \alpha, T \colon \mathbb{N} \to \mathbb{N}$. *An* (n_L, n_R, B, α, T)-Man-In-the-Middle (MIM) adversary *is a non-uniform algorithm* $\mathcal{A} = \{\mathcal{A}_\lambda\}_{\lambda\in\mathbb{N}}$ *satisfying* $\mathsf{depth}(\mathcal{A}_\lambda) \le T(\lambda)/\alpha(\lambda)$ *and* $\mathsf{size}(\mathcal{A}_\lambda) \in \mathrm{poly}(\lambda, B(\lambda))$ *for all* $\lambda \in \mathbb{N}$ *that receives* $n_L(\lambda)$ *puzzles on the left and outputs* $n_R(\lambda)$ *puzzles on the right.*

We next define the MIM distribution, which corresponds to the values underlying the puzzles output by the MIM adversary. To capture adversaries that simply forward one of the puzzles on the left to a receiver on the right, we set the value for any forwarded puzzle to be $\perp$.

Definition 3.3 (MIM Distribution)**.** *Let* $n_L, n_R, B, \alpha, T \colon \mathbb{N} \to \mathbb{N}$. *Let* $\mathcal{A} = \{\mathcal{A}_\lambda\}_{\lambda\in\mathbb{N}}$ *be an* (n_L, n_R, B, α, T)*-MIM adversary. For any* $\lambda \in \mathbb{N}$ *and* $\vec{s} = (s_1, \ldots, s_{n_L(\lambda)}) \in (\{0,1\}^\lambda)^{n_L(\lambda)}$*, we define the distribution*

$$(\widetilde{s}_1, \ldots, \widetilde{s}_{n_R(\lambda)}) \leftarrow \mathsf{mim}_{\mathcal{A}}(1^\lambda, T(\lambda), \vec{s})$$

as follows. $\mathcal{A}_\lambda$ *receives puzzles* $z_i \leftarrow \mathsf{Gen}(1^\lambda, T(\lambda), s_i)$ *for all* $i \in [n_L(\lambda)]$ *and outputs puzzles* $(\widetilde{z}_1, \ldots, \widetilde{z}_{n_R(\lambda)})$*. Then for each* $i \in [n_R(\lambda)]$*, we define*

$$\widetilde{s}_i = \begin{cases} \perp & \text{if there exists a } j \in [n_L(\lambda)]) \text{ such that } \widetilde{z}_i = z_j, \\ s & \text{if there exists a unique } s \text{ such that } \widetilde{z}_i = \mathsf{Gen}(1^\lambda, T(\lambda), s; r) \text{ for some } r, \text{ or} \\ \perp & \text{otherwise.} \end{cases}$$

Intuitively, a time-lock puzzle is non-malleable if the MIM distribution of a bounded depth attacker does not depend on the solutions underlying the puzzles it receives on the left. We formalize this definition below.

Definition 3.4 (Concurrent Non-malleable)**.** *Let* $n_L, n_R, B, \alpha, T \colon \mathbb{N} \to \mathbb{N}$. *A* B*-secure time-lock puzzle* $(\mathsf{Gen}, \mathsf{Sol})$ *is* (n_L, n_R)-concurrent non-malleable *if there exist a positive polynomial* α *such that for every function* T *with* $\alpha(\lambda) \le T(\lambda) \in \mathrm{poly}(\lambda, B(\lambda))$ *for all* $\lambda \in \mathbb{N}$ *and every* (n_L, n_R, B, α, T)*-MIM adversary* $\mathcal{A} = \{\mathcal{A}_\lambda\}_{\lambda\in\mathbb{N}}$*, the following holds.*

For any distinguisher $\mathcal{D}$*, there exists a negligible function* negl *such that for all* $\lambda \in \mathbb{N}$ *and* $\vec{s} = (s_1, \ldots, s_{n_L(\lambda)}) \in (\{0,1\}^\lambda)^{n_L(\lambda)}$,

$$\left| \Pr\left[\mathcal{D}(\mathsf{mim}_{\mathcal{A}}(1^\lambda, T(\lambda), \vec{s})) = 1\right] - \Pr\left[\mathcal{D}(\mathsf{mim}_{\mathcal{A}}(1^\lambda, T(\lambda), (0^\lambda)^{n_L(\lambda)})) = 1\right] \right| \le \mathsf{negl}(\lambda).$$

We consider standard variants for the definition of non-malleable above.

Definition 3.5. *Let* $(\mathsf{Gen}, \mathsf{Sol})$ *be a* B*-secure time-lock puzzle. We say the following non-malleability properties are satisfied when Definition 3.4 holds against* (n_L, n_R, B, α, T)*-MIM adversaries for the following settings of* n_L *and* n_R*:*

- fully concurrent non-malleable *if the definition holds against any* $n_L, n_R \in \text{poly}(\lambda)$*,*
- one-many non-malleable *if the definition holds for any* $n_R(\lambda) \in \text{poly}(\lambda)$ *and* $n_L = 1$*,*
- n-concurrent non-malleable *if the definition holds for* $n_L = n_R = n$*,*
- one-n non-malleable *for* $n_L(\lambda) = 1$ *and* $n_R = n$*,*
- *and simply* non-malleable *(not concurrent) for* $n_L(\lambda) = n_R(\lambda) = 1$*.*

3.3 Time-Lock Puzzles in the Auxiliary-Input Random Oracle Model

In the random oracle model [BR93], security is proven in the case where all relevant parties have oracle access to a common random function. For security in the (plain) random oracle model, it is assumed that a fixed adversary is independent of the common random function and must break security with probability over the choice of the random function. Security in the auxiliary-input random oracle model, introduced by [Unr07], is proven assuming that the attacker may depend *arbitrarily* on the random oracle, as long as the attacker is of polynomial (or bounded) size.

One of the main benefits of proving security in the random oracle model is that you can argue security when the random oracle is sampled "lazily." The following lemma, adapted from Unruh [Unr07], shows how we can use similar lazy sampling techniques in the auxiliary-input random oracle model without significant loss in security. At a high level, it says that for any computationally bounded adversary $\mathcal{A}$ in the AI-ROM model, we can "switch" a random $\mathcal{O}$ (that the attacker may depend arbitrarily on) with an oracle $\mathcal{P}$ that is lazily sampled on almost all points. In other words, $\mathcal{A}$ cannot distinguish the output of $\mathcal{O}$ from $\mathcal{P}$ on a *random* query.

Lemma 3.6 (Unruh [Unr07])**.** *For any functions* p, f*,* $\lambda \in \mathbb{N}$*, and unbounded algorithm* Z *that on input a random oracle* $\mathcal{O} \in \mathsf{RF}_\lambda^\lambda$ *outputs* $p(\lambda)$*-size oracle machines, there is an (inefficient) algorithm* Sam *that outputs a partial assignment* F *to* $\{0,1\}^\lambda$ *on* $f(\lambda)$ *points, such that for any* $\lambda \in \mathbb{N}$ *and unbounded distinguisher* $\mathcal{D}$*,*

$$\left| \Pr\left[\begin{array}{l} \mathcal{O} \leftarrow \mathsf{RF}_\lambda^\lambda \\ \mathcal{A} = Z(\mathcal{O}) \\ y \leftarrow \mathcal{A}^{\mathcal{O}} \end{array} : \mathcal{D}(y) = 1 \right] - \Pr\left[\begin{array}{l} \mathcal{O} \leftarrow \mathsf{RF}_\lambda^\lambda \\ \mathcal{A} = Z(\mathcal{O}) \\ F = \mathsf{Sam}(\mathcal{A}) \\ \mathcal{P} \leftarrow \mathsf{RF}_\lambda^\lambda[F] \\ y \leftarrow \mathcal{A}^{\mathcal{P}} \end{array} : \mathcal{D}(y) = 1 \right] \right| \leq \sqrt{\frac{(p(\lambda))^2}{f(\lambda)}}.$$

We note that the above lemma also holds in the case where the random oracle has input and output length are fixed polynomials in λ.

We now formalize non-malleable time-lock puzzles in the auxiliary-input random oracle model. As the AI-ROM only affects the security properties against computationally bounded adversaries, the syntax, correctness, efficiency, and completeness properties are left relatively unchanged. As a result, we focus on the definitions of hardness and non-malleability. In the AI-ROM, we model computationally bounded adversaries that are allowed to depend arbitrarily on the random oracle. To formalize this, we consider inefficient algorithms Z that, for any $\lambda \in \mathbb{N}$, take as input a random oracle $\mathcal{O} \in \mathsf{RF}_\lambda^\lambda$ (of exponential size) and output circuits of bounded size. Additionally, we consider a MIM distribution for a (n, n, B, α, T)-oracle adversary $\mathcal{A}$, $\mathsf{mim}_\mathcal{A}^\mathcal{O}$, where $\mathcal{A}$ and the distribution have oracle access to the same oracle $\mathcal{O}$.

Definition 3.7. *Let $B, n \colon \mathbb{N} \to \mathbb{N}$. A B-secure n-*concurrent non-malleable time-lock puzzle in the AI-ROM *is a tuple of oracle algorithms* $(\mathsf{Gen}, \mathsf{Sol})$ *that satisfy correctness, efficiency, and completeness relative to any $\mathcal{O} \in \mathsf{RF}_\lambda^\lambda$, and the following:*

- **B-Hardness:** *There exists a polynomial α such that for any function T with $\alpha(\lambda) \le T(\lambda) \in \mathrm{poly}(\lambda, B(\lambda))$ for all $\lambda \in \mathbb{N}$ and unbounded algorithm Z that on input $\mathcal{O} \in \mathsf{RF}_\lambda^\lambda$ outputs circuits of size $\mathrm{poly}(\lambda, B(\lambda))$ and depth at most $T(\lambda)/\alpha(\lambda)$, there exists a negligible function* negl *such that for all $\lambda \in \mathbb{N}$ and values $s_0, s_1 \in \{0,1\}^\lambda$,*

$$\left| \Pr\left[\begin{array}{l} \mathcal{O} \leftarrow \mathsf{RF}_\lambda^\lambda \\ \mathcal{A} \leftarrow Z(\mathcal{O}) \\ z \leftarrow \mathsf{Gen}^\mathcal{O}(1^\lambda, T(\lambda), s_0) \end{array} : \mathcal{A}^\mathcal{O}(z) = 1 \right] - \Pr\left[\begin{array}{l} \mathcal{O} \leftarrow \mathsf{RF}_\lambda^\lambda \\ \mathcal{A} = Z(\mathcal{O}) \\ z \leftarrow \mathsf{Gen}^\mathcal{O}(1^\lambda, T(\lambda), s_1) \end{array} : \mathcal{A}^\mathcal{O}(z) = 1 \right] \right| \le \mathsf{negl}(\lambda).$$

- **Concurrent Non-malleable:** *There exist a polynomial α such that for every function T with $\alpha(\lambda) \le T(\lambda) \in \mathrm{poly}(\lambda, B(\lambda))$ for all $\lambda \in \mathbb{N}$, polynomials n_L, n_R, and unbounded algorithm Z that on input $\mathcal{O} \in \mathsf{RF}_\lambda^\lambda$ outputs an (n_L, n_R, B, α, T)-MIM adversary, the following holds.*

  For any distinguisher $\mathcal{D}$, there exists a negligible function negl *such that for all $\lambda \in \mathbb{N}$ and $\vec{s} = (s_1, \ldots, s_{n(\lambda)}) \in (\{0,1\}^\lambda)^{n_L(\lambda)}$,*

$$\left| \Pr\left[\begin{array}{l} \mathcal{O} \leftarrow \mathsf{RF}_\lambda^\lambda \\ \mathcal{A} = Z(\mathcal{O}) \\ \vec{\tilde{s}} \leftarrow \mathsf{mim}_\mathcal{A}^\mathcal{O}(1^\lambda, T(\lambda), \vec{s}) \end{array} : \mathcal{D}(\vec{\tilde{s}}) = 1 \right] - \Pr\left[\begin{array}{l} \mathcal{O} \leftarrow \mathsf{RF}_\lambda^\lambda \\ \mathcal{A} \leftarrow Z(\mathcal{O}) \\ \vec{\tilde{s}} \leftarrow \mathsf{mim}_\mathcal{A}^\mathcal{O}(1^\lambda, T(\lambda), (0^\lambda)^{n(\lambda)}) \end{array} : \mathcal{D}(\vec{\tilde{s}}) = 1 \right] \right| \le \mathsf{negl}(\lambda).$$

We note that the above definitions can also be naturally extended to hold relative to random oracles with input and output length that are fixed polynomials in λ.

4 Non-Malleable Time-Lock Puzzles

In Section 4.1, we define the notion of concurrent functional non-malleability for a class of functions $\mathcal{F}$. Then, in Section 4.2, we give a transformation from any time-lock puzzle to one that satisfies concurrent functional non-malleable for depth-bounded functions $\mathcal{F}$, in the auxiliary-input random oracle model. The security of our construction depends on the output length of the functions $f \in \mathcal{F}$. We discuss how this general result for concurrent functional non-malleability implies our result for bounded concurrent (standard) non-malleability. In Section 4.4, we show that no time-lock puzzle satisfies fully concurrent (standard) non-malleability.

4.1 Functional Non-Malleable Time-Lock Puzzles

We formally define concurrent *functional* non-malleability. For simplicity, we define it in the case of unbounded concurrency, but we note that it can be defined for restricted cases as in Section 3.2.

Definition 4.1 (Concurrent Functional Non-malleable)**.** *Let $B, L\colon \mathbb{N} \to \mathbb{N}$, and* $(\mathsf{Gen}, \mathsf{Sol})$ *be a B-secure time-lock puzzle for messages of length $L(\lambda)$. Let $\mathcal{F}$ be a class of functions of the form $f\colon (\{0,1\}^{L(\lambda)})^* \to \{0,1\}^*$. We say that* $(\mathsf{Gen}, \mathsf{Sol})$ *is* concurrent functional non-malleable *for $\mathcal{F}$ if for any function $f \in \mathcal{F}$ and polynomial n, there exists a polynomial α such that for every function T with $\alpha(\lambda) \le T(\lambda) \in \mathrm{poly}(\lambda, B(\lambda))$ for all $\lambda \in \mathbb{N}$, every (n, n, B, α, T)-MIM adversary $\mathcal{A} = \{\mathcal{A}_\lambda\}_{\lambda \in \mathbb{N}}$, the following holds.*

For any distinguisher $\mathcal{D}$, there exists a negligible function negl *such that for all $\lambda \in \mathbb{N}$ and $\vec{s} = (s_1, \ldots, s_{n(\lambda)}) \in (\{0,1\}^{L(\lambda)})^{n(\lambda)}$,*

$$\Big|\Pr\left[\vec{\tilde{s}} \leftarrow \mathsf{mim}_{\mathcal{A}}(1^\lambda, T(\lambda), \vec{s}) : \mathcal{D}(f(\vec{\tilde{s}})) = 1\right] \\ - \Pr\left[\vec{\tilde{s}} \leftarrow \mathsf{mim}_{\mathcal{A}}(1^\lambda, T(\lambda), (0^{L(\lambda)})^{n(\lambda)}) : \mathcal{D}(f(\vec{\tilde{s}})) = 1\right]\Big| \le \mathsf{negl}(\lambda).$$

We note that functional non-malleability for a class $\mathcal{F}$ that contains the identity function id implies standard non-malleability as $\mathcal{D}(\mathsf{id}(\vec{\tilde{s}})) = \mathcal{D}(\vec{\tilde{s}})$.

4.2 Non-Malleable Time-Lock Puzzles Construction

In this section, we give our construction of a fully concurrent functional non-malleable time-lock puzzle for functions with bounded depth and output length. We rely on the following building blocks and parameters.

- A function m denoting the output length for our function non-malleability. We require $m(\lambda) \in \lambda^{O(1)}$. Throughout this section, where λ is clear from context, we let $m = m(\lambda)$.
- A B-secure time-lock puzzle $\mathsf{TLP} = (\mathsf{Gen}_{\mathsf{tlp}}, \mathsf{Sol}_{\mathsf{tlp}})$. We let $\lambda_{\mathsf{tlp}} = \lambda_{\mathsf{tlp}}(\lambda) = m + \lambda$ be the bits of randomness needed for TLP on security parameter λ, and we let $\lambda + \lambda_{\mathsf{tlp}}$ be the length of the solutions for TLP. We require that the time-lock puzzle satisfies $B(\lambda) \in 2^m \cdot \mathrm{poly}(\lambda)$. We note that our function Gen defined below will also require λ_{tlp} bits of randomness.
- A class of functions $\mathcal{F}_{B,m}$ of the form $f\colon (\{0,1\}^\lambda)^* \to \{0,1\}^{m(\lambda)}$. We assume that there exists a polynomial d such that for every polynomial n, every function $f \in \mathcal{F}_{B,m}$ can be computed in depth $d(\lambda, \log n(\lambda))$ and size $\mathrm{poly}(\lambda, B(\lambda))$ on inputs of length at most $\lambda \cdot n(\lambda)$.
- A random oracle $\mathcal{O}$, where $\mathcal{O}$ on input $(s, r) \in \{0,1\}^{\lambda + \lambda_{\mathsf{tlp}}}$ outputs a random value $r' \in \{0,1\}^{\lambda_{\mathsf{tlp}}}$. Note that by definition of λ_{tlp}, the random oracle $\mathcal{O}$ is in the set $\mathsf{RF}^{\lambda+m}_{2\lambda+m}$.

Our construction $\mathsf{nmTLP}_m = (\mathsf{Gen}, \mathsf{Sol})$ in the auxiliary-input random oracle model:

- $z = \mathsf{Gen}^{\mathcal{O}}(1^\lambda, t, s; r)$:
  1. Get $r' = \mathcal{O}(s, r)$.
  2. Output $z = \mathsf{Gen}_{\mathsf{tlp}}(1^\lambda, t, (s||r); r')$.
- $s = \mathsf{Sol}^{\mathcal{O}}(1^\lambda, t, z)$:
  1. Compute $s' = \mathsf{Sol}_{\mathsf{tlp}}(1^\lambda, t, z)$ and parse $s' = s||r$.

2. If $z = \mathsf{Gen}^{\mathcal{O}}(1^\lambda, t, s; r)$, output s.
3. If not, output $\perp$.

Theorem 4.2 (Fully Concurrent Functional Non-Malleable TLPs)**.** *Let $B\colon \mathbb{N} \to \mathbb{N}$ and $m(\lambda) \in \lambda^{O(1)}$, where $B(\lambda) \in 2^m \cdot \mathrm{poly}(\lambda)$. Assuming* TLP *is a B-secure time-lock puzzle, then* nmTLP_m *is a B-secure fully concurrent functional non-malleable time-lock puzzle in the AI-ROM for the class of functions $\mathcal{F}_{B,m}$.*

We provide the proof of this theorem in Section 4.3 below. Before doing so, we observe the following corollaries to the above theorem:

- If $m(\lambda) \in O(\log(\lambda))$ then we can simply assume a polynomially-secure TLP.
- For any $m(\lambda) \in \mathrm{poly}(\lambda)$, then our theorem follows by assuming a sub-exponentially secure TLP. Specifically, it suffices that there exists a constant $\gamma \in (0,1)$ such that $B(\lambda) = 2^{\lambda^\gamma}$, and we can instantiate this with $\lambda_{\mathsf{tlp}} = (\lambda \cdot m(\lambda))^{1/\gamma}$ bits of randomness.

We also observe that the above theorem can be used to get n-bounded concurrency for any polynomial n, simply by setting the output length m of the functions in $\mathcal{F}_{B,m}$ to $\lambda \cdot n(\lambda)$. Specifically, let f_{id} be the identity function with input and output length $\lambda \cdot n(\lambda)$. Since $f_{\mathsf{id}} \in \mathcal{F}_{B,\lambda \cdot n(\lambda)}$, a fully concurrent functional non-malleable TLP for $\mathcal{F}_{B,\lambda \cdot n(\lambda)}$ implies an n-concurrent non-malleable TLP, which gives the following corollary.

Corollary 4.3 (n-Concurrent Non-Malleable TLPs)**.** *Let $B\colon \mathbb{N} \to \mathbb{N}$ and $n(\lambda) \in \lambda^{O(1)}$, where $B(\lambda) \in 2^{\lambda \cdot n(\lambda)} \cdot \mathrm{poly}(\lambda)$. Let $m(\lambda) = \lambda \cdot n(\lambda)$. Assuming* TLP *is a B-secure time-lock puzzle, then* nmTLP_m *is a B-secure n-concurrent non-malleable time-lock puzzle in the AI-ROM.*

4.3 Proof of Concurrent Functional Non-Malleability

We prove Theorem 4.2 by showing correctness in Lemma 4.4, efficiency in Lemma 4.5, hardness in Lemma 4.6, and non-malleability in Lemma 4.11.

Lemma 4.4 (Correctness)**.** *Assuming* $(\mathsf{Gen}_{\mathsf{tlp}}, \mathsf{Sol}_{\mathsf{tlp}})$ *satisfies correctness, then* $(\mathsf{Gen}, \mathsf{Sol})$ *satisfies correctness.*

Proof. Fix any $\lambda, t \in \mathbb{N}$ and $\mathcal{O}$ in $\mathsf{RF}^{\lambda+m}_{2\lambda+m}$. We want to show that for every $z = \mathsf{Gen}^{\mathcal{O}}(1^\lambda, t, s; r)$ for some $s \in \{0,1\}^\lambda$ and $r \in \{0,1\}^{\lambda_{\mathsf{tlp}}}$, it holds that $\mathsf{Sol}^{\mathcal{O}}(1^\lambda, t, z) = s$.

This follows from correctness of TLP. Recall that the algorithm $\mathsf{Sol}^{\mathcal{O}}(1^\lambda, t, z)$ runs $\widehat{s}||\widehat{r} = \mathsf{Sol}_{\mathsf{tlp}}(1^\lambda, t, z)$ and outputs $\widehat{s}$ if $z = \mathsf{Gen}^{\mathcal{O}}(1^\lambda, t, \widehat{s}; \widehat{r})$. Since we assumed z was in the support of $\mathsf{Gen}^{\mathcal{O}}$, it follows by definition of $\mathsf{Gen}^{\mathcal{O}}$ that $z = \mathsf{Gen}_{\mathsf{tlp}}(1^\lambda, t, (s||r); r')$ for some s, r, r'. By correctness of TLP, this implies that $\mathsf{Sol}_{\mathsf{tlp}}(1^\lambda, t, z) = s||r$ so $s||r = \widehat{s}||\widehat{r}$ and Sol outputs the correct solution. □

Lemma 4.5 (Efficiency)**.** *Assuming that* $(\mathsf{Gen}_{\mathsf{tlp}}, \mathsf{Sol}_{\mathsf{tlp}})$ *satisfies efficiency, then* $(\mathsf{Gen}, \mathsf{Sol})$ *satisfies efficiency.*

Proof. Fix any $\lambda, t \in \mathbb{N}$ and $\mathcal{O} \in \mathsf{RF}^{\lambda+m}_{2\lambda+m}$. First, we show that $\mathsf{Gen}^{\mathcal{O}}$ is PPT. We have that $\mathsf{Gen}^{\mathcal{O}}(1^\lambda, t, \cdot)$ makes a single oracle call to $\mathcal{O}$, which takes time $\lambda_{\mathsf{tlp}} = \lambda + m$ to read the output, and evaluates $\mathsf{Gen}_{\mathsf{tlp}}(1^\lambda, t, \cdot)$, which takes time $\mathrm{poly}(\lambda, \log t)$. It follows that there exists a polynomial p_1 such that $\mathsf{Gen}^{\mathcal{O}}(1^\lambda, t, \cdot)$ can be computed in time $p_1(\lambda, \log t)$.

For $\mathsf{Sol}^{\mathcal{O}}$, recall that $\mathsf{Sol}^{\mathcal{O}}(1^\lambda, t, \cdot)$ runs $\mathsf{Sol}_{\mathsf{tlp}}(1^\lambda, t, \cdot)$ and $\mathsf{Gen}^{\mathcal{O}}(1^\lambda, t, \cdot)$. By the above, $\mathsf{Gen}^{\mathcal{O}}$ can be run in time $p_1(\lambda, \log t)$. For $\mathsf{Sol}_{\mathsf{tlp}}$, recall that by the efficiency of TLP there exists a polynomial p_2 such that for all $\lambda, t \in \mathbb{N}$, $\mathsf{Sol}_{\mathsf{tlp}}(1^\lambda, t, \cdot)$ can be computed in time $t \cdot p_2(\lambda, \log t)$. Putting these together, we have that $\mathsf{Sol}^{\mathcal{O}}(1^\lambda, t, \cdot)$ can be computed in time $p_1(\lambda, \log t) + t \cdot p_2(\lambda, \log t)$. □

Lemma 4.6 (Hardness). *Assuming that* $(\mathsf{Gen}_{\mathsf{tlp}}, \mathsf{Sol}_{\mathsf{tlp}})$ *satisfies* B*-hardness, then* $(\mathsf{Gen}, \mathsf{Sol})$ *satisfies* B*-hardness in the AI-ROM.*

Proof. To show hardness, suppose for contradiction that for all polynomials α, there exists a function T with $\alpha(\lambda) \leq T(\lambda) \in \text{poly}(\lambda, B(\lambda))$ for all $\lambda \in \mathbb{N}$, an unbounded algorithm Z that outputs circuits of size $\text{poly}(\lambda, B(\lambda))$ and depth at most $T(\lambda)/\alpha(\lambda)$, and a polynomial q such that for infinitely many $\lambda \in \mathbb{N}$, there exist values $s_0, s_1 \in \{0,1\}^\lambda$ such that

$$\left| \Pr\left[\begin{array}{l} \mathcal{O} \leftarrow \mathsf{RF}^{\lambda+m}_{2\lambda+m} \\ \mathcal{D} = Z(\mathcal{O}) \\ z \leftarrow \mathsf{Gen}^{\mathcal{O}}(1^\lambda, T(\lambda), s_0) \end{array} : \mathcal{D}^{\mathcal{O}}(z) = 1 \right] - \Pr\left[\begin{array}{l} \mathcal{O} \leftarrow \mathsf{RF}^{\lambda+m}_{2\lambda+m} \\ \mathcal{D} = Z(\mathcal{O}) \\ z \leftarrow \mathsf{Gen}^{\mathcal{O}}(1^\lambda, T(\lambda), s_1) \end{array} : \mathcal{D}^{\mathcal{O}}(z) = 1 \right] \right| \geq \frac{1}{q(\lambda)}. \tag{4.1}$$

As the above holds for all α by assumption, we will give a specific polynomial α and use it to reach a contradiction. Specifically, let α_{tlp} be the polynomial guaranteed by the hardness of TLP. We will show a contradiction for $\alpha(\lambda) = \alpha_{\mathsf{tlp}}(\lambda) \cdot p(\lambda)$, where p is a fixed polynomial specified in the proof of Claim 4.10. To derive a contradiction, we will define a sequence of hybrid experiments, and we will use the fact that the statistical distance between the above probabilities is noticeable to construct an adversary that breaks the hiding of TLP. For any $\lambda \in \mathbb{N}$ and $s \in \{0,1\}^\lambda$, we define the following sequence of hybrid experiments. Throughout these hybrids, we let $t = T(\lambda)$.

- $\mathsf{Hyb}^s_0(\lambda)$: This hybrid is equivalent to the terms in the probabilities above for a puzzle generated with solution $s \in \{0,1\}^\lambda$, where $\mathsf{Gen}^{\mathcal{O}}$ is written out explicitly.

$$\mathsf{Hyb}^s_0(\lambda) = \left\{ \begin{array}{ll} \mathcal{O} \leftarrow \mathsf{RF}^{\lambda+m}_{2\lambda+m}; & \mathcal{D} = Z(\mathcal{O}) \\ r \leftarrow \{0,1\}^{\lambda_{\mathsf{tlp}}}; & r' = \mathcal{O}(s, r) \\ z = \mathsf{Gen}_{\mathsf{tlp}}(1^\lambda, t, (s \| r); r') & \end{array} : \mathcal{D}^{\mathcal{O}}(z) = 1 \right\}$$

- $\mathsf{Hyb}^s_1(\lambda)$: Let Z' be the algorithm such that for any $\mathcal{O} \in \mathsf{RF}^{\lambda+m}_{2\lambda+m}$ and $\mathcal{D} = Z(\mathcal{O})$, the algorithm $Z'(\mathcal{D})$ outputs an oracle algorithm $\mathcal{A}^{\mathcal{O}}$ which does the following:

  1. Sample $r \leftarrow \{0,1\}^{\lambda_{\mathsf{tlp}}}$ and query $r' = \mathcal{O}(s, r)$.
  2. Compute $z = \mathsf{Gen}_{\mathsf{tlp}}(1^\lambda, t, (s \| r); r')$.
  3. Output $\mathcal{D}^{\mathcal{O}}(z)$.

  This hybrid uses $\mathcal{A}$ to determine the output of the experiment.

$$\mathsf{Hyb}^s_1(\lambda) = \left\{ \mathcal{O} \leftarrow \mathsf{RF}^{\lambda+m}_{2\lambda+m}; \quad \mathcal{D} = Z(\mathcal{O}); \quad \mathcal{A} = Z'(\mathcal{D}) \; : \mathcal{A}^{\mathcal{O}} = 1 \right\}$$

  Note that $\mathsf{size}(\mathcal{A}) \in \text{poly}(\lambda, 2^m)$, which holds by the efficiency of Gen and because $\lambda_{\mathsf{tlp}} = m + \lambda$ and $\mathsf{size}(\mathcal{D}) \in \text{poly}(\lambda, B(\lambda))$.

- $\mathsf{Hyb}^s_2(\lambda)$: Let $q_{\mathcal{A}}$ be the polynomial such that $\mathsf{size}(\mathcal{A}) = q_{\mathcal{A}}(\lambda, 2^m)$. Let $f_{\mathsf{hard}}(\lambda) = (q_{\mathcal{A}}(\lambda, 2^m) \cdot 4q(\lambda))^2$, and let Sam be the inefficient algorithm from Lemma 3.6 that on input an adversary $\mathcal{A}$ outputs a partial assignment F on $f_{\mathsf{hard}}(\lambda)$ points. This hybrid swaps $\mathcal{O}$ with a random function $\mathcal{P}$ fixed at the set of points F determined by Sam.

$$\mathsf{Hyb}^s_2(\lambda) = \left\{ \begin{array}{lll} \mathcal{O} \leftarrow \mathsf{RF}^{\lambda+m}_{2\lambda+m}; & \mathcal{D} = Z(\mathcal{O}); & \mathcal{A} = Z'(\mathcal{D}) \\ F \leftarrow \mathsf{Sam}(\mathcal{A}); & \mathcal{P} \leftarrow \mathsf{RF}^{\lambda+m}_{2\lambda+m}[F] & \end{array} : \mathcal{A}^{\mathcal{P}} = 1 \right\}$$

- $\mathsf{Hyb}_3^s(\lambda)$: This hybrid samples r' uniformly randomly from $\{0,1\}^{\lambda_{\mathsf{tlp}}}$, rather than using $\mathcal{P}$ to compute it. It also gives $\mathcal{D}$ oracle access to a modified version of $\mathcal{P}$, where on input (s,r) it outputs r', and agrees with $\mathcal{P}$ on all other inputs. We denote this oracle by $\mathcal{P}[(s,r) \to r']$. In this hybrid, we additionally write out $\mathcal{A}$'s actions explicitly.

$$\mathsf{Hyb}_3^s(\lambda) = \left\{ \begin{array}{l} \mathcal{O} \leftarrow \mathsf{RF}_{2\lambda+m}^{\lambda+m}; \quad \mathcal{D} = Z(\mathcal{O}); \quad \mathcal{A} = Z'(\mathcal{D}) \\ F \leftarrow \mathsf{Sam}(\mathcal{A}); \quad \mathcal{P} \leftarrow \mathsf{RF}_{2\lambda+m}^{\lambda+m}[F] \\ r \leftarrow \{0,1\}^{\lambda_{\mathsf{tlp}}}; \quad r' \leftarrow \{0,1\}^{\lambda_{\mathsf{tlp}}} \\ z = \mathsf{Gen}_{\mathsf{tlp}}(1^\lambda, t, (s\|r); r') \end{array} : \mathcal{D}^{\mathcal{P}[(s,r)\to r']}(z) = 1 \right\}$$

To conclude the proof, fix any $\lambda \in \mathbb{N}$ and values $s_0, s_1 \in \{0,1\}^\lambda$ for which Equation 4.1 holds. Since $\mathsf{Hyb}_0^{s_b}(\lambda)$ is equal to the probability in Equation 4.1 for value s_b, we get that

$$|\mathsf{Hyb}_0^{s_0}(\lambda) - \mathsf{Hyb}_0^{s_1}(\lambda)| \geq \frac{1}{q(\lambda)}.$$

In the following claims, we bound the distance between each pair of consecutive hybrids. Specifically, in Claim 4.7 we show that $\Pr[\mathsf{Hyb}_0^s(\lambda)] = \Pr[\mathsf{Hyb}_1^s(\lambda)]$ for all $s \in \{0,1\}^\lambda$. Then, in Claim 4.8, we bound the statistical distance between $\mathsf{Hyb}_1^s(\lambda)$ and $\mathsf{Hyb}_2^s(\lambda)$ by $\frac{1}{4q(\lambda)}$ for all $s \in \{0,1\}^\lambda$. Then, we show in Claim 4.9 that there exists a negligible function negl such that $|\Pr[\mathsf{Hyb}_2^s(\lambda)] - \Pr[\mathsf{Hyb}_3^s(\lambda)]| \leq \mathsf{negl}(\lambda)$ for all $s \in \{0,1\}^\lambda$. Given these claims, it follows that the statistical distance between $\mathsf{Hyb}_0^{s_b}(\lambda)$ and $\mathsf{Hyb}_3^{s_b}(\lambda)$ is at most $1/(2q(\lambda)) + 2\mathsf{negl}(\lambda)$ for $b \in \{0,1\}$. By combining this with the above, it holds that $|\Pr[\mathsf{Hyb}_3^{s_0}(\lambda)] - \Pr[\mathsf{Hyb}_3^{s_1}(\lambda)]| \geq \frac{1}{4q(\lambda)}$ for infinitely many $\lambda \in \mathbb{N}$. To conclude the proof, we show in Claim 4.10 that this implies that there exists a negligible function negl and an adversary that breaks the hardness of TLP with probability $1/(4q(\lambda)) - \mathsf{negl}(\lambda) \geq 1/8q(\lambda)$, in contradiction.

Claim 4.7. *For all $s \in \{0,1\}^\lambda$, $\Pr[\mathsf{Hyb}_0^s(\lambda)] = \Pr[\mathsf{Hyb}_1^s(\lambda)]$.*

Proof. This follows immediately from the definition of $\mathcal{A}$. Specifically, in $\mathsf{Hyb}_1^s(\lambda)$, the algorithm $\mathcal{A}$ samples r, r', and z exactly as done in $\mathsf{Hyb}_0^s(\lambda)$, and then outputs 1 exactly in the event that $\mathsf{Hyb}_0^s(\lambda)$ holds. ∎

Claim 4.8. *For all $s \in \{0,1\}^\lambda$,*

$$|\Pr[\mathsf{Hyb}_1^s(\lambda)] - \Pr[\mathsf{Hyb}_2^s(\lambda)]| \leq 1/(4q(\lambda)).$$

Proof. Recall that $\mathsf{size}(\mathcal{A}) = q_{\mathcal{A}}(\lambda, 2^m)$. We can therefore apply Lemma 3.6 (by viewing Z and Z' as a single sampling algorithm outputting $\mathcal{A}$ based on $\mathcal{O}$). Therefore, the statistical distance between the output of $\mathcal{A}^{\mathcal{O}}$ and $\mathcal{A}^{\mathcal{P}}$ is at most

$$\sqrt{\frac{(q_{\mathcal{A}}(\lambda, 2^m))^2}{f_{\mathsf{hard}}(\lambda)}} = \sqrt{\frac{(q_{\mathcal{A}}(\lambda, 2^m))^2}{(q_{\mathcal{A}}(\lambda, 2^m) \cdot 4q(\lambda))^2}} = \frac{1}{4q(\lambda)},$$

which implies the claim. ∎

Claim 4.9. *There exists a negligible function negl_2 such that for all $s \in \{0,1\}^\lambda$,*

$$|\Pr[\mathsf{Hyb}_2^s(\lambda)] - \Pr[\mathsf{Hyb}_3^s(\lambda)]| \leq \mathsf{negl}_2(\lambda).$$

Proof. We start by using the definition of $\mathcal{A}$ to rewrite $\mathsf{Hyb}_2^s(\lambda)$ as

$$\mathsf{Hyb}_2^s(\lambda) = \left\{ \begin{array}{ll} \mathcal{O} \leftarrow \mathsf{RF}_{2\lambda+m}^{\lambda+m}; \quad \mathcal{D} = Z(\mathcal{O}); \quad \mathcal{A} = Z'(\mathcal{D}) & \\ F \leftarrow \mathsf{Sam}(\mathcal{A}); \quad \mathcal{P} \leftarrow \mathsf{RF}_{2\lambda+m}^{\lambda+m}[F] & \\ r \leftarrow \{0,1\}^{\lambda_{\mathsf{tlp}}}; \quad r' = \mathcal{P}(s,r) & : \ \mathcal{D}^{\mathcal{P}}(z) = 1 \\ z = \mathsf{Gen}(1^\lambda, t, (s\|r); r') & \end{array} \right\}.$$

The difference between this and $\mathsf{Hyb}_3^s(\lambda)$ is that r' is sampled uniformly at random in $\mathsf{Hyb}_3^s(\lambda)$, and $\mathcal{D}$ has oracle access to $\mathcal{P}[(s,r) \to r']$ rather than $\mathcal{P}$. Let us denote this oracle by $\mathcal{P}'$. We will show that these hybrids occur with the same probability, except in the case that the query to $\mathcal{P}$ in $\mathsf{Hyb}_2^s(\lambda)$ (which results in r') appears in F, where we recall that F is the partial assignment on $f(\lambda)$ points given by $\mathsf{Sam}(\mathcal{A})$.

To show this, let E be the event that $(s,r) \notin F$. When E holds, then the distribution of $(\mathcal{P}, r, r')$ in $\mathsf{Hyb}_2^s(\lambda)$ is identical to that of $(\mathcal{P}', r, r')$ in $\mathsf{Hyb}_3^s(\lambda)$. Namely, in both hybrids r is uniformly sampled from $\{0,1\}^{\lambda_{\mathsf{tlp}}}$. For the oracles, in $\mathsf{Hyb}_2^s(\lambda)$, $\mathcal{P}$ is sampled uniformly from $\mathsf{RF}_{2\lambda+m}^{\lambda+m}[F]$ and $r' = \mathcal{P}(s,r)$, whereas in $\mathsf{Hyb}_3^s(\lambda)$, $\mathcal{P}'$ can be sampled according to $\mathsf{RF}_{2\lambda+m}^{\lambda+m}[F]$ for all points except r, and then lazily evaluated it on r to obtain r'. Note that this relies on E holding, since otherwise $\mathcal{P}'$ on input r would be determined by F. As $\mathcal{D}$ has oracle access to the oracle ($\mathcal{P}$ or $\mathcal{P}'$), and receives z as input which is fully determined by r, r', it follows that for all $\lambda \in \mathbb{N}$,

$$\Pr\left[\mathsf{Hyb}_1^s(\lambda) \mid \mathsf{E}\right] = \Pr\left[\mathsf{Hyb}_2^s(\lambda) \mid \mathsf{E}\right].$$

Furthermore, since $r \leftarrow \{0,1\}^{\lambda_{\mathsf{tlp}}}$, the probability that E fails to hold, i.e. that $(s,r) \in F$, is at most

$$\frac{f_{\mathsf{hard}}(\lambda)}{2^{\lambda_{\mathsf{tlp}}}} = \frac{(q_{\mathcal{A}}(\lambda, 2^m) \cdot 4q(\lambda))^2}{2^{\lambda_{\mathsf{tlp}}}} = \frac{(q_{\mathcal{A}}(\lambda, 2^m) \cdot 4q(\lambda))^2}{2^{m+\lambda}} \in \frac{2^{O(m)} \cdot \mathrm{poly}(\lambda)}{2^{m+\lambda}} = \mathsf{negl}_2(\lambda),$$

which is negligible, where we used the fact that $\lambda_{\mathsf{tlp}} = m + \lambda$. So, it holds that for all $\lambda \in \mathbb{N}$, $\Pr\left[\neg\mathsf{E}\right] = \mathsf{negl}_2(\lambda)$. Putting these together, we conclude that for all $\lambda \in \mathbb{N}$,

$$\begin{aligned} &|\Pr[\mathsf{Hyb}_1^s(\lambda)] - \Pr[\mathsf{Hyb}_2^s(\lambda)]| \\ &= |\Pr[\mathsf{E}] \cdot (\Pr[\mathsf{Hyb}_1^s(\lambda) \mid \mathsf{E}] - \Pr[\mathsf{Hyb}_2^s(\lambda) \mid \mathsf{E}]) + \Pr[\neg\mathsf{E}] \cdot (\Pr[\mathsf{Hyb}_1^s(\lambda) \mid \neg\mathsf{E}] - \Pr[\mathsf{Hyb}_2^s(\lambda) \mid \neg\mathsf{E}])| \\ &\le 1 \cdot 0 + \mathsf{negl}_2(\lambda) \cdot 1 = \mathsf{negl}_2(\lambda), \end{aligned}$$

so the claim follows. ■

Claim 4.10. *If there exists function μ such that for infinitely many $\lambda \in \mathbb{N}$ and values $s_0, s_1 \in \{0,1\}^\lambda$, it holds that*

$$|\Pr[\mathsf{Hyb}_3^{s_0}(\lambda)] - \Pr[\mathsf{Hyb}_3^{s_1}(\lambda)]| \ge \mu(\lambda),$$

then there exists a negligible function negl *and an adversary that breaks the hardness of* TLP *with probability $\mu(\lambda) - 2\mathsf{negl}(\lambda)$.*

Proof. We first note that by an averaging argument, the inequality in the statement of the claim implies that for infinitely many $\lambda \in \mathbb{N}$ there exists an $\mathcal{O} \in \mathsf{RF}_{2\lambda+m}^{\lambda+m}$ and $F \in \mathrm{Supp}(\mathsf{Sam}(\mathcal{A}))$, where $\mathcal{D} = Z(\mathcal{O})$, $\mathcal{A} = Z'(\mathcal{D})$ such that

$$\left| \Pr\left[\begin{array}{l} \mathcal{P} \leftarrow \mathsf{RF}_{2\lambda+m}^{\lambda+m}[F] \\ r_0, r_0' \leftarrow \{0,1\}^{\lambda_{\mathsf{tlp}}} \\ z_0 = \mathsf{Gen}_{\mathsf{tlp}}(1^\lambda, t, (s_0\|r_0); r_0') \end{array} : \mathcal{D}^{\mathcal{P}[(s_0,r_0)\to r_0']}(z_0) = 1 \right] \right.$$
$$\left. - \Pr\left[\begin{array}{l} \mathcal{P} \leftarrow \mathsf{RF}_{2\lambda+m}^{\lambda+m}[F] \\ r_1, r_1' \leftarrow \{0,1\}^{\lambda_{\mathsf{tlp}}} \\ z_1 = \mathsf{Gen}_{\mathsf{tlp}}(1^\lambda, t, (s_1\|r_1); r_1') \end{array} : \mathcal{D}^{\mathcal{P}[(s_1,r_1)\to r_1']}(z_1) = 1 \right] \right| \ge \mu(\lambda).$$

The above implies that

$$\Pr\left[\begin{array}{l} \mathcal{P} \leftarrow \mathsf{RF}_{2\lambda+m}^{\lambda+m}[F] \\ b \leftarrow \{0,1\} \\ r_b, r'_b \leftarrow \{0,1\}^{\lambda_{\mathsf{tlp}}} \\ z_b = \mathsf{Gen}_{\mathsf{tlp}}(1^\lambda, t, (s_b \| r_b); r'_b) \end{array} : \mathcal{D}^{\mathcal{P}[(s_b,r_b)\to r'_b]}(z_b) = b\right] \geq \frac{1}{2} + \frac{\mu(\lambda)}{2}. \tag{4.2}$$

Note that $\mathcal{D}$'s success may depend on it's ability to "hit" the randomness r_b in one of its oracle queries, since in this case, it can trivially check if z_b corresponds to a puzzle for $s_b||r_b$. Looking ahead, we will be fixing values of r_0, r_1 and using $\mathcal{D}$ to try to distinguish a puzzle corresponding to $s_0||r_0$ from a puzzle corresponding to $s_1||r_1$, based on which values of r_b he queries. Therefore, it will also be important to consider the event that $\mathcal{D}(z_b)$ makes a query corresponding to r_{1-b}. Next, we formally define these two events:

- Let $\mathsf{hit} = \mathsf{hit}(\mathcal{D}, \mathcal{P}, z)$ denote the event that $\mathcal{D}$ on input a puzzle corresponding to $(s||r)$ queries (s, r). Note that in the case that hit does not occur, then the answers to the oracle queries made by $\mathcal{D}$ are distributed according to $\mathcal{P}$.
- Let $\mathsf{bad} = \mathsf{bad}(\mathcal{D}, \mathcal{P}, z_b, r_b, r_{1-b})$ denote the event that $\mathcal{D}(z_b)$ queries (s_{1-b}, r_{1-b}), where z_b is generated using r_b.

We note that for a uniformly random value of r_{1-b}, it holds that

$$\Pr[\mathsf{bad}] \in \frac{\mathrm{poly}(\lambda, B(\lambda))}{2^{|r_{1-b}|}} \in \frac{2^{O(m)} \cdot \mathrm{poly}(\lambda)}{2^{\lambda_{\mathsf{tlp}}}} \in \frac{2^{O(m)} \cdot \mathrm{poly}(\lambda)}{2^{m+\lambda}} \leq \mathsf{negl}(\lambda),$$

for all $\lambda \in \mathbb{N}$, where the probability is over $\mathcal{P}$, r_0, r_1, z, b, and the randomness of $\mathcal{D}$. This holds because $\mathcal{D}$ has size $\mathrm{poly}(\lambda, m, B(\lambda)) \in \mathrm{poly}(\lambda, 2^m)$, and r_{1-b} is independent of $\mathcal{D}$'s input and the answers to its queries. Combining this with Equation 4.2, we get the following, where all probabilities are over the choice of $\mathcal{P} \leftarrow \mathsf{RF}_{2\lambda+m}^{\lambda+m}[F]$, $b \leftarrow \{0,1\}$, $r_0, r_1 \leftarrow \{0,1\}^{\lambda_{\mathsf{tlp}}}$, and $z_b \leftarrow \mathsf{Gen}_{\mathsf{tlp}}(1^\lambda, t, (s_b||r_b))$:

$$\begin{aligned}
\frac{1}{2} + \frac{\mu(\lambda)}{2} &\leq \Pr\left[\mathcal{D}^{\mathcal{P}[(s_b,r_b)\to r'_b]}(z_b) = b\right] \\
&\leq \Pr\left[\mathcal{D}^{\mathcal{P}[(s_b,r_b)\to r'_b]}(z_b) = b \ \wedge\ \overline{\mathsf{bad}}\right] + \Pr[\mathsf{bad}] \\
&\leq \Pr\left[\mathcal{D}^{\mathcal{P}[(s_b,r_b)\to r'_b]}(z_b) = b \ \wedge\ \overline{\mathsf{hit}} \ \wedge\ \overline{\mathsf{bad}}\right] + \Pr\left[\mathsf{hit} \ \wedge\ \overline{\mathsf{bad}}\right] + \mathsf{negl}(\lambda) \\
&= \Pr\left[\mathcal{D}^{\mathcal{P}}(z_b) = b \ \wedge\ \overline{\mathsf{hit}} \ \wedge\ \overline{\mathsf{bad}}\right] + \Pr\left[\mathsf{hit} \ \wedge\ \overline{\mathsf{bad}}\right] + \mathsf{negl}(\lambda),
\end{aligned}$$

where in the last line we used the fact that when $\overline{\mathsf{hit}}$ occurs, the answers to $\mathcal{D}$'s queries are distributed according to $\mathcal{P}$.

By an averaging argument, there exist *fixed* values r_0, r_1 such that the above holds for those fixed values, namely,

$$\Pr\left[\mathcal{D}^{\mathcal{P}}(z_b) = b \ \wedge\ \overline{\mathsf{hit}} \ \wedge\ \overline{\mathsf{bad}}\right] + \Pr\left[\mathsf{hit} \ \wedge\ \overline{\mathsf{bad}}\right] + \mathsf{negl}(\lambda) \geq \frac{1}{2} + \frac{\mu(\lambda)}{2} \tag{4.3}$$

where the probability is only over $\mathcal{P} \leftarrow \mathsf{RF}_{2\lambda+m}^{\lambda+m}[F]$, $b \leftarrow \{0,1\}$, and $z_b \leftarrow \mathsf{Gen}_{\mathsf{tlp}}(1^\lambda, t, (s_b||r_b))$.

We will use this to construct an adversary that breaks the hardness of TLP. The adversary $\mathcal{B}$ that has s_0, s_1, r_0, r_1, F, and $\mathcal{D}$ hardcoded, and receives as input a puzzle z corresponding to $s_b||r_b$ for $b \in \{0,1\}$. It does the following:

1. Run $\mathcal{D}^{(\cdot)}(z)$. For each query q made by $\mathcal{D}$, if $q \in F$, give the corresponding answer; otherwise, if q has been asked previously, give the same answer as before; otherwise, give a uniformly random answer sampled from $\{0,1\}^{\lambda_{\mathsf{tlp}}}$.
2. If there exists unique bit b such that $\mathcal{D}$ queries (s_b, r_b) but not (s_{1-b}, r_{1-b}), output b. Otherwise, output the bit output by $\mathcal{D}$.

To analyze the success probability of $\mathcal{B}$, we can look at whether hit or bad occur. By definition of $\mathcal{B}$, we have the following, where all probabilities are over $\mathcal{P} \leftarrow \mathsf{RF}^{\lambda+m}_{2\lambda+m}[F]$, $b \leftarrow \{0,1\}$, and $z_b \leftarrow \mathsf{Gen}_{\mathsf{tlp}}(1^\lambda, t, (s_b||r_b))$:

- $\Pr\left[\mathcal{B}(z_b) = b \ \wedge \ \mathsf{hit} \ \wedge \ \overline{\mathsf{bad}}\right] = \Pr\left[\mathsf{hit} \ \wedge \ \overline{\mathsf{bad}}\right]$, since when hit and $\overline{\mathsf{bad}}$ occur, then $B(z_b)$ always outputs b.
- $\Pr\left[\mathcal{B}(z_b) = b \ \wedge \ \overline{\mathsf{hit}} \ \wedge \ \overline{\mathsf{bad}}\right] = \Pr\left[\mathcal{D}^{\mathcal{P}}(z_b) = b \ \wedge \ \overline{\mathsf{hit}} \ \wedge \ \overline{\mathsf{bad}}\right]$, since when neither hit nor bad occur, then the query answers that $\mathcal{B}$ gives to $\mathcal{D}$ are distributed according to $\mathcal{P}$, and $\mathcal{B}$ simply outputs what $\mathcal{D}$ outputs.
- $\Pr\left[\mathcal{B}(z_b) = b \ \wedge \ \mathsf{bad}\right] \geq 0$.

Combining these with Equation 4.3, we get

$$\Pr\left[\mathcal{B}(z_b) = b\right] \geq \Pr\left[\mathsf{hit} \ \wedge \ \overline{\mathsf{bad}}\right] + \Pr\left[\mathcal{D}^{\mathcal{P}}(z_b) = b \ \wedge \ \overline{\mathsf{hit}} \ \wedge \ \overline{\mathsf{bad}}\right] \geq \frac{1}{2} + \frac{\mu(\lambda)}{2} - \mathsf{negl}(\lambda)$$

for infinitely many $\lambda \in \mathbb{N}$, where the probabilities are over $\mathcal{P} \leftarrow \mathsf{RF}^{\lambda+m}_{2\lambda+m}[F]$, $b \leftarrow \{0,1\}$, and $z_b \leftarrow \mathsf{Gen}_{\mathsf{tlp}}(1^\lambda, t, (s_b||r_b))$.

To put everything together and letting $s = s_0||r_0$ and $s' = s_1||r_1$, the above implies that

$$\left|\Pr\left[z \leftarrow \mathsf{Gen}_{\mathsf{tlp}}(1^\lambda, t, s) : \mathcal{B}(z) = 1\right] \quad - \Pr\left[z = \mathsf{Gen}_{\mathsf{tlp}}(1^\lambda, t, s') : \mathcal{B}(z) = 1\right]\right| > \mu(\lambda) - 2\mathsf{negl}(\lambda).$$

We will show that this contradicts hardness of TLP for T. Recall that α_{tlp} is the polynomial guaranteed to exist by the hardness of TLP, and we set $\alpha(\lambda) = \alpha_{\mathsf{tlp}}(\lambda) \cdot p(\lambda)$ for a polynomial p specified below. To complete the proof, we need to show that $\alpha_{\mathsf{tlp}}(\lambda) \leq T(\lambda) \in \mathrm{poly}(\lambda, B(\lambda))$ and that the size and depth of $\mathcal{B}$ are such that $\mathcal{B}$ breaks the hardness of TLP. The bounds on T are immediate as $\alpha(\lambda) \leq T(\lambda) \in \mathrm{poly}(\lambda, B(\lambda))$ and $\alpha(\lambda) = \alpha_{\mathsf{tlp}}(\lambda) \cdot p(\lambda)$, since we will set $p(\lambda) \geq 1$.

For the size and depth of $\mathcal{B}$, we have that $\mathcal{B}$ needs to run $\mathcal{D}$, and needs to be able to answer $\mathcal{D}$'s oracle queries consistently with F and with each other. One way to implement this is for $\mathcal{B}$ to keep track of a set $\mathcal{Q}$ of the queries asked so far, initialized to F. At any point during the emulation of $\mathcal{B}$, the set $\mathcal{Q}$ contains at most $\mathsf{size}(\mathcal{D}) + f_{\mathsf{hard}}(\lambda)$ queries. For each query made by $\mathcal{B}$, checking if it appears in $\mathcal{Q}$ and adding it to $\mathcal{Q}$ if necessary can be done in size $\mathrm{poly}(|\mathcal{Q}|)$, and so in total results in adding $\mathsf{size}(\mathcal{D}) \cdot \mathrm{poly}(|\mathcal{Q}|)$ to the size of $\mathcal{B}$ to account for all the queries. For the depth, each query can be checked in $\mathrm{poly}(\lambda, \log|\mathcal{Q}|)$ depth and added to $\mathcal{Q}$ (while keeping $\mathcal{Q}$ sorted) with an additional $\mathrm{poly}(\lambda, \log|\mathcal{Q}|)$ depth. Finally, for queries made in parallel, we can answer them in parallel.[8] Overall, this results in an extra additive depth of $\mathsf{depth}(\mathcal{D}) \cdot \mathrm{poly}(\lambda, \log f_{\mathsf{hard}}(\lambda), \log B(\lambda))$.

To put everything together, recall that $f_{\mathsf{hard}}(\lambda) = (q_{\mathcal{A}}(\lambda, 2^m) \cdot 4q(\lambda))^2$, $B(\lambda) \in 2^m \cdot \mathrm{poly}(\lambda)$, and $\mathsf{size}(\mathcal{D}) \in \mathrm{poly}(B(\lambda))$. Therefore, we get that

$$\mathsf{size}(\mathcal{B}) \in \mathrm{poly}(\mathsf{size}(\mathcal{D}), f_{\mathsf{hard}}(\lambda)) \in \mathrm{poly}(B(\lambda), (q_{\mathcal{A}}(\lambda, 2^m) \cdot 4q(\lambda))^2) \in \mathrm{poly}(\lambda, 2^m) \in \mathrm{poly}(\lambda, B(\lambda))$$

[8] In more detail, when given n parallel queries, we can check membership in $\mathcal{Q}$ in parallel, and then sort the new queries in $O(\log n)$ depth, remove duplicates from the sorted list in parallel, and then add them to $\mathcal{Q}$ along with corresponding answers.

and

$$\begin{aligned}\mathsf{depth}(\mathcal{B}) &\leq \mathsf{depth}(\mathcal{D}) + \mathsf{depth}(\mathcal{D}) \cdot \mathrm{poly}(\lambda, \log f_{\mathsf{hard}}(\lambda), \log B(\lambda)) \\ &\in \mathsf{depth}(\mathcal{D}) \cdot \mathrm{poly}(\lambda, m) \in \mathsf{depth}(\mathcal{D}) \cdot p(\lambda)\end{aligned}$$

for a fixed polynomial p, depending only on m and on $\log f_{\mathsf{hard}}(\lambda)$. Note that $\log f_{\mathsf{hard}}(\lambda)$ is in $\mathrm{poly}(\lambda, m, \log(q(\lambda)))$. Since q was assumed to be a polynomial, this can be upper bounded by a fixed polynomial in λ, which is independent of q, for sufficiently large λ. It follows that we can assume p is independent of q.

Recall that $\mathsf{depth}(\mathcal{D}) \leq T(\lambda)/\alpha(\lambda)$ where $\alpha(\lambda) = \alpha_{\mathsf{tlp}} \cdot p(\lambda)$. We can therefore upper bound $\mathsf{depth}(\mathcal{D})$ in the above to get that

$$\mathsf{depth}(\mathcal{B}) \leq T(\lambda)/(\alpha(\lambda)) \cdot p(\lambda) = T(\lambda)/(\alpha_{\mathsf{tlp}}(\lambda) \cdot p(\lambda)) \cdot p(\lambda) = T(\lambda)/(\alpha_{\mathsf{tlp}}(\lambda)).$$

Therefore, $\mathcal{B}$ breaks the hardness of TLP with probability $\mu(\lambda) - 2\mathsf{negl}(\lambda)$ for infinitely many λ. ■

This completes the proof of Lemma 4.6. □

Lemma 4.11. *Assuming that* $(\mathsf{Gen}, \mathsf{Sol})$ *is a* B*-secure and correct TLP, and that* $(\mathsf{Gen}_{\mathsf{tlp}}, \mathsf{Sol}_{\mathsf{tlp}})$ *is a* B*-secure TLP, then* $(\mathsf{Gen}, \mathsf{Sol})$ *is fully concurrent functional non-malleable for* $\mathcal{F}_{B,m}$.

Proof. We will show that $(\mathsf{Gen}, \mathsf{Sol})$ satisfies one-many functional non-malleability for $\mathcal{F}_{B,m}$, which suffices for fully concurrent functional non-malleability for $\mathcal{F}_{B,m}$ by Lemma C.1.

To show one-many functional non-malleability, suppose for contradiction that there exists an $f \in \mathcal{F}_{B,m}$ such that for all positive polynomials α, there exists a function T satisfying $\alpha(\lambda) \leq T(\lambda) \in \mathrm{poly}(\lambda, B(\lambda))$ for all $\lambda \in \mathbb{N}$, an unbounded algorithm Z and polynomial n such that Z outputs $(1, n, B, \alpha, T)$-MIM adversaries, an unbounded distinguisher $\mathcal{D}$, and a polynomial q, such that for infinitely many $\lambda \in \mathbb{N}$ and $s \in \{0,1\}^\lambda$, it holds that

$$\left| \Pr\left[\begin{array}{l} \mathcal{O} \leftarrow \mathsf{RF}^{\lambda+m}_{2\lambda+m} \\ \mathcal{A} = Z(\mathcal{O}) \\ \vec{\vec{s}} \leftarrow \mathsf{mim}^{\mathcal{O}}_{\mathcal{A}}(1^\lambda, T(\lambda), s) \end{array} : \mathcal{D}(f(\vec{\vec{s}})) = 1 \right] \right. \tag{4.4}$$

$$\left. - \Pr\left[\begin{array}{l} \mathcal{O} \leftarrow \mathsf{RF}^{\lambda+m}_{2\lambda+m} \\ \mathcal{A} = Z(\mathcal{O}) \\ \vec{\vec{s}} \leftarrow \mathsf{mim}^{\mathcal{O}}_{\mathcal{A}}(1^\lambda, T(\lambda), 0^\lambda) \end{array} : \mathcal{D}(f(\vec{\vec{s}})) = 1 \right] \right| > \frac{1}{q(\lambda)}. \tag{4.5}$$

Let α_{hard} and α_{tlp} be the positive polynomials from the hardness properties of nmTLP_m (given by 4.6) and TLP, respectively. We will derive a contradiction to the above for α given by $\alpha(\lambda) = \alpha_{\mathsf{hard}} \cdot p_1(\lambda) + \alpha_{\mathsf{tlp}} \cdot p_2(\lambda)$, where p_1 is a fixed polynomial specified in the proof of Claim 4.16, and p_2 is a fixed polynomial specified in the proof of Subclaim 4.15.

In order to show a contradiction, we will give a sequence of hybrid experiments starting with the event in the first probability above and ending with the second, and we will show that each consecutive pair either has large statistical distance, or can be used to break hardness of nmTLP_m or of TLP. Before giving the formal description of the hybrids, we give a short overview. At a high level, our strategy will be to change the way that $\vec{\vec{s}}$ is computed, so as to make it independent of s. Specifically, we start with a hybrid corresponding to the first probability above, where $\vec{\vec{s}}$ is computed using the *unbounded* sampler $\mathsf{mim}_{\mathcal{A}}$. Next, we switch $\mathsf{mim}_{\mathcal{A}}$ to a sampler that has *polynomial size*. This allows us to transition from the random oracle $\mathcal{O}$ to an oracle $\mathcal{P}$ that is lazily sampled on most points using Lemma 3.6, so that the MIM adversary $\mathcal{A}$ only depends on a small fraction of points in the oracle (rather than the whole oracle). This enables us to then switch to sampling $\vec{\widetilde{s}}$ using a *depth-bounded, but sub-expenential size* sampler. Once we have done so, we can apply the hardness of our time-lock puzzle to switch the initial puzzle for s to a puzzle for 0^λ, thereby making the experiment independent of s. The hybrids are formalized next.

For any $\lambda \in \mathbb{N}$ and value $s \in \{0,1\}^\lambda$, we consider the following sequence of hybrid experiments. Throughout these hybrids, we let $t = T(\lambda)$.

- $\mathsf{Hyb}_0^s(\lambda)$: This hybrid is equivalent to the terms in the probabilities above for $s \in \{0,1\}^\lambda$.

$$\mathsf{Hyb}_0^s(\lambda) = \left\{ \begin{array}{l} \mathcal{O} \leftarrow \mathsf{RF}_{2\lambda+m}^{\lambda+m}; \quad \mathcal{A} = Z(\mathcal{O}) \\ \vec{\widetilde{s}} \leftarrow \mathsf{mim}_{\mathcal{A}}^{\mathcal{O}}(1^\lambda, t, s) \end{array} : \ \mathcal{D}(f(\vec{\widetilde{s}})) = 1 \right\}$$

- $\mathsf{Hyb}_1^s(\lambda)$: In this hybrid, we replace $\mathsf{mim}_{\mathcal{A}}$ with a polynomial-time algorithm $\mathcal{B}$, which is defined based on $\mathcal{A}$, and sampled by an algorithm Z'. Specifically, let Z' be the algorithm such that $Z'(\mathcal{A})$ outputs the following probabilistic oracle algorithm $\mathcal{B}^{\mathcal{O}}$:

  1. Sample $z \leftarrow \mathsf{Gen}^{\mathcal{O}}(1^\lambda, t, s)$.
  2. Run $\vec{\widetilde{z}} \leftarrow \mathcal{A}^{\mathcal{O}}(z)$.
  3. Do the following to compute the output vector $\vec{\widetilde{s}}$. For each $i \in [n(\lambda)]$, if $\vec{\widetilde{z}}_i = z$, set $\widetilde{s}_i = \bot$. Otherwise, compute $s' = \mathsf{Sol}^{\mathcal{O}}(1^\lambda, t, \widetilde{z}_i)$, and set $\widetilde{s}_i = s'$.
  4. Output $\vec{\widetilde{s}}$.

  Note that $\mathcal{B}^{\mathcal{O}}$ forwards all queries made by internal algorithms to its own oracle. We can now define the hybrid experiment:

$$\mathsf{Hyb}_1^s(\lambda) = \left\{ \begin{array}{l} \mathcal{O} \leftarrow \mathsf{RF}_{2\lambda+m}^{\lambda+m}; \quad \mathcal{A} = Z(\mathcal{O}); \quad \mathcal{B} = Z'(\mathcal{A}) \\ \vec{\widetilde{s}} \leftarrow \mathcal{B}^{\mathcal{O}} \end{array} : \mathcal{D}(f(\vec{\widetilde{s}})) = 1 \right\}$$

  Note that $\mathsf{size}(\mathcal{B}) \in \text{poly}(\lambda, 2^m)$, which follows by the efficiency of Gen and Sol, and because $n(\lambda) \in \text{poly}(\lambda)$ and $\mathsf{size}(\mathcal{A}) \in \text{poly}(\lambda, B)$.

- $\mathsf{Hyb}_2^s(\lambda)$: Let $q_{\mathcal{B}}$ be the polynomial such that $\mathsf{size}(\mathcal{B}) = q_{\mathcal{B}}(\lambda, 2^m)$. Let $f_{\mathsf{nm}}(\lambda) = (q_{\mathcal{B}}(\lambda, 2^m) \cdot 2q(\lambda))^2$ and let Sam be the inefficient algorithm given in Lemma 3.6 that on input an adversary, outputs a partial assignment F on $f_{\mathsf{nm}}(\lambda)$ points. This hybrid switches the random oracle $\mathcal{O}$ with a random function $\mathcal{P}$ fixed at the set of points F.

$$\mathsf{Hyb}_1^s(\lambda) = \left\{ \begin{array}{l} \mathcal{O} \leftarrow \mathsf{RF}_{2\lambda+m}^{\lambda+m}; \quad \mathcal{A} = Z(\mathcal{O}); \quad \mathcal{B} = Z'(\mathcal{A}) \\ F \leftarrow \mathsf{Sam}(\mathcal{B}); \quad \mathcal{P} \leftarrow \mathsf{RF}_{2\lambda+m}^{\lambda+m}[F] \\ \vec{\widetilde{s}} \leftarrow \mathcal{B}^{\mathcal{P}} \end{array} : \ \mathcal{D}(f(\vec{\widetilde{s}})) = 1 \right\}$$

- $\mathsf{Hyb}_3^s(\lambda)$: In this hybrid, we change the experiment so that $\vec{\widetilde{s}}$ can be computed in depth less than t (given $\mathcal{A}$ and F). Specifically, we define the distribution bmim (standing for "bounded MIM") and replace $\mathcal{B}$ with this distribution, defined as follows. Define bmim as follows:

  $\underline{\mathsf{bmim}_{\mathcal{A},F}^{\mathcal{P}}(1^\lambda, t, s)}$:

  1. Sample $z \leftarrow \mathsf{Gen}^{\mathcal{P}}(1^\lambda, t, s)$
  2. Run $\vec{\widetilde{z}} \leftarrow \mathcal{A}^{(\cdot)}(z)$ by forwarding all of $\mathcal{A}$'s queries to the oracle $\mathcal{P}$.

3. Let $\mathcal{Q}$ be the set containing all oracle queries made by $\mathcal{A}$ and all points in the partial assignment F, where $Q_j = (s_j, r_j)$ for each $j \in [|\mathcal{Q}|]$.
4. Next, we form the output vector $\vec{\widetilde{s}}$. For each $i \in [n(\lambda)]$, if $\widetilde{z}_i = z$, set $\widetilde{s}_i = \bot$. Otherwise, check if there exists a query $Q_j \in \mathcal{Q}$ with $\widetilde{z}_i = \mathsf{Gen}^{\mathcal{P}}(1^\lambda, t, s_j; r_j)$, and if so set $\widetilde{s}_i = s_j$. Otherwise, set $\widetilde{s}_i = \bot$. Note that this check can be done in parallel for each pair (i, j).
5. Output $\vec{\widetilde{s}}$.

We define this hybrid experiment from the previous one by using bmim instead of $\mathcal{B}$ to sample $\vec{\widetilde{s}}$, as follows.

$$\mathsf{Hyb}_3^s(\lambda) = \left\{ \begin{array}{ll} \mathcal{O} \leftarrow \mathsf{RF}_{2\lambda+m}^{\lambda+m}; \quad \mathcal{A} = Z(\mathcal{O}); \quad \mathcal{B} = Z'(\mathcal{A}) & \\ F \leftarrow \mathsf{Sam}(\mathcal{B}); \quad \mathcal{P} \leftarrow \mathsf{RF}_{2\lambda+m}^{\lambda+m}[F] & : \mathcal{D}(f(\vec{\widetilde{s}})) = 1 \\ \vec{\widetilde{s}} \leftarrow \mathsf{bmim}_{\mathcal{A},F}^{\mathcal{P}}(1^\lambda, t, s) & \end{array} \right\}$$

To complete the proof, we show that for each consecutive pair of hybrids, either the statistical distance is bounded or we can use it to break security of TLP or nmTLP_m. Next, we discuss how to use these claims to complete the proof, and then we give the claims.

We show in Claim 4.12 that $\Pr[\mathsf{Hyb}_0^s(\lambda)] = \Pr[\mathsf{Hyb}_1^s(\lambda)]$ and in Claim 4.13 that $|\Pr[\mathsf{Hyb}_1^s(\lambda)] - \Pr[\mathsf{Hyb}_2^s(\lambda)]| \leq 1/(2q(\lambda))$, for all $\lambda \in \mathbb{N}$ and $s \in \{0,1\}^\lambda$. Combining these with Equation 4.4, we get that for infinitely many $\lambda \in \mathbb{N}$, there exists a value $s \in \{0,1\}^\lambda$ such that

$$\left| \Pr[\mathsf{Hyb}_2^s(\lambda)] - \Pr\left[\mathsf{Hyb}_2^{0^\lambda}(\lambda)\right] \right| \geq \frac{1}{2q(\lambda)}.$$

It follows that for infinitely many $\lambda \in \mathbb{N}$, there exists a value $s \in \{0,1\}^\lambda$ and a pair of consecutive hybrids in the sequence $\mathsf{Hyb}_2^s(\lambda), \mathsf{Hyb}_3^s(\lambda), \mathsf{Hyb}_3^{0^\lambda}(\lambda), \mathsf{Hyb}_2^{0^\lambda}(\lambda)$ whose statistical distance is at least $1/(6q(\lambda))$.

In the first case where the statistical distance between $\mathsf{Hyb}_2^s(\lambda)$ and $\mathsf{Hyb}_3^s(\lambda)$ is at least $1/(6q(\lambda))$, we show in Claim 4.14 that this implies an adversary that breaks the hardness of TLP with probability $1/(6q(\lambda) \cdot n(\lambda))$ which is a contradiction, since n is a polynomial. In the second case where the statistical distance between $\mathsf{Hyb}_3^s(\lambda)$ and $\mathsf{Hyb}_3^{0^\lambda}(\lambda)$ is at least $1/(6q(\lambda))$, we show in Claim 4.16 that this implies a negligible function negl and an adversary that breaks the hardness of nmTLP_m with probability $1/(6q(\lambda)) - \mathsf{negl}(\lambda) \geq 1/(12q(\lambda))$, which contradicts the hardness of nmTLP_m. The third case follows identically to the first one, which concludes the proof.

Claim 4.12. *For all $s \in \{0,1\}^\lambda$, $\Pr[\mathsf{Hyb}_0^s(\lambda)] = \Pr[\mathsf{Hyb}_1^s(\lambda)]$.*

Proof. In both of these hybrids, the oracle $\mathcal{O}$ and adversary $\mathcal{A}$ are sampled equivalently. The difference between the two distributions is the following:

- In $\mathsf{Hyb}_0^s(\lambda)$, the vector $\vec{\widetilde{s}}$ is sampled by $\mathsf{mim}_{\mathcal{A}}^{\mathcal{O}}(1^\lambda, t, s)$.
- In $\mathsf{Hyb}_1^s(\lambda)$, the vector $\vec{\widetilde{s}}$ is determined by letting $\mathcal{B} = Z(\mathcal{A})$ and sampling $\vec{\widetilde{s}} \leftarrow \mathcal{B}^{\mathcal{O}}$.

We will show that these have the same distribution.

To sample $\vec{\widetilde{s}}$, both $\mathcal{B}$ and $\mathsf{mim}_{\mathcal{A}}$ compute a puzzle z for s and run $\vec{\widetilde{z}} \leftarrow \mathcal{A}(z)$, but then form the output vector $\vec{\widetilde{s}}$ differently. For ease of understanding, denote this vector as computed by $\mathsf{mim}_{\mathcal{A}}$ in

$\mathsf{Hyb}_0^s(\lambda)$ as $\widetilde{s}^{(0)}$, and denote the vector as computed by $\mathcal{B}$ in $\mathsf{Hyb}_1^s(\lambda)$ as $\widetilde{s}^{(1)}$. Specifically, for each output element $\widetilde{s}_i$, $\mathsf{mim}_{\mathcal{A}}$ computes it as

$$\widetilde{s}_i^{(0)} = \begin{cases} \bot & \text{if } \widetilde{z}_i = z \\ s' & \text{if there is a unique } s' \text{ with } \widetilde{z}_i = \mathsf{Gen}^{\mathcal{O}}(1^\lambda, t, s'; r) \text{ for some } r \\ \bot & \text{otherwise,} \end{cases}$$

while $B^{\mathcal{O}}$ computes it as

$$\widetilde{s}_i^{(1)} = \begin{cases} \bot & \text{if } \widetilde{z}_i = z \\ s' & \text{otherwise, where } s' = \mathsf{Sol}^{\mathcal{O}}(1^\lambda, t, \widetilde{z}_i). \end{cases}$$

We will show that $\widetilde{s}_i^{(0)} = \widetilde{s}_i^{(1)}$. Note that if $\widetilde{z}_i = z$, then $\widetilde{s}_i = \bot$ in both cases, so we only need to focus on the case when $\widetilde{z}_i \neq z$. To complete the proof, we will show that there exists a unique $s' \in \{0,1\}^*$ with $\widetilde{z}_i = \mathsf{Gen}^{\mathcal{O}}(1^\lambda, t, s'; r)$ for some r if and only if $\mathsf{Sol}^{\mathcal{O}}(1^\lambda, t, \widetilde{z}_i) \neq \bot$. This suffices for the claim since whenever this occurs, correctness of nmTLP_m implies that both $\mathsf{mim}_{\mathcal{A}}$ and $\mathcal{B}^{\mathcal{O}}$ will set $\widetilde{s}_i = s'$. Otherwise, both will set it to $\bot$, implying that $\widetilde{s}_i$ is identically distributed in both hybrids for all i.

For the "if" direction, suppose that there exists a unique value $s' \in \{0,1\}^*$ with $\widetilde{z}_i = \mathsf{Gen}^{\mathcal{O}}(1^\lambda, t, s'; r)$ for some r. Correctness of nmTLP_m directly implies that $\mathsf{Sol}^{\mathcal{O}}(1^\lambda, t, \widetilde{z}_i)$ will output s', and hence the output of Sol is not equal to $\bot$.

For the "only if" direction, suppose $\mathsf{Sol}^{\mathcal{O}}(1^\lambda, t, \widetilde{z}_i) \neq \bot$. Recall that Sol solves the underlying time-lock puzzle to obtain a value $s''||r$, and then outputs s'' if $\widetilde{z}_i = \mathsf{Gen}^{\mathcal{O}}(1^\lambda, t, s''; r)$ and $\bot$ otherwise. As we are in the case where Sol does not output $\bot$, it holds that $\widetilde{z}_i = \mathsf{Gen}^{\mathcal{O}}(1^\lambda, t, s''; r)$. Correctness of nmTLP_m implies that s'' is the unique solution to $\widetilde{z}_i$. This completes the proof of the claim. ∎

Claim 4.13. *There exists a negligible function* negl_1 *such that for all* $s \in \{0,1\}^\lambda$,

$$|\Pr[\mathsf{Hyb}_1^s(\lambda)] - \Pr[\mathsf{Hyb}_2^s(\lambda)]| \leq 1/(2q(\lambda)).$$

Proof. We start by noting that $\mathsf{Hyb}_1^s(\lambda)$ can be rewritten as

$$\mathsf{Hyb}_1^s(\lambda) = \left\{ \begin{array}{ll} \mathcal{O} \leftarrow \mathsf{RF}_{2\lambda+m}^{\lambda+m}; & \mathcal{B} = Z''(\mathcal{O}) \\ \vec{\widetilde{s}} \leftarrow \mathcal{B}^{\mathcal{O}} & \end{array} : \mathcal{D}(f(\vec{\widetilde{s}})) = 1 \right\}$$

where $Z''(\mathcal{O})$ samples $\mathcal{A} = Z(\mathcal{O})$ and $\mathcal{B} = Z'(\mathcal{A})$. Using Z'', we can also rewrite $\mathsf{Hyb}_2^s(\lambda)$ as

$$\left\{ \begin{array}{ll} \mathcal{O} \leftarrow \mathsf{RF}_{2\lambda+m}^{\lambda+m}; & \mathcal{B} = Z''(\mathcal{O}) \\ F \leftarrow \mathsf{Sam}(\mathcal{B}); & \mathcal{P} \leftarrow \mathsf{RF}_{2\lambda+m}^{\lambda+m}[F] \\ \vec{\widetilde{s}} \leftarrow \mathcal{B}^{\mathcal{P}} & \end{array} : \mathcal{D}(f(\vec{\widetilde{s}})) = 1 \right\}.$$

Since $|F| = f_{\mathsf{nm}}(\lambda) = (q_{\mathcal{B}}(\lambda, 2^m) \cdot 2q(\lambda))^2$, then by Lemma 3.6 the statistical distance between these hybrids is at most

$$\sqrt{\frac{(q_{\mathcal{B}}(\lambda, 2^m))^2}{f_{\mathsf{nm}}(\lambda)}} = \sqrt{\frac{(q_{\mathcal{B}}(\lambda, 2^m))^2}{(q_{\mathcal{B}}(\lambda, 2^m) \cdot 2q(\lambda))^2}} = \frac{1}{2q(\lambda)}$$

where we recall that $\mathsf{size}(\mathcal{B}) = q_{\mathcal{B}}(\lambda, 2^m)$. ∎

Claim 4.14. *If there exists a function μ such that for infinitely many $\lambda \in \mathbb{N}$ there exists a value $s \in \{0,1\}^\lambda$ with*

$$|\Pr[\mathsf{Hyb}_2^s(\lambda)] - \Pr[\mathsf{Hyb}_3^s(\lambda)]| \geq \mu(\lambda),$$

then there exists an adversary that breaks the hardness of TLP *with probability $\mu(\lambda)/n(\lambda)$ for infinitely many $\lambda \in \mathbb{N}$.*

Proof. The difference between these two hybrids is that in $\mathsf{Hyb}_2^s(\lambda)$, the values $\vec{\widetilde{s}}$ are sampled from $\mathcal{B}^{\mathcal{P}}$, while in $\mathsf{Hyb}_3^s(\lambda)$ they are sampled by $\mathsf{bmim}_{\mathcal{A},F}^{\mathcal{P}}(1^\lambda, t, s)$. We will show that if these two distributions are far, then we can construct an adversary that breaks the hardness of TLP with probability roughly the statistical distance between the two hybrids.

The way that $\vec{\widetilde{s}}$ is sampled in each hybrid is as follows. Both $\mathcal{B}^{\mathcal{P}}$ and $\mathsf{bmim}_{\mathcal{A},F}^{\mathcal{P}}(1^\lambda, t, s)$ first sample a puzzle z for s, and then run $\vec{\widetilde{z}} \leftarrow \mathcal{A}(z)$. They then calculate $\vec{\widetilde{s}}$ as follows. For ease of understanding, denote this vector as computed by $\mathcal{B}$ in $\mathsf{Hyb}_2^s(\lambda)$ as $\widetilde{s}^{(2)}$, and denote the vector as computed by bmim in $\mathsf{Hyb}_3^s(\lambda)$ as $\widetilde{s}^{(3)}$. Then, we have that for each output element i, $\mathcal{B}^{\mathcal{P}}$ computes it as

$$\widetilde{s}_i^{(2)} = \begin{cases} \perp & \text{if } \widetilde{z}_i = z \\ s' & \text{otherwise, where } s' = \mathsf{Sol}^{\mathcal{P}}(1^\lambda, t, \widetilde{z}_i) \end{cases}$$

while $\mathsf{bmim}_{\mathcal{A},F}^{\mathcal{P}}(1^\lambda, t, s)$ computes it as

$$\widetilde{s}_i^{(3)} = \begin{cases} \perp & \text{if } \widetilde{z}_i = z \\ v_j & \text{if there exists a } Q_j = (s_j, r_j) \in \mathcal{Q} \text{ with } \widetilde{z}_i = \mathsf{Gen}^{\mathcal{P}}(1^\lambda, t, s_j; r_j) \\ \perp & \text{otherwise,} \end{cases}$$

where we recall that $\mathcal{Q}$ consists of all queries made by $\mathcal{A}$ and all elements of the partial assignment F. Unless otherwise stated, note that all following probabilities are over $\mathcal{O} \leftarrow \mathsf{RF}_{2\lambda+m}^{\lambda+m}$, $\mathcal{A} = Z(\mathcal{O})$, $\mathcal{B} = Z'(\mathcal{A})$, $F = \mathsf{Sam}(\mathcal{B})$, $\mathcal{P} \leftarrow \mathsf{RF}_{2\lambda+m}^{\lambda+m}[F]$, $\vec{\widetilde{s}}^{(2)} \leftarrow \mathcal{B}^{\mathcal{P}}$, $\vec{\widetilde{s}}^{(3)} \leftarrow \mathsf{bmim}_{\mathcal{A},F}^{\mathcal{P}}(1^\lambda, t, s)$. By assumption, we have that

$$\begin{aligned} \mu(\lambda) &\leq |\Pr[\mathsf{Hyb}_2^s(\lambda)] - \Pr[\mathsf{Hyb}_3^s(\lambda)]| \leq \Pr\left[\widetilde{s}^{(2)} \neq \widetilde{s}^{(3)}\right] \\ &= \Pr\left[\exists i \in [n(\lambda)] : \widetilde{s}_i^{(2)} \neq \widetilde{s}_i^{(3)}\right] \leq \sum_{i \in [n(\lambda)]} \Pr\left[\widetilde{s}_i^{(2)} \neq \widetilde{s}_i^{(3)}\right] \end{aligned}$$

by a union bound. Therefore, there exists some $i \in [n(\lambda)]$ such that

$$\Pr\left[\widetilde{s}_i^{(2)} \neq \widetilde{s}_i^{(3)}\right] \geq \mu(\lambda)/n(\lambda). \tag{4.6}$$

Fix this index i. We will continue by expanding this probability.

Let E be the event that there exists a $Q_j = (s_j, r_j) \in \mathcal{Q}$ such that $\widetilde{z}_i = \mathsf{Gen}^{\mathcal{P}}(1^\lambda, t, s_j; r_j)$. We have that

$$\begin{aligned} \Pr\left[\widetilde{s}_i^{(2)} \neq \widetilde{s}_i^{(3)}\right] &= \Pr\left[\widetilde{s}_i^{(2)} \neq \widetilde{s}_i^{(3)} \ \wedge \ \mathsf{E}\right] + \Pr\left[\widetilde{s}_i^{(2)} \neq \widetilde{s}_i^{(3)} \ \wedge \ \neg\mathsf{E}\right] \\ &\leq \Pr\left[\widetilde{s}_i^{(2)} \neq \widetilde{s}_i^{(3)} \ \mid \ \mathsf{E}\right] + \Pr\left[\widetilde{s}_i^{(2)} \neq \widetilde{s}_i^{(3)} \ \mid \ \neg\mathsf{E}\right], \end{aligned} \tag{4.7}$$

We continue by bounding each term in Equation 4.7 separately. For the first term, we have that

$$\Pr\left[\widetilde{s}_i^{(2)} \neq \widetilde{s}_i^{(3)} \ \mid \ \mathsf{E}\right] = 0.$$

To see this, note that conditioning on E implies $\widetilde{s}_i^{(3)} = s_j$, where $Q_j = (s_j, r_j) \in \mathcal{Q}$ is the query such that $\widetilde{z}_i = \mathsf{Gen}^{\mathcal{P}}(1^\lambda, t, s_j; r_j)$ (note that correctness implies that if there is more than one such query, it must correspond to the same value of s_j). By the correctness of nmTLP_m, this implies that $\mathsf{Sol}^{\mathcal{P}}(1^\lambda, t, \widetilde{z}_i)$ outputs s_j, so $\widetilde{s}_i^{(2)} = s_j$.

To bound the second term of Equation 4.7, conditioning on $\neg\mathsf{E}$ implies $\widetilde{s}_i^{(3)} = \bot$, and so

$$\Pr\left[\widetilde{s}_i^{(2)} \neq \widetilde{s}_i^{(3)} \mid \neg\mathsf{E}\right] = \Pr\left[\widetilde{s}_i^{(2)} \neq \bot \mid \neg\mathsf{E}\right].$$

Recall that $\widetilde{s}_i^{(2)}$ is sampled as $\mathsf{Sol}^{\mathcal{P}}(1^\lambda, t, \widetilde{z}_i)$. By definition of Sol, it holds that its output is not $\bot$ when $\widetilde{z}_i = \mathsf{Gen}^{\mathcal{P}}(1^\lambda, s', t; r)$ for $s'||r = \mathsf{Sol}_{\mathsf{tlp}}(1^\lambda, t, \widetilde{z}_i)$. Therefore, the above is equal to

$$\begin{aligned}\Pr\left[\mathsf{Sol}^{\mathcal{P}}(1^\lambda, t, \widetilde{z}_i) \neq \bot \mid \neg\mathsf{E}\right] &= \Pr\left[\widetilde{z}_i = \mathsf{Gen}^{\mathcal{P}}(1^\lambda, s', t; r) \mid \neg\mathsf{E}\right] \\ &= \Pr\left[\widetilde{z}_i = \mathsf{Gen}_{\mathsf{tlp}}\left(1^\lambda, t, (s'||r);\ \mathcal{P}(s', r)\right) \mid \neg\mathsf{E}\right],\end{aligned}$$

where the above probabilities are over the choice of $\mathcal{O}, \mathcal{A}, \mathcal{B}, F, \mathcal{P}$ and the randomness used by $\mathcal{A}$ to produce $\widetilde{z}_i$. Since we are conditioning on $\neg\mathsf{E}$, it follows that $\mathcal{P}(s', r)$ is uniformly random and independent from $\mathcal{A}$ and hence from $\widetilde{z}_i$, so it suffices to bound the following probability, relative to any $\widetilde{z}_i$. Therefore, by combining this with equations 4.7 and 4.6, we get the following, where the probability is only over $r' \leftarrow \{0,1\}^{\lambda_{\mathsf{tlp}}}$:

$$\Pr\left[r' \leftarrow \{0,1\}^{\lambda_{\mathsf{tlp}}} : \mathsf{Gen}_{\mathsf{tlp}}(1^\lambda, t, (s'||r); r') = \widetilde{z}_i\right] \geq \mu(\lambda)/n(\lambda).$$

In the following sub-claim, we show that this implies we can break the hardness of TLP with the same probability, which completes the claim. Note that the following sub-claim is general, so the notation is independent from the above.

Subclaim 4.15. *If for infinitely many $\lambda \in \mathbb{N}$ there exist values $z^\star \in \{0,1\}^*$ and $s \in \{0,1\}^{\lambda_{\mathsf{tlp}}+\lambda}$ with*

$$\Pr\left[r \leftarrow \{0,1\}^{\lambda_{\mathsf{tlp}}} : \mathsf{Gen}_{\mathsf{tlp}}(1^\lambda, T(\lambda), s; r) = z^\star\right] \geq \mu'(\lambda),$$

then there exists an adversary that breaks the hardness of TLP *with probability $\mu'(\lambda)$.*

Proof. We will show that there exists a depth-bounded distinguisher $\mathcal{D}' = \{\mathcal{D}'_\lambda\}_{\lambda \in \mathbb{N}}$ that breaks the hardness of the time-lock puzzle with respect to T and inputs s, s' for any input $s' \neq s$. For any $\lambda \in \mathbb{N}$, we define $\mathcal{D}'_\lambda$ on input z to simply output 1 if $z = z^\star$ and 0 otherwise.

To show that $\mathcal{D}'$ contradicts the hardness of TLP, we start by bounding the distinguishing probabilities. Let Λ be the infinitely large set of $\lambda \in \mathbb{N}$ such that the statement of the claim holds. It follows that for all $\lambda \in \Lambda$,

$$\Pr\left[\mathcal{D}'_\lambda(\mathsf{Gen}_{\mathsf{tlp}}(1^\lambda, T(\lambda), s)) = 1\right] = \Pr\left[\mathsf{Gen}_{\mathsf{tlp}}(1^\lambda, T(\lambda), s) = z^\star\right] \geq \mu'(\lambda).$$

In particular, it holds that $z^\star$ is in the support of $\mathsf{Gen}_{\mathsf{tlp}}(1^\lambda, T(\lambda), s)$ for all $\lambda \in \Lambda$. By correctness of TLP, this implies that $\mathsf{Sol}_{\mathsf{tlp}}(1^\lambda, T(\lambda), z^\star) = s$. Correctness also implies that $z^\star$ is not in the support of $\mathsf{Gen}_{\mathsf{tlp}}(1^\lambda, T(\lambda), s')$ for any $\lambda \in \Lambda$. As a result, for all $\lambda \in \Lambda$,

$$\Pr\left[\mathcal{D}'_\lambda(\mathsf{Gen}_{\mathsf{tlp}}(1^\lambda, T(\lambda), s')) = 1\right] = \Pr\left[\mathsf{Gen}_{\mathsf{tlp}}(1^\lambda, T(\lambda), s') = z^\star\right] = 0$$

and therefore $\mathcal{D}'_\lambda$ distinguishes puzzles corresponding to s from s' with probability $\mu'(\lambda)$.

To complete the proof, we need to show bounds on T and the size and depth of $\mathcal{D}'$. Recall that α_{tlp} is the positive polynomial given by the hardness of TLP. We want to show that $\alpha_{\mathsf{tlp}}(\lambda) \leq T(\lambda) \in \mathrm{poly}(\lambda, B(\lambda))$, that $\mathsf{size}(\mathcal{D}'_\lambda) \in \mathrm{poly}(\lambda, B(\lambda))$, and that $\mathsf{depth}(\mathcal{D}'_\lambda) \leq T(\lambda)/(\alpha_{\mathsf{tlp}}(\lambda))$. The bounds on $T(\lambda)$ hold because $\alpha(\lambda) \leq T(\lambda) \in \mathrm{poly}(\lambda, B(\lambda))$ by assumption, and we set $\alpha(\lambda) \geq \alpha_{\mathsf{tlp}}(\lambda) \cdot p_1(\lambda)$, for a polynomial $p_1(\lambda) \geq 1$ specified below.

It remains to bound the size and depth of $\mathcal{D}'$. For each $\lambda \in \mathbb{N}$, the size of $\mathcal{D}_\lambda$ depends on the hardcoded value $z^\star$ for that security parameter. When $\lambda \in \Lambda$ then $z^\star$ is in the support of Gen, so the efficiency of TLP implies that $|z^\star| \in \mathrm{poly}(\lambda, \log T(\lambda)) \in \mathrm{poly}(\lambda)$ by the above bounds on T. For the cases where $\lambda \notin \Lambda$, then we can assume $\mathcal{D}_\lambda$ simply has $\bot$ encoded instead of a value for $z^\star$, and compares its input, which also has length $\mathrm{poly}(\lambda, \log T(\lambda)) \in \mathrm{poly}(\lambda)$, to the hardcoded value. Putting everything together, for every $\lambda \in \mathbb{N}$ the size and depth of $\mathcal{D}'_\lambda$ can be bounded by a fixed polynomial p_1. Therefore

$$\mathsf{size}(\mathcal{D}'_\lambda) \leq p_1(\lambda) \in \mathrm{poly}(\lambda, B(\lambda)).$$

For the depth, recalling that $\alpha(\lambda) \geq \alpha_{\mathsf{tlp}}(\lambda) \cdot p_1(\lambda)$ and $T(\lambda) \geq \alpha(\lambda)$, we have

$$\mathsf{depth}(\mathcal{D}'_\lambda) \leq p_1(\lambda) \leq \frac{\alpha(\lambda)}{\alpha_{\mathsf{tlp}}(\lambda)} \leq \frac{T(\lambda)}{\alpha_{\mathsf{tlp}}(\lambda)},$$

which completes the proof of the subclaim. ■

This completes the proof of Claim 4.14. ■

Claim 4.16. *If there exists a function μ such that for infinitely many $\lambda \in \mathbb{N}$ there exists a value $s \in \{0,1\}^\lambda$ with*

$$\left|\Pr\left[\mathsf{Hyb}_3^s(\lambda)\right] - \Pr\left[\mathsf{Hyb}_3^{0^\lambda}(\lambda)\right]\right| \geq \mu(\lambda),$$

then there exists an adversary that breaks the hardness of nmTLP_m *with probability $\mu(\lambda) - \mathsf{negl}(\lambda)$ for a negligible function* negl *for infinitely many $\lambda \in \mathbb{N}$.*

Proof. By an averaging argument, the inequality in the statement of the claim implies that for infinitely many $\lambda \in \mathbb{N}$, there exists an $\mathcal{O} \in \mathsf{RF}_{2\lambda+m}^{\lambda+m}$, such that for $\mathcal{A} = Z(\mathcal{O})$, $\mathcal{B} = Z'(\mathcal{A})$ and $F = \mathsf{Sam}(\mathcal{B})$, it holds that

$$\left|\Pr\left[\begin{array}{l}\mathcal{P} \leftarrow \mathsf{RF}_{2\lambda+m}^{\lambda+m}[F] \\ \tilde{\vec{s}} \leftarrow \mathsf{bmim}_{\mathcal{A},F}^{\mathcal{P}}(1^\lambda, t, s)\end{array} : \mathcal{D}(f(\tilde{\vec{s}})) = 1\right]\right.$$
$$\left. - \Pr\left[\begin{array}{l}\mathcal{P} \leftarrow \mathsf{RF}_{2\lambda+m}^{\lambda+m}[F] \\ \tilde{\vec{s}} \leftarrow \mathsf{bmim}_{\mathcal{A},F}^{\mathcal{P}}(1^\lambda, t, 0^\lambda)\end{array} : \mathcal{D}(f(\tilde{\vec{s}})) = 1\right]\right| \geq \mu(\lambda).$$

In order to complete the proof of the claim, our goal is to use bmim and $\mathcal{D}$ to construct an adversary against the hardness of nmTLP_m, for which we need the probability to be over the choice of a random oracle $\mathcal{O}'$ rather than a partially fixed on $\mathcal{P}$. Toward that goal, for any oracle $\mathcal{O}'$, let $\widetilde{\mathsf{bmim}}_{\mathcal{A},F}^{\mathcal{O}'}(1^\lambda, t, s)$ be the same as bmim, except that whenever any internal algorithm (such as $\mathcal{A}$ or Gen) makes an oracle query Q, bmim first checks if $Q \in F$. If so, it returns the corresponding point as given by F, and otherwise it forwards the query to its oracle $\mathcal{O}'$.

We observe that sampling $\mathcal{P} \leftarrow \mathsf{RF}_{2\lambda+m}^{\lambda+m}[F]$ and $\vec{\tilde{s}} \leftarrow \mathsf{bmim}_{\mathcal{A},F}^{\mathcal{P}}(1^\lambda, t, s)$ has the same distribution as sampling a fully-defined oracle $\mathcal{O}' \leftarrow \mathsf{RF}_{2\lambda+m}^{\lambda+m}$ and then $\vec{\tilde{s}} \leftarrow \widetilde{\mathsf{bmim}}_{\mathcal{A},F}^{\mathcal{O}'}(1^\lambda, t, s)$. Therefore, we have that

$$\left|\Pr\left[\begin{array}{l}\mathcal{O}' \leftarrow \mathsf{RF}_{2\lambda+m}^{\lambda+m} \\ \vec{\tilde{s}} \leftarrow \widetilde{\mathsf{bmim}}_{\mathcal{A},F}^{\mathcal{O}'}(1^\lambda, t, s)\end{array} : \mathcal{D}(f(\vec{\tilde{s}})) = 1\right]\right.$$
$$\left. - \Pr\left[\begin{array}{l}\mathcal{O}' \leftarrow \mathsf{RF}_{2\lambda+m}^{\lambda+m} \\ \vec{\tilde{s}} \leftarrow \widetilde{\mathsf{bmim}}_{\mathcal{A},F}^{\mathcal{O}'}(1^\lambda, t, 0^\lambda)\end{array} : \mathcal{D}(f(\vec{\tilde{s}})) = 1\right]\right| \geq \mu(\lambda). \tag{4.8}$$

We will use this to break the hiding of $\widetilde{\mathsf{nmTLP}_m}$ by constructing an algorithm $\mathcal{D}'$ that receives a puzzle z either to s or 0^λ, behaves exactly as $\widetilde{\mathsf{bmim}}$ does to obtain $\vec{\tilde{s}}$, computes $f(\vec{\tilde{s}})$, and finally uses $\mathcal{D}$ to distinguish between the two cases. Specifically, let $\mathcal{D}'^{\mathcal{O}'}$ be the oracle algorithm with F hardcoded that on input a puzzle z, does the following:

1. Run $\vec{\tilde{z}} \leftarrow \mathcal{A}^{(\cdot)}(z)$, where for each query Q made by $\mathcal{A}$, if $Q \in F$ then answer with the corresponding image given by F, and otherwise forward Q to $\mathcal{O}'$. Let $\mathcal{Q}$ be the set containing all queries and answers made by $\mathcal{A}$ as well as all points in F.

2. In parallel, for each $i \in [n(\lambda)]$ and $j \in [|\mathcal{Q}|]$, compute $\tilde{s}_i$ as done by $\widetilde{\mathsf{bmim}}_{\mathcal{A},F}^{(\cdot)}(1^\lambda, t, \cdot)$ by checking if $\tilde{z}_i$ is the result of generating a puzzle using $Q_j \in \mathcal{Q}$.

3. Compute $y = f(\vec{\tilde{s}})$.

4. Let $\mathsf{tt}_\mathcal{D}$ be the circuit with width 2^m and depth $O(m)$ corresponding to the truth table of $\mathcal{D}$. Run ${}_\mathcal{D}(\mathsf{y})$ to get $b = \mathcal{D}(y)$, and output b.[9]

To analyze the success probability of $\mathcal{D}'$ in breaking the hardness of $\widetilde{\mathsf{TLP}}$, we observe that the only difference between running $\mathcal{D}'^{\mathcal{O}'}$ and running $\mathcal{D}$ on the output of $\widetilde{\mathsf{bmim}}_{\mathcal{A},F}^{\mathcal{O}'}$ is that the puzzle z given to $\mathcal{D}'$ is sampled using $\mathcal{O}'$, while the puzzle that bmim uses is sampled using $\mathcal{O}'$, but replacing any queries to F with the image given in F. Specifically, let z be the puzzle given as input to $\mathcal{D}'$ and let r be randomness sampled for running Gen. In the first case, where z is a puzzle for s, then $z = \mathsf{Gen}_{\mathsf{tlp}}(1^\lambda, t, (s||r); \mathcal{O}'(s, r))$. Whenever $(s, r) \notin F$, it follows that the output of $\mathcal{D}'$ is distributed as in the first probability in Equation 4.8. By the same argument, when z is a puzzle for 0^λ then the output of $\mathcal{D}'$ is distributed according to the second probability in Equation 4.8, so long as $(0^\lambda, r) \notin F$. As r is chosen uniformly at random from $\{0,1\}^{\lambda_{\mathsf{tlp}}}$, it follows that the probability that this occurs is at most

$$\frac{f_{\mathsf{nm}}(\lambda)}{2^{\lambda_{\mathsf{tlp}}}} = \frac{(q_\mathcal{B}(\lambda, 2^m) \cdot 2q(\lambda))^2}{2^{\lambda_{\mathsf{tlp}}}} = \frac{(q_\mathcal{B}(\lambda, 2^m) \cdot 2q(\lambda))^2}{2^{m+\lambda}} \in \frac{\mathrm{poly}(\lambda, 2^m)}{2^{m+\lambda}} \leq \mathsf{negl}'(\lambda)$$

which is negligible, where we used the fact that $\lambda_{\mathsf{tlp}} = m + \lambda$. Therefore

$$\left|\Pr\left[\begin{array}{l}\mathcal{O}' \leftarrow \mathsf{RF}_{2\lambda+m}^{\lambda+m} \\ z \leftarrow \mathsf{Gen}^{\mathcal{O}'}(1^\lambda, t, s)\end{array} : \mathcal{D}'^{\mathcal{O}'}(z) = 1\right]\right.$$
$$\left. - \Pr\left[\begin{array}{l}\mathcal{O}' \leftarrow \mathsf{RF}_{2\lambda+m}^{\lambda+m} \\ z \leftarrow \mathsf{Gen}^{\mathcal{O}'}(1^\lambda, t, 0^\lambda)\end{array} : \mathcal{D}'^{\mathcal{O}'}(z) = 1\right]\right| \geq \mu(\lambda) - \mathsf{negl}'(\lambda).$$

[9]This trick, of "flattening" $\mathcal{D}$ using its truth table, was used in this context by [DKP20].

It will follow that $\mathcal{D}'$ succeeds at breaking the hardness of nmTLP_m for T as long as its size is in $\mathrm{poly}(\lambda, B(\lambda))$, its depth is bounded by $T(\lambda)/(\alpha_{\mathsf{hard}}(\lambda))$, and $\alpha_{\mathsf{hard}}(\lambda) \leq T(\lambda) \in \mathrm{poly}(\lambda, B(\lambda))$ for all $\lambda \in \mathbb{N}$, where α_{hard} is the positive polynomial guaranteed by the hardness of nmTLP_m. We bound the size and depth required for each step of $\mathcal{D}'$ next, and then bound T. For the size and depth of $\mathcal{D}'$, we have that:

1. The first step requires running $\mathcal{A}$ and checking if each of its queries appears in F, and if so returning the correct answer, and so requires size $\mathrm{poly}(|\mathcal{A}|, |F|)$ and can be done in depth $\mathsf{depth}(\mathcal{A}) \cdot \mathrm{poly}(\lambda, \log |F|)$ since for each of $\mathcal{A}$'s queries, it takes depth $\mathrm{poly}(\lambda, \log |F|)$ to check in parallel if the query appears in F and output the answer.

2. The second step can be done with size $\mathrm{poly}(\lambda, m, \log t, |\mathcal{Q}|)$ and depth $\mathrm{poly}(\lambda, m, \log t)$ since it requires generating a puzzle in parallel for each $i \in [n(\lambda)]$ and $j \in [|\mathcal{Q}|]$.

3. The third step requires computing $y = f(\vec{\tilde{s}})$. Since the input length is $n(\lambda)$, this can be done in depth $\mathrm{poly}(\lambda, \log n(\lambda))$ and size $\mathrm{poly}(\lambda, B(\lambda))$ by assumption on $f \in \mathcal{F}_{B,m}$.

4. The third step requires running $\mathsf{tt}_{\mathcal{D}'}(y)$, which can be done with size $\mathrm{poly}(\lambda, 2^m)$ and depth $\mathrm{poly}(\lambda, m)$.

Putting everything together, we have that

$$\begin{aligned}\mathsf{size}(\mathcal{D}') &\in \mathrm{poly}(\lambda, m, n(\lambda), B(\lambda), \log t, |\mathcal{Q}|, |\mathcal{A}|, |F|, 2^m) \in \mathrm{poly}(\lambda, m, B(\lambda), f_{\mathsf{nm}}(\lambda), 2^m) \\ &\in \mathrm{poly}(\lambda, m, 2^m) \in \mathrm{poly}(\lambda, B(\lambda)),\end{aligned}$$

where we used the fact that $n(\lambda) \in \lambda^{O(1)}$, $B(\lambda) = 2^m \cdot \mathrm{poly}(\lambda)$, and $|F| = f_{\mathsf{nm}}(\lambda) = (q_{\mathcal{B}}(\lambda, 2^m) \cdot 2q(\lambda))^2$. For the depth,

$$\begin{aligned}\mathsf{depth}(\mathcal{D}') &\leq \mathsf{depth}(\mathcal{A}) \cdot \mathrm{poly}(\lambda, \log |F|) + \mathrm{poly}(\lambda, m, \log n(\lambda), \log T(\lambda)) \\ &\leq \mathsf{depth}(\mathcal{A}) \cdot \mathrm{poly}(\lambda, m, \log n(\lambda)) \leq \mathsf{depth}(\mathcal{A}) \cdot p_2(\lambda)\end{aligned}$$

for a fixed polynomial p_2 (which depends only on m, $\log n$, and $\log f_{\mathsf{hard}}(\lambda)$), where we used the fact that $m(\lambda) \in \lambda^{O(1)}$ and $T(\lambda) \in \mathrm{poly}(\lambda, B(\lambda))$. Note that $\log(n(\lambda))$ can be bounded by λ for sufficiently large λ. Similarly, $\log f_{\mathsf{hard}}(\lambda)$ can be bounded by a fixed polynomial in λ which depends on $\log q(\lambda)$. As q is a polynomial, this can also be bounded by a fixed polynomial in λ independently of q, for sufficiently large λ. Moreover, we can simply have $\mathcal{D}'$ output $\perp$ on security parameters which are not sufficiently large. Therefore, $\mathsf{depth}(\mathcal{D}') \leq \mathsf{depth}(\mathcal{A}) \cdot p_2(\lambda)$ for all $\lambda \in \mathbb{N}$. Recall that $\mathsf{depth}(\mathcal{A}) \leq T(\lambda)/\alpha(\lambda)$, where we set $\alpha(\lambda) \geq \alpha_{\mathsf{hard}} \cdot p_2(\lambda)$. Therefore, the above is bounded by $T(\lambda)/(\alpha_{\mathsf{hard}}(\lambda))$.

Finally, to bound T, we have that $T(\lambda) \geq \alpha(\lambda) \geq \alpha_{\mathsf{hard}}(\lambda) \cdot p_2(\lambda) \geq \alpha_{\mathsf{hard}}(\lambda)$ since we can set $p_2(\lambda) \geq 1$, and $T(\lambda) \in \mathrm{poly}(\lambda, B(\lambda))$ by assumption. It follows that $\mathcal{D}'$ breaks the hardness of nmTLP_m with probability $\mu(\lambda) - \mathsf{negl}'(\lambda)$, which completes the claim. ■

This completes the proof of Lemma 4.11. □

4.4 Impossibility of Fully Concurrent Non-malleability

In the following theorem, we show that there does not exist a concurrent non-malleable time-lock puzzle. Specifically, consider a time-lock puzzle with message length L that has puzzles of length at most L'. We give an explicit attack that violates n-concurrent non-malleability for $n = \lceil L'/L \rceil$.

As L, L' are polynomial, this is an explicit polynomial for which non-malleability cannot hold. Furthermore, this attack works even in the random oracle model. This means that for a time-lock puzzle to satisfy n-concurrent non-malleability, the puzzles must be sufficiently long. In the context of functional non-malleability, the following theorem shows that concurrent functional non-malleability is impossible to achieve for any class $\mathcal{F}$ that contains the identity function.

Theorem 4.17. *Let $B, L, L' \colon \mathbb{N} \to \mathbb{N}$ where $B(\lambda) \in 2^{\mathrm{poly}(\lambda)}$ and $L \in \mathrm{poly}(\lambda)$. Suppose that $(\mathsf{Gen}, \mathsf{Sol})$ is a B-secure time-lock puzzle for messages of length $L(\lambda)$ with puzzles of length at most $L'(\lambda)$. Then, $(\mathsf{Gen}, \mathsf{Sol})$ does not satisfy $n(\lambda)$-concurrent non-malleability for $n(\lambda) = \lceil L'(\lambda)/L(\lambda) \rceil$.*

Proof. We note that as n-concurrent non-malleability implies one-n non-malleability, it suffices to break one-n non-malleability.

For any $\lambda \in \mathbb{N}$, let $L = L(\lambda)$, $L' = L'(\lambda)$, and $n = n(\lambda) = \lceil L'/L \rceil$. Note that, since Gen is a PPT algorithm, L' is at most $\mathrm{poly}(\lambda, L(\lambda), \log T(\lambda))$ for any T. This implies that L' and n are both bounded by polynomials.

Let α be any positive polynomial and T be any function with $\alpha(\lambda) \leq T(\lambda) \in \mathrm{poly}(\lambda, B(\lambda))$. We define a $(1, n, B, \alpha, T)$-MIM adversary $\mathcal{A} = \{\mathcal{A}_\lambda\}_{\lambda \in \mathbb{N}}$ as follows. On input a puzzle $z \in \{0,1\}^{L'}$, $\mathcal{A}_\lambda$ splits z into n parts $z_1, \ldots, z_n \in \{0,1\}^L$, padding the last part with zeroes if necessary. $\mathcal{A}_\lambda$ outputs $\widetilde{z}^{(1)}, \ldots, \widetilde{z}^{(n)}$ where $\widetilde{z}^{(i)} \leftarrow \mathsf{Gen}(1^\lambda, T(\lambda), z_i)$ for all $i \in [n]$. Note that the size (and depth) of $\mathcal{A}_\lambda$ is at most $n \cdot \mathrm{poly}(\lambda, \log T(\lambda)) \in n \cdot q(\lambda)$ for some polynomial q since Gen is a PPT algorithm, $T(\lambda) \in \mathrm{poly}(\lambda, B(\lambda))$, and $B(\lambda) \in 2^{\mathrm{poly}(\lambda)}$. Thus, $\mathcal{A}$ is a valid $(1, n, B, \alpha, T)$-MIM adversary for any T such that $n \cdot q(\lambda) \leq T(\lambda)/\alpha(\lambda)$. In particular, this is true for $T(\lambda) = n \cdot q(\lambda) \cdot \alpha(\lambda)$. We show for this function T, $\mathcal{A}$ violates one-n non-malleability.

For any $\lambda \in \mathbb{N}$ and message $s \in \{0,1\}^\lambda$, consider the following distinguisher $\mathcal{D}$ for the MIM distribution of $\mathcal{A}_\lambda$. $\mathcal{D}$ gets as input values $\widetilde{s}^{(1)}, \ldots, \widetilde{s}^{(n)}$ corresponding to $(\widetilde{z}^{(1)}, \ldots \widetilde{z}^{(n)})$. $\mathcal{D}$ computes $\hat{z}$ to be the first L' bits of $\widetilde{s}^{(1)} \| \ldots \| \widetilde{s}^{(n)}$ $\mathcal{D}$ then computes $s' = \mathsf{Sol}(1^\lambda, T(\lambda), \hat{z})$. $\mathcal{D}$ outputs 1 if $s = s'$ and 0 otherwise. By correctness of $(\mathsf{Gen}, \mathsf{Sol})$, it holds that $\mathcal{D}$ outputs 1 only on input $\mathsf{mim}_{\mathcal{A}}(1^\lambda, T(\lambda), s)$, which contradicts one-$n$ non-malleability of $(\mathsf{Gen}, \mathsf{Sol})$, as required. Furthermore, we note that $\mathcal{D}$ only needs to run in depth $T(\lambda) \cdot \mathrm{poly}(\lambda, \log T(\lambda))$. □

5 Publicly Verifiable Non-Malleable Time-Lock Puzzles

In this section, we define and construct a publicly verifiable time-lock puzzle (PV TLP) that is additionally non-malleable as in Section 4. At a high level, a PV TLP is one where the Sol function additionally outputs a succinct proof of correctness that can be checked by a Verify function.

In order for the proof to be sound, we use a weak form of setup independent of a cheating prover's advice. Specifically, we rely on what we call the all-but-one (ABO) string model. In this model, the Sol and Verify algorithms take as input a multi-common random string, $\mathsf{mcrs} \in (\{0,1\}^\lambda)^n$ for some $n \in \mathbb{N}$, and we require soundness to hold as long as a single random string is honest. In the case that $n = 1$, this corresponds to the standard common random string model. The ABO-string model is also very related to the multi-string model of [GO14], except that the ABO-string model requires that only one string—as opposed to a majority of strings—is honestly generated.

While the ABO-string model is a weak form of setup, we prove security in the relatively strong auxiliary-input random oracle model (AI-ROM). At a high level, we do this to ensure that the puzzles generated by Gen are independent of *any* setup, while also being able to prove a meaningful notion of security in essentially the "plain" random oracle model. Furthermore, the combination is realistic for our application in Section 6.

We define a publicly verifiable time-lock puzzle in the ABO-string model as follows. We note that we don't explicitly define the functions and properties with respect a random oracle, and defer such a treatment to the proofs of security.

Definition 5.1 (Publicly Verifiable Time-Lock Puzzle)**.** *A tuple* $(\mathsf{Gen}, \mathsf{Sol}, \mathsf{Verify})$ *is a* B-secure publicly-verifiable time-lock puzzle in the ABO-string model *if* $(\mathsf{Gen}, \mathsf{Sol}, \mathsf{Verify})$ *have the following syntax:*

- $z \leftarrow \mathsf{Gen}(1^\lambda, t, s)$*: A PPT algorithm that on input a security parameter* $\lambda \in \mathbb{N}$*, a difficulty parameter* $t \in \mathbb{N}$*, and a solution* $s \in \{0,1\}^\lambda$*, outputs a puzzle* $z \in \{0,1\}^*$*.*
- $(s, \pi) \leftarrow \mathsf{Sol}(1^\lambda, \mathsf{mcrs}, t, z)$*: A randomized algorithm that on input the security parameter* $\lambda \in \mathbb{N}$*, a multi-common random string* $\mathsf{mcrs} \in (\{0,1\}^\lambda)^*$*, a difficulty parameter* $t \in \mathbb{N}$*, and a puzzle* z*, outputs a solution* $s \in (\{0,1\}^\lambda \cup \{\perp\})$ *and a proof* $\pi \in \{0,1\}^*$*. We denote* Sol_1 *and* Sol_2 *as the first and second outputs of* Sol*, respectively.*
- $b = \mathsf{Verify}(1^\lambda, \mathsf{mcrs}, t, z, (s, \pi))$*: A deterministic algorithm that on input a security parameter* $\lambda \in \mathbb{N}$*, a multi-common random string* $\mathsf{mcrs} \in (\{0,1\}^\lambda)^*$*, a difficulty parameter* $t \in \mathbb{N}$*, a puzzle* z*, a possible solution* $s \in (\{0,1\}^\lambda \cup \{\perp\})$*, and a proof* π*, outputs a bit* b *indicating whether to accept or reject.*

We require that $(\mathsf{Gen}, \mathsf{Sol}, \mathsf{Verify})$ *satisfy the following properties.*

- **Full correctness:** *For every* $\lambda, t, n \in \mathbb{N}$, $\mathsf{mcrs} \in (\{0,1\}^\lambda)^n$*, and* $z \in \{0,1\}^*$*, the following hold:*
  - *If* $z \in \mathrm{Supp}\left(\mathsf{Gen}(1^\lambda, t, s)\right)$ *for some* $s \in \{0,1\}^\lambda$*, then* $\mathsf{Sol}_1(1^\lambda, \mathsf{mcrs}, t, z) = s$*.*
  - *If* $z \notin \mathrm{Supp}\left(\mathsf{Gen}(1^\lambda, t, s)\right)$ *for any* $s \in \{0,1\}^\lambda$*, then* $\mathsf{Sol}_1(1^\lambda, \mathsf{mcrs}, t, z) = \perp$*.*
- **Efficiency:** *There exist a polynomial* p *such that for all* $\lambda, t, n \in \mathbb{N}$*, and* $\mathsf{mcrs} \in (\{0,1\}^\lambda)^n$*, it holds that* $\mathsf{Sol}(1^\lambda, \mathsf{mcrs}, t, \cdot)$ *is computable in time* $t \cdot p(\lambda, \log t, n)$*.*
- B**-Hardness:** *The same as for time-lock puzzles as in Definition 3.1.*
- **Completeness:** *For any* $\lambda, t, n \in \mathbb{N}$*,* $z \in \{0,1\}^*$*,* $\mathsf{mcrs} \in (\{0,1\}^\lambda)^n$*, it holds that*
$$\mathsf{Verify}(1^\lambda, \mathsf{mcrs}, t, z, \mathsf{Sol}(1^\lambda, \mathsf{mcrs}, t, z)) = 1.$$
- **Soundness:** *For all non-uniform probabilistic polynomial-time adversaries* $\mathcal{A} = \{\mathcal{A}_\lambda\}_{\lambda \in \mathbb{N}}$ *and polynomials* T*, there exists a negligible function* negl *such that for all* $\lambda, n \in \mathbb{N}$ *and* $i \in [n]$*, it holds that*
$$\Pr\left[\begin{array}{l} \mathsf{crs}_i \leftarrow \{0,1\}^\lambda \\ (z, s', \pi, \mathsf{mcrs}_{-i}) \leftarrow \mathcal{A}_\lambda(1^\lambda, \mathsf{crs}_i, T(\lambda)) \\ \mathsf{mcrs} = (\mathsf{crs}_1, \ldots, \mathsf{crs}_n) \\ s = \mathsf{Sol}_1(1^\lambda, \mathsf{mcrs}, T(\lambda), z) \end{array} : \begin{array}{l} \mathsf{Verify}(1^\lambda, \mathsf{mcrs}, T(\lambda), z, (s', \pi)) = 1 \\ \wedge\ s \neq s' \end{array}\right]$$
$$\leq \mathsf{negl}(\lambda),$$
where mcrs_{-i} *is a tuple of* $n-1$ *common random strings* $(\mathsf{crs}_1, \ldots, \mathsf{crs}_{i-1}, \mathsf{crs}_{i+1}, \ldots, \mathsf{crs}_n)$*.*

We note that the above notion of full correctness is stronger than the standard definition of correctness for TLPs given in Definition 3.1. Also, the soundness notion is strong in the sense that the adversary can try to break soundness even for invalid puzzles. We emphasize that puzzles can be generated independently of the setup mcrs, as the setup is only used for soundness of the proof given by Sol.

Towards achieving this strong definition, we define a notion of a *one-sided* publicly verifiabble time-lock puzzle in which correctness holds only for z in the support of Gen and soundness requires that the adversary cheats on puzzles in the support of Gen. We formalize this as follows.

Definition 5.2 (One-sided PV TLP)**.** *A tuple* $(\mathsf{Gen}, \mathsf{Sol}, \mathsf{Verify})$ *is a* B-secure one-sided publicly-verifiable time-lock puzzle in the ABO-string model *if correctness holds only for* z *in the support of* $\mathsf{Gen}(1^\lambda, t, \cdot)$ *and the soundness property is replaced with the following:*

- **One-sided Soundness:** *For all non-uniform probabilistic polynomial-time adversaries* $\mathcal{A} = \{\mathcal{A}_\lambda\}_{\lambda \in \mathbb{N}}$ *and polynomials* T*, there exists a negligible function* negl *such that for all* $\lambda, n \in \mathbb{N}$ *and* $i \in [n]$*, it holds that*

$$\Pr\left[\begin{array}{l} \mathsf{crs}_i \leftarrow \{0,1\}^\lambda \\ (z, s', \pi, \mathsf{mcrs}_{-i}) \leftarrow \mathcal{A}_\lambda(1^\lambda, \mathsf{crs}_i, T(\lambda)) \\ \mathsf{mcrs} = (\mathsf{crs}_1, \dots, \mathsf{crs}_n) \\ s = \mathsf{Sol}_1(1^\lambda, \mathsf{mcrs}, T(\lambda), z) \end{array} : \begin{array}{l} \mathsf{Verify}(1^\lambda, \mathsf{mcrs}, T(\lambda), z, (s', \pi)) = 1 \\ \wedge\ s \neq s' \\ \wedge\ z \in \mathrm{Supp}\big(\mathsf{Gen}(1^\lambda, T(\lambda), \cdot)\big) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

In Section 5.1, we formalize the notion of a strong trapdoor verifiable delay function (VDF), which satisfies the requirements needed for our one-sided PV TLP construction. We then give a construction of a strong trapdoor VDF based on the repeated squaring assumption. We note that it may be possible to give a construction that satisfies the necessary properties based on randomized encodings as in [BGJ+16], but we instead focus on a more practical construction.

In Section 5.2, we formalize the construction of a one-sided PV TLP given a strong trapdoor VDF. Finally in Section 5.3, we construct a full PV TLP by applying our non-malleability transformation of Section 4.

5.1 Strong Trapdoor VDFs

A trapdoor VDF provides a way to generate inputs to a function that takes a long time to compute. At the same time, the function can be computed efficiently using a trapdoor, and even without the trapdoor, can be efficiently verified given a proof. Trapdoor VDFs were first defined by Wesolowski [Wes19] as an extension of standard VDFs [BBBF18, Pie19, Wes19, FMPS19].

We define a strong notion of a trapdoor VDF in the ABO-string model. While the formal definition is highly tailored towards our definition of PV TLPs and application to multi-party coin-flipping, we believe that some of the stronger requirements we define—and achieve—may be of independent interest. Conceptually, we require the following additional properties over previous definitions of VDFs (or trapdoor VDFs):

1. We have no setup algorithm, and instead are in the ABO-string model. In particular, this means that sampling inputs for the VDF can be done completely independently of any trusted setup.

2. We allow the sampling procedure to specify the domain $\mathcal{X}$ of the evaluation function.

3. We require that completeness must hold for all inputs $x \in \{0,1\}^*$ and domains $\mathcal{X} \in \{0,1\}^*$ rather than only honestly generated inputs.

4. We require soundness to hold for any adversarially chosen—yet in the support of the Sample algorithm—input x and domain $\mathcal{X}$, rather than with high probability over randomly sampled x and $\mathcal{X}$.

5. We require a natural encoding property for elements in the domain $\mathcal{X}$ with strings, and additionally require that $\mathcal{X}$ defines a natural bijection that is amenable to being used as a one-time pad (like $\oplus$ for strings or $+$ for rings, for example).

We formally define the requirements of a strong trapdoor VDF in the ABO-string model as follows.

Definition 5.3. *Let $B\colon \mathbb{N} \to \mathbb{N}$. A B-*secure strong trapdoor verifiable delay function in the ABO-string model *is a tuple* $(\mathsf{Sample}, \mathsf{Eval}, \mathsf{TDEval}, \mathsf{Verify})$ *with the following syntax:*

- $(x, \mathcal{X}, \mathsf{td}) \leftarrow \mathsf{Sample}(1^\lambda, t)$**:** *A PPT algorithm that on input a security parameter $\lambda \in \mathbb{N}$, a difficulty parameter $t \in \mathbb{N}$, outputs a value x in a specified domain $\mathcal{X}$, and a trapdoor $\mathsf{td} \in \{0,1\}^*$.*
- $(y, \pi) \leftarrow \mathsf{Eval}(1^\lambda, \mathsf{mcrs}, t, (x, \mathcal{X}))$**:** *An algorithm that on input a security parameter $\lambda \in \mathbb{N}$, a difficulty parameter $t \in \mathbb{N}$, and a value x in a specified domain $\mathcal{X}$, outputs a value $y \in \mathcal{X}$ and a proof $\pi \in \{0,1\}^*$. We denote Eval_1 and Eval_2 as the first and second outputs of Eval, respectively, and require that Eval_1 can be implemented as a deterministic function.*
- $y = \mathsf{TDEval}(1^\lambda, t, (x, \mathcal{X}), \mathsf{td})$**:** *A polynomial-time algorithm that on input a security parameter $\lambda \in \mathbb{N}$, a difficulty parameter $t \in \mathbb{N}$, a value x in a specified domain $\mathcal{X}$, and a trapdoor $\mathsf{td} \in \{0,1\}^*$, outputs a value $y \in \mathcal{X}$.*
- $b = \mathsf{Verify}(1^\lambda, \mathsf{mcrs}, t, (x, \mathcal{X}), (y, \pi))$**:** *A polynomial-time algorithm that on input a security parameter $\lambda \in \mathbb{N}$, a difficulty parameter $t \in \mathbb{N}$, values x, y in a specified domain $\mathcal{X}$, and a proof $\pi \in \{0,1\}^*$, outputs a bit b indicating whether to accept or reject.*

We require that $(\mathsf{Sample}, \mathsf{Eval}, \mathsf{TDEval}, \mathsf{Verify})$ *satisfy the following properties.*

- **Completeness:** *For every $\lambda, t, n \in \mathbb{N}$, $x, \mathcal{X} \in \{0,1\}^*$, $\mathsf{mcrs} \in (\{0,1\}^\lambda)^n$, it holds that*

$$\mathsf{Verify}(1^\lambda, \mathsf{mcrs}, t, (x, \mathcal{X}), \mathsf{Eval}(1^\lambda, \mathsf{mcrs}, t, (x, \mathcal{X}))) = 1.$$

- **Soundness:** *For all non-uniform PPT adversaries $\mathcal{A} = \{\mathcal{A}_\lambda\}_{\lambda \in \mathbb{N}}$ and polynomials T, there exists a negligible function* negl *such that for all $\lambda, n \in \mathbb{N}$, $i \in [n]$, it holds that*

$$\Pr\left[\begin{array}{lcl} \mathsf{crs}_i \leftarrow \{0,1\}^\lambda & & \\ (x, \mathcal{X}, y', \pi, \mathsf{mcrs}_{-i}) & & \mathsf{Verify}(1^\lambda, \mathsf{mcrs}, T(\lambda), (x, \mathcal{X}), (y', \pi)) = 1 \\ \qquad \leftarrow \mathcal{A}_\lambda(1^\lambda, \mathsf{crs}_i, T(\lambda), n) & : & \wedge\ y \neq y' \\ \mathsf{mcrs} = (\mathsf{crs}_1, \dots, \mathsf{crs}_n) & & \wedge\ (x, \mathcal{X}, *) \in \mathrm{Supp}\big(\mathsf{Sample}(1^\lambda, T(\lambda))\big) \\ y = \mathsf{Eval}_1(1^\lambda, \mathsf{mcrs}, T(\lambda), (x, \mathcal{X})) & & \end{array}\right]$$
$$\leq \mathsf{negl}(\lambda).$$

- **Trapdoor Evaluation:** *For every $\lambda, t, n \in \mathbb{N}$, $(x, \mathcal{X}, \mathsf{td}) \in \mathrm{Supp}\big(\mathsf{Sample}(1^\lambda, t)\big)$, and $\mathsf{mcrs} \in (\{0,1\}^\lambda)^n$, it holds that*

$$\mathsf{Eval}_1(1^\lambda, \mathsf{mcrs}, t, (x, \mathcal{X})) = \mathsf{TDEval}(1^\lambda, t, (x, \mathcal{X}), \mathsf{td}).$$

- **Honest Evaluation:** *There exists a polynomial p such that for all $\lambda, t, n \in \mathbb{N}$, and $\mathsf{mcrs} \in (\{0,1\}^\lambda)^n$, $\mathsf{Eval}(1^\lambda, \mathsf{mcrs}, t, \cdot)$ is computable in time $t \cdot p(\lambda, \log t, n)$.*

- **B-Sequentiality:** *There exists a positive polynomial function α such that for all functions T and non-uniform adversaries $\mathcal{A} = \{\mathcal{A}_\lambda\}_{\lambda\in\mathbb{N}}$ satisfying $\alpha(\lambda) \leq T(\lambda) \in \text{poly}(\lambda, B(\lambda))$, $\mathsf{size}(\mathcal{A}_\lambda) \in \text{poly}(\lambda, B(\lambda))$, and $\mathsf{depth}(\mathcal{A}_\lambda) \leq T(\lambda)/\alpha(\lambda)$ for all $\lambda \in \mathbb{N}$, there exists a negligible function negl such that for all $\lambda \in \mathbb{N}$, $n \in \mathbb{N}$, $\mathsf{mcrs} \in (\{0,1\}^\lambda)^n$,*

$$\left| \Pr\left[\begin{array}{l} (x, \mathcal{X}, \mathsf{td}) \leftarrow \mathsf{Sample}(1^\lambda, T(\lambda)) \\ y = \mathsf{Eval}_1(1^\lambda, \mathsf{mcrs}, T(\lambda), (x, \mathcal{X})) \end{array} : \mathcal{A}_\lambda(x, \mathcal{X}, y) = 1 \right] \right.$$
$$\left. - \Pr\left[\begin{array}{l} (x, \mathcal{X}, \mathsf{td}) \leftarrow \mathsf{Sample}(1^\lambda, T(\lambda)) \\ r \leftarrow \mathcal{X} \end{array} : \mathcal{A}_\lambda(x, \mathcal{X}, r) = 1 \right] \right| \leq \mathsf{negl}(\lambda).$$

- **Encoding:** *For $\lambda, t \in \mathbb{N}$, and any domain $\mathcal{X}$ output by $\mathsf{Sample}(1^\lambda, t)$, it holds that strings in $\{0,1\}^\lambda$ can be uniquely encoded as elements in $\mathcal{X}$, and elements in $\mathcal{X}$ can be uniquely decoded to elements in $\{0,1\}^*$. Additionally, for any element $x \in \mathcal{X}$, there is an efficiently computable bijective map $f_x \colon \mathcal{X} \to \mathcal{X}$, written as $f_x(y) = x \oplus y$.*

In the above definition, we note that only Eval and Verify receive as input the multi-common random string mcrs, as they require it for public verifiability. In particular, TDEval can be computed without access to mcrs since we do not require it to output a proof of correctness (in fact, the trapdoor itself can be thought of as a *privately verifiable* proof). By trapdoor evaluation, this implies that the output y of the function is actually independent of mcrs. Lastly, we note that we've adapted the notion of sequentiality for VDFs to fit with our notion of hardness for time-lock puzzles.

Candidate strong trapdoor VDF. Our candidate strong trapdoor VDF is based on repeated squaring with a publicly verifiable proof of correctness from Pietrzak [Pie19]. As in [Pie19], we use the group $\mathbb{G} = \text{QR}_N^+$ of signed quadratic residues mod N, where N is the product of safe primes p, q such that $(p-1)/2$ and $(q-1)/2$ are λ-bit primes. We note that QR_N^+ has size $|\text{QR}_N^+| = (p-1)\cdot(q-1)/4$ and has the property that it only subgroups are of size $(p-1)/2$ and $(q-1)/2$ which are both at least 2^λ by construction. Elements in this group can be encoded as integers in $[0, (N-1)/2]$. For any two elements $a, b \in \text{QR}_N^+$, we can define multiplication (and similarly addition) by computing $x = a \cdot b \bmod N$ and taking the smaller of x and $N - x$, which is in $[0, (N-1)/2]$. For further discussion of the group, we refer the reader to [Pie19].

To generate such a group, we can sample random numbers in $[2^{\lambda+1}, 2^{\lambda+2})$ until we find two safe primes p and q and output QR_N^+ where $N = p \cdot q$. As discussed in [Pie19], it is conjectured (in [VZGS13]) that for some constant c, there are $c \cdot 2^\lambda/\lambda^2$ safe λ-bit primes. Under this conjecture, it takes expected polynomial time to sample N which is a product of two safe primes. Rather than doing this, we define a group generator algorithm $\mathsf{RSWGen}(1^\lambda)$ that samples a number in $[2^{\lambda+1}, 2^{\lambda+2})$, checks if it is a safe prime, and repeats this process for some *fixed* polynomial time until it fails to halt with probability at most $2^{-\lambda}$. Specifically, suppose it takes expected $p(\lambda)$ time to find two safe primes. If we run for $2 \cdot \lambda \cdot p(\lambda)$ time, we will halt with probability at least $1 - 2^{-\lambda}$. When this failure event occurs, $\mathsf{RSWGen}(1^\lambda)$ deterministically searches for safe primes starting with $2^{\lambda+1}, 2^{\lambda+1} + 1, \ldots$ until finding the first two safe primes (alternatively, we could hard code these safe primes with preprocessing). Assuming that the safe primes are evenly spread out, this will only need to search through $O(\lambda^2)$ numbers. We additionally define $\mathsf{RSWGen}(1^\lambda)$ to output a random group element and the size of the group to be used as a trapdoor. In summary, we have the following:

- $(g, \mathbb{G}, |\mathbb{G}|) \leftarrow \mathsf{RSWGen}(1^\lambda)$: An algorithm that outputs $(g, \mathbb{G}, |\mathbb{G}|)$ where $\mathbb{G} = \text{QR}_N^+$ such that N is the product of two safe primes p and q, $g \leftarrow \mathbb{G}$ is a random element in the group, and

$|\mathbb{G}| = (p-1)\cdot(q-1)/4$ is the size of the group. It holds that with probability at least $1-2^{-\lambda}$, p and q are uniformly random safe primes in $[2^{\lambda+1}, 2^{\lambda+2})$.

We next formalize the repeated squaring assumption we make for RSWGen.

Assumption 5.4 (Repeated Squaring Assumption for RSWGen)**.** *Let $B\colon \mathbb{N}\to\mathbb{N}$. We say that the B-*repeated squaring assumption for RSWGen holds *if there exists a PPT algorithm implementing RSWGen and there exists a positive polynomial function α such that for all functions T and non-uniform distinguishers $\mathcal{A} = \{\mathcal{A}_\lambda\}_{\lambda\in\mathbb{N}}$ satisfying $\alpha(\lambda) \le T(\lambda) \in \mathrm{poly}(\lambda, B(\lambda))$, $\mathsf{size}(\mathcal{A}_\lambda) \in \mathrm{poly}(\lambda, B(\lambda))$, and $\mathsf{depth}(\mathcal{A}_\lambda) \le T(\lambda)/\alpha(\lambda)$ for all $\lambda\in\mathbb{N}$, there exists a negligible function negl such that for all $\lambda\in\mathbb{N}$,*

$$\Bigg| \Pr\left[\begin{array}{l}(g,\mathbb{G},|\mathbb{G}|)\leftarrow \mathsf{RSWGen}(1^\lambda)\\ y = g^{2^{T(\lambda)}}\end{array} : \mathcal{A}_\lambda(g,\mathbb{G},y)=1\right] - \Pr\left[\begin{array}{l}(g,\mathbb{G},|\mathbb{G}|)\leftarrow \mathsf{RSWGen}(1^\lambda)\\ r\leftarrow \mathbb{G}\end{array} : \mathcal{A}_\lambda(g,\mathbb{G},r)=1\right]\Bigg| \le \mathsf{negl}(\lambda).$$

We emphasize again that the assumption that RSWGen can be implemented by a PPT algorithm follows from a conjecture of [VZGS13] about the density of safe primes. As discussed in [Pie19], the hardness assumption for repeated squaring in QR_N^+ is implied by the more standard hardness assumption for repeated squaring in $\mathbb{Z}_N^\star$ with at most a factor of 8 loss in advantage. Recent works [RS20, KLX20] show that generically speeding-up repeated squaring is equivalent to factoring. Rotem and Segev [RS20] show this in a generic-ring model relative to an RSA modulus. Katz et al. [KLX20] show this within a strengthened version of the algebraic group model model [FKL18] relative to the group of quadratic residues.

Remark 1 (Choice of Group)**.** *As pointed out by Boneh et al. [BBF18], it is possible to instantiate Pietrzak's proof of repeated squaring [Pie19] with any group that does not have low order elements while still maintaining statistical soundness. As done in [Pie19] and is common in this line of work, we use the group QR_N^+ where N is a product of two $(\lambda+1)$-bit safe primes because its smallest subgroups have size at least 2^λ. However, it is unknown whether a random such group can be sampled in fixed polynomial time.*

Proof of repeated squaring. We next discuss the publicly verifiable proof of repeated squaring for the group QR_N^+ given in Pietrzak [Pie19]. We use a slightly modified version of this protocol, which is in the ABO-string model and where we explicitly assume access to a random oracle $\mathcal{O}\colon\{0,1\}^*\to[2^\lambda]$.

We first describe the interactive protocol of [Pie19]. In order to prove the claim that $x^{2^T} = y$ in QR_N^+[10], the interactive protocol does the following. If $T=1$, the verifier directly checks that $x^2 = y$. Otherwise, the prover sends to the verifier the value $\mu = x^{2^{T/2}}$ and the verifier replies with $r\leftarrow[2^\lambda]$. The prover and verifier recursively engage in a protocol to prove that $(x^r\mu)^{T/2} = \mu^r y$. This interactive proof consists of $2\cdot\log T$ rounds of communication, where the prover sends a single group element and the verifier responds with a λ-bit challenge. The proof has soundness error at most $3\cdot\log T/2^\lambda$ even against unbounded cheating provers. In particular, [Pie19] shows that at there are at most 3 "bad challenges" that the verifier might send in each round of the protocol, in the sense that a bad challenge might cause the verifier to wrongly accept false statements.

In order to make the above protocol non-interactive via the Fiat-Shamir heuristic, the prover uses the random oracle $\mathcal{O}$ on the transcript so far to generate the random challenges of the verifier

[10]we assume for simplicity that T is a power of 2, which can be easily dealt with as in [Pie19] if this is not the case

itself. The verifier then accepts the proof if all challenges are consistent with $\mathcal{O}$ and the interactive verifier would have accepted. We emphasize that in our version of the protocol, the oracle $\mathcal{O}$ is indexed with mcrs, which is assumed to have at least λ bits of entropy *independent* of any possible adversary. Following the above idea, we formalize the algorithms $\mathsf{RSWProve}$ and $\mathsf{RSWVerify}$ that we use:

- $\pi = \mathsf{RSWProve}^{\mathcal{O}}(1^\lambda, \mathsf{mcrs}, t, (g, \mathbb{G}), y)$:

```
1. Let $x_0 = g$, $y_0 = y$.
2. For $i = 1, \dots, \log t$,
    (a) Compute $\pi_i = x_{i-1}^{2^{t/2^i}}$.
    (b) Let $r_i = \mathcal{O}(1^\lambda, \mathsf{mcrs}, t, (g, \mathbb{G}), y, \pi_1, \dots, \pi_i)$.
    (c) Compute $x_i = x_{i-1}^{r_i} \cdot \pi_i$ and $y_i = \pi_i^{r_i} \cdot y_{i-1}$.
3. Output $\pi = (\pi_1, \dots, \pi_{\log t})$.
```

- $b = \mathsf{RSWVerify}^{\mathcal{O}}(1^\lambda, \mathsf{mcrs}, t, (g, \mathbb{G}), y, \pi)$:

```
1. Let $x_0 = g$, $y_0 = y$.
2. For $i = 1, \dots, \log t$,
    (a) Let $r_i = \mathcal{O}(1^\lambda, \mathsf{mcrs}, t, (g, \mathbb{G}), y, \pi_1, \dots, \pi_i)$.
    (b) Compute $x_i = x_{i-1}^{r_i} \cdot \pi_i$ and $y_i = \pi_i^{r_i} \cdot y_{i-1}$.
3. Output 1 if and only if $x_{\log t}^2 = y_{\log t}$.
```

We note that $\mathsf{RSWProve}$ can be computed in time $t \cdot \mathrm{poly}(\lambda, \log t, n)$ and $\mathsf{RSWVerify}$ can be computed in time $\mathrm{poly}(\lambda, \log t, n)$. We note that [Pie19] shows how to compute $\mathsf{RSWProve}$ in time $t + \sqrt{t}$ when using $\sqrt{t}$ memory from computing $y = g^{2^t}$ (ignoring $\mathrm{poly}(\lambda, n)$ factors).

In the following lemmas, we show that these algorithms satisfy the following completeness and soundness properties. We let $\ell_{\mathsf{in}} = \ell_{\mathsf{in}}(\lambda, n, t)$ denote the input length of $\mathcal{O}$, where $\ell_{\mathsf{in}}(\lambda, n, t) = \lambda + n \cdot \lambda + \log t + (3 + \log t) * (\log(\lambda + 2))$.

Lemma 5.5 (Completeness)**.** *For any $\lambda, t, n \in \mathbb{N}$, $\mathbb{G}$ which is a valid representation of QR_N^+ for some N, $g \in \mathbb{G}$, and $\mathsf{mcrs} \in (\{0,1\}^\lambda)^n$, and $\mathcal{O} \in \mathsf{RF}_{\ell_{\mathsf{in}}}^\lambda$, let $y = g^{2^t}$ and $\pi = \mathsf{RSWProve}^{\mathcal{O}}(1^\lambda, \mathsf{mcrs}, t, (g, \mathbb{G}), y)$. Then, it holds that*

$$\mathsf{RSWVerify}^{\mathcal{O}}(1^\lambda, \mathsf{mcrs}, t, (g, \mathbb{G}), y, \pi) = 1.$$

Completeness follows immediately from [Pie19].

Lemma 5.6 (Soundness)**.** *For any polynomial T and unbounded algorithm Z that on input a random oracle $\mathcal{O}$ outputs polynomial-size circuits, there exists a negligible function negl such that for all $\lambda, n \in \mathbb{N}$, $i \in [n]$, it holds that*

$$\Pr\left[\begin{array}{l} \mathcal{O} \leftarrow \mathsf{RF}_{\ell_{\mathsf{in}}}^\lambda \\ \mathcal{A} = Z(\mathcal{O}) \\ \mathsf{crs}_i \leftarrow \{0,1\}^\lambda \\ (g, \mathbb{G}, y', \pi, \mathsf{mcrs}_{-i}) \\ \qquad \leftarrow \mathcal{A}^{\mathcal{O}}(1^\lambda, \mathsf{crs}_i, T(\lambda), n) \\ \mathsf{mcrs} = (\mathsf{crs}_1, \dots, \mathsf{crs}_n) \end{array} : \begin{array}{l} \mathsf{RSWVerify}^{\mathcal{O}}(1^\lambda, \mathsf{mcrs}, T(\lambda), (g, \mathbb{G}), y', \pi) = 1 \\ \wedge\ y' \neq g^{2^{T(\lambda)}} \\ \wedge\ (g, \mathbb{G}, *) \in \mathrm{Supp}\big(\mathsf{RSWGen}(1^\lambda, T(\lambda))\big) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

At a high level, the proof follows by a simple application of Lemma 3.6 of Unruh [Unr07] followed by an analysis of Pietrzak's VDF [Pie19] in the (plain) random oracle model. We give the proof in Appendix D.

Strong trapdoor VDF construction. We are now ready to state our strong trapdoor VDF construction $\mathsf{VDF} = (\mathsf{Sample}_{\mathsf{vdf}}, \mathsf{Eval}_{\mathsf{vdf}}, \mathsf{TDEval}_{\mathsf{vdf}}, \mathsf{Verify}_{\mathsf{vdf}})$ in the ABO-string model. We give $\mathsf{Eval}_{\mathsf{vdf}}$ and $\mathsf{Verify}_{\mathsf{vdf}}$ access to an oracle function $\mathcal{O}$. We note that we can efficiently check if $\mathbb{G}$ is a valid representation of QR_N^+ for some N, and we can efficiently check membership in $\mathbb{G} = \mathrm{QR}_N^+$. In particular, we say that $(g, \mathbb{G})$ are valid if $\mathbb{G}$ can be parsed as a valid representation of QR_N^+ and $g \in \mathbb{G}$.

- $(g, \mathbb{G}, |\mathbb{G}|) \leftarrow \mathsf{Sample}_{\mathsf{vdf}}(1^\lambda, t)$:
  1. Output $(g, \mathbb{G}, |\mathbb{G}|) \leftarrow \mathsf{RSWGen}(1^\lambda)$.
- $(y, \pi) \leftarrow \mathsf{Eval}^{\mathcal{O}}_{\mathsf{vdf}}(1^\lambda, \mathsf{mcrs}, t, (g, \mathbb{G}))$:
  1. If $(g, \mathbb{G})$ are invalid, output $y = \pi = \perp$.
  2. Otherwise, compute $y = g^{2^t}$ in $\mathbb{G}$ and $\pi = \mathsf{RSWProve}^{\mathcal{O}}(1^\lambda, \mathsf{mcrs}, t, (g, \mathbb{G}), y)$.
- $y = \mathsf{TDEval}_{\mathsf{vdf}}(1^\lambda, \mathsf{mcrs}, t, (g, \mathbb{G}), |\mathbb{G}|)$:
  1. If $(g, \mathbb{G})$ are invalid, output $y = \perp$.
  2. Otherwise, output $y = g^{(2^t \bmod |\mathbb{G}|)}$.
- $b = \mathsf{Verify}^{\mathcal{O}}_{\mathsf{vdf}}(1^\lambda, \mathsf{mcrs}, t, (g, \mathbb{G}), (y, \pi))$:
  1. If $(g, \mathbb{G})$ are invalid, output 1 if and only if $y = \pi = \perp$.
  2. Otherwise, output 1 if and only if $\mathsf{RSWVerify}^{\mathcal{O}}(1^\lambda, \mathsf{mcrs}, t, (g, \mathbb{G}), y, \pi) = 1$.

Theorem 5.7. *Let $B\colon \mathbb{N} \to \mathbb{N}$. Assuming the B-repeated squaring assumption for* RSWGen *holds, then there exists a B-secure strong trapdoor VDF in the ABO-string model. Soundness holds in the auxiliary-input random oracle model.*

Each of the required properties follow almost directly from the definition of a strong trapdoor VDF. We provide the full proof in Appendix D.

5.2 One-sided PV TLPs from Strong Trapdoor VDFs

Let $\mathsf{VDF} = (\mathsf{Sample}_{\mathsf{vdf}}, \mathsf{Eval}_{\mathsf{vdf}}, \mathsf{TDEval}_{\mathsf{vdf}}, \mathsf{Verify}_{\mathsf{vdf}})$ be any strong trapdoor VDF. Our construction is given by the tuple $\mathsf{TLP} = (\mathsf{Gen}_{\mathsf{tlp}}, \mathsf{Sol}_{\mathsf{tlp}}, \mathsf{Verify}_{\mathsf{tlp}})$ defined as follows.

- $z \leftarrow \mathsf{Gen}_{\mathsf{tlp}}(1^\lambda, t, s)$:
  1. Compute $(x, \mathcal{X}, \mathsf{td}) \leftarrow \mathsf{Sample}_{\mathsf{vdf}}(1^\lambda, t)$ and $y = \mathsf{TDEval}_{\mathsf{vdf}}(1^\lambda, t, (x, \mathcal{X}), \mathsf{td})$.
  2. Encode s as an element $x_s \in \mathcal{X}$ and compute $c = x_s \oplus y$.
  3. Output $z = (x, \mathcal{X}, c)$.
- $(s, \pi) \leftarrow \mathsf{Sol}_{\mathsf{tlp}}(1^\lambda, \mathsf{mcrs}, t, z)$:
  1. Parse z as $(x, \mathcal{X}, c)$. If z cannot be parsed this way, output $(\perp, \perp)$.
  2. Compute $(y, \pi_{\mathsf{vdf}}) \leftarrow \mathsf{Eval}_{\mathsf{vdf}}(1^\lambda, \mathsf{mcrs}, t, (x, \mathcal{X}))$.
  3. Let $x_s = y \oplus c$, and let s be the string encoding of x_s.
  4. Output $(s, (y, \pi_{\mathsf{vdf}}))$.

- $b = \mathsf{Verify}_{\mathsf{tlp}}(1^\lambda, \mathsf{mcrs}, t, z, (s, \pi))$:

  1. Parse z as $(x, \mathcal{X}, c)$ and π as (y, π_{vdf}). If z cannot be parsed this way, output 1 if and only if $s = \pi = \bot$.
  2. Otherwise, output 1 if and only if $\mathsf{Verify}_{\mathsf{vdf}}(1^\lambda, \mathsf{mcrs}, t, (x, \mathcal{X}), (y, \pi_{\mathsf{vdf}})) = 1$ and $c \oplus y = x_s$ where x_s is the encoding of s in $\mathcal{X}$.

In the following theorem, we show that TLP is one-sided publicly verifiable time-lock puzzle in the ABO-string model, assuming that VDF is *any* strong trapdoor VDF. However, we emphasize that for our explicit construction of VDF, soundness holds in the auxiliary-input random oracle model, so we achieve the same soundness for our explicit TLP construction.

Theorem 5.8. *Let $B \colon \mathbb{N} \to \mathbb{N}$. Suppose there exists a B-secure strong trapdoor VDF. Then, there exists a B-secure one-sided publicly verifiable time-lock puzzle.*

We provide a full proof of this thoerem in Appendix D.

5.3 Non-Malleable PV TLP from One-sided PV TLPs

By applying a similar transformation as in Section 4 to any one-sided PV TLP, we achieve a publicly verifiable time-lock puzzle (with full correctness and soundness) that is additionally non-malleable. For completeness, we restate the transformation with the syntax of *publicly verifiable* time-lock puzzles, and note that we assume the same parameters as in Section 4. Let m be a polynomial representing the output length for our function non-malleability, and let $\mathsf{TLP}_{\mathsf{os}} = (\mathsf{Gen}_{\mathsf{os}}, \mathsf{Sol}_{\mathsf{os}}, \mathsf{Verify}_{\mathsf{os}})$ be a one-sided PV TLP that uses $m(\lambda) + \lambda$ bits of randomness on security parameter λ. We construct a non-malleable PV TLP $(\mathsf{Gen}, \mathsf{Sol}, \mathsf{Verify})$ in the ABO-string model, where all algorithms have oracle access to a function $\mathcal{O} \in \mathsf{RF}^{\lambda+m}_{2\lambda+m}$.

- $z = \mathsf{Gen}^{\mathcal{O}}(1^\lambda, t, s; r)$:

  1. Get $r_{\mathsf{os}} = \mathcal{O}(s, r)$.
  2. Output $z \leftarrow \mathsf{Gen}_{\mathsf{os}}(1^\lambda, t, (s||r); r_{\mathsf{os}})$.

- $(s, \pi) \leftarrow \mathsf{Sol}^{\mathcal{O}}(1^\lambda, \mathsf{mcrs}, t, z)$:

  1. Compute $(s_{\mathsf{os}}, \pi_{\mathsf{os}}) = \mathsf{Sol}_{\mathsf{os}}(1^\lambda, \mathsf{mcrs}, t, z)$ and parse $s_{\mathsf{os}} = s||r$.
  2. If $z = \mathsf{Gen}^{\mathcal{O}}(1^\lambda, t, s; r)$, output (s, r).
  3. Otherwise, output $(\bot, (s_{\mathsf{os}}, \pi_{\mathsf{os}}))$.

- $b = \mathsf{Verify}^{\mathcal{O}}(1^\lambda, \mathsf{mcrs}, t, z, (s, \pi))$:

  1. If $s \neq \bot$, parse $\pi = r$. Output 1 if and only if $z = \mathsf{Gen}^{\mathcal{O}}(1^\lambda, t, s; r)$.
  2. If $s = \bot$, parse $\pi = (s_{\mathsf{os}}, \pi_{\mathsf{os}})$ and $s_{\mathsf{os}} = s||r$. Output 1 if and only if $z \neq \mathsf{Gen}^{\mathcal{O}}(1^\lambda, t, s; r)$ and $\mathsf{Verify}_{\mathsf{os}}(1^\lambda, \mathsf{mcrs}, t, z, (s_{\mathsf{os}}, \pi_{\mathsf{os}})) = 1$.

Recall the class of depth-bounded functions $\mathcal{F}_{B,m}$ with $m(\lambda)$-bit outputs defined in Section 4. We get the following theorem.

Theorem 5.9 (Non-malleable PV TLP from one-sided PV TLP)**.** *Let $B \colon \mathbb{N} \to \mathbb{N}$ and $m(\lambda) \in \lambda^{O(1)}$ where $B(\lambda) \in 2^{m(\lambda)} \cdot \mathrm{poly}(\lambda)$. Assuming the existence of a B-secure one-sided publicly verifiable time-lock puzzle in the ABO-string model, then there exists a B-secure publicly verifiable time-lock puzzle in the ABO-string model. Soundness holds in the auxiliary-input random oracle model. Furthermore, the construction satisfies fully concurrent functional non-malleability for the class of functions $\mathcal{F}_{B,m}$.*

Before proving the above theorem, we state the following corollary by combining Theorems 5.7, 5.8, and 5.9. We emphasize that the resulting construction is proven secure in the auxiliary-input random oracle model.

Corollary 5.10 (NM PV TLP from Repeated Squaring)**.** *Let $B \colon \mathbb{N} \to \mathbb{N}$ and $m(\lambda) \in \lambda^{O(1)}$ where $B(\lambda) \in 2^{m(\lambda)} \cdot \mathrm{poly}(\lambda)$. Assuming the B-repeated squaring assumption for RSWGen holds, then there exists a B-secure publicly verifiable time-lock puzzle in the ABO-string model. Soundness holds in the auxiliary-input random oracle model. Furthermore, the construction satisfies concurrent functional non-malleability for the class of functions $\mathcal{F}_{B,m}$.*

The proof of Theorem 5.9 follows by considering the construction $(\mathsf{Gen}, \mathsf{Sol}, \mathsf{Verify})$ assuming $(\mathsf{Gen}_{\mathsf{os}}, \mathsf{Sol}_{\mathsf{os}}, \mathsf{Verify}_{\mathsf{os}})$ is a B-secure one-sided publicly verifiable time-lock puzzle. The proofs of efficiency, hardness, and concurrent functional non-malleability follow immediately from Thorem 4.2. We provide proofs of full correctness and completeness in Appendix D. We proceed to prove that the construction satisfies the full notion of soundness in the auxiliary-input random oracle model assuming the underlying construction satisfies only one-sided soundness.

Lemma 5.11 (Soundness)**.** *Let $B \colon \mathbb{N} \to \mathbb{N}$. Assuming $(\mathsf{Gen}_{\mathsf{os}}, \mathsf{Sol}_{\mathsf{os}}, \mathsf{Verify}_{\mathsf{os}})$ is a B-secure one-sided publicly verifiable time-lock puzzle, then the construction $(\mathsf{Gen}, \mathsf{Sol}, \mathsf{Verify})$ satisfies the soundness property in the auxiliary-input random oracle model for publicly verifiable time-lock puzzles in the ABO-string model.*

Proof. Suppose by way of contradiction that $(\mathsf{Gen}, \mathsf{Sol}, \mathsf{Verify})$ does not satisfy soundness. Namely, there exists a polynomial T, unbounded algorithm Z that outputs polynomial-size circuits, a polynomial q, and integers $n \in \mathbb{N}$, $i \in [n]$ such that for infinitely many $\lambda \in \mathbb{N}$, it holds that

$$\Pr\left[\begin{array}{l} \mathcal{O} \leftarrow \mathsf{RF}^{\lambda+m}_{2\lambda+m} \\ \mathcal{A} = Z(\mathcal{O}) \\ \mathsf{crs}_i \leftarrow \{0,1\}^\lambda \\ (z, s', \pi', \mathsf{mcrs}_{-i}) \leftarrow \mathcal{A}^{\mathcal{O}}(1^\lambda, \mathsf{crs}_i, T(\lambda)) \\ \mathsf{mcrs} = (\mathsf{crs}_1, \ldots, \mathsf{crs}_n) \\ s = \mathsf{Sol}_1^{\mathcal{O}}(1^\lambda, \mathsf{mcrs}, T(\lambda), z) \end{array} : \begin{array}{l} \mathsf{Verify}^{\mathcal{O}}(1^\lambda, \mathsf{mcrs}, T(\lambda), z, (s', \pi')) = 1 \\ \wedge\ s' \neq s \end{array}\right] > 1/q(\lambda).$$

Fix any λ for which this holds. Throughout the proof, we let $t = T(\lambda)$. Additionally, it will be helpful to define $(s, \pi) = \mathsf{Sol}^{\mathcal{O}}(1^\lambda, \mathsf{mcrs}, t, z)$ to be the output of Sol for the values given by $\mathcal{A}$. Furthermore, by completeness, we know that $\mathsf{Verify}^{\mathcal{O}}(1^\lambda, \mathsf{mcrs}, t, z, (s, \pi)) = 1$.

We consider three possible events that may occur when $\mathcal{A}$ succeeds in the above experiment. Either (1) $s \neq \bot$ and $s' \neq \bot$, (2) $s = \bot$ and $s' \neq \bot$, or (3) $s \neq \bot$ and $s' = \bot$. As $s \neq s'$ when $\mathcal{A}$ succeeds, this covers all of the possible cases. We show that (1) and (2) cannot occur, and then proceed to bound the probability that (3) occurs.

For (1), suppose that it is the case that neither s nor s' are equal to $\bot$. We know that $\mathsf{Verify}^{\mathcal{O}}(1^\lambda, \mathsf{mcrs}, t, z, (s, \pi)) = 1$ and $\mathsf{Verify}^{\mathcal{O}}(1^\lambda, \mathsf{mcrs}, t, z, (s', \pi')) = 1$. This implies, by definition of Verify, that z is both in the support of $\mathsf{Gen}^{\mathcal{O}}(1^\lambda, t, s)$ and the support of $\mathsf{Gen}^{\mathcal{O}}(1^\lambda, t, s')$, but this contradicts correctness since it means that $\mathsf{Sol}_1^{\mathcal{O}}(1^\lambda, \mathsf{mcrs}, t, z)$ must be equal to both s and s'.

For (2), suppose that $s = \bot$ and $s' \neq \bot$. Becuase $\mathsf{Verify}^{\mathcal{O}}(1^\lambda, \mathsf{mcrs}, t, z, (s', \pi')) = 1$ and $s' \neq \bot$, this means that $z = \mathsf{Gen}^{\mathcal{O}}(1^\lambda, t, s'; r')$ where $r' = \pi'$. Then by correctness, it holds that $\mathsf{Sol}_1^{\mathcal{O}}(1^\lambda, \mathsf{mcrs}, t, z) = s'$, which contradicts the fact that $s' \neq s$.

As a result, we know that (3) occurs with probability at least $1/q(\lambda)$, meaning that when $\mathcal{A}$ wins, $s \neq \bot$ and $s' = \bot$. We show that this can be used to break the one-sided soundness of $(\mathsf{Gen}_{\mathsf{os}}, \mathsf{Sol}_{\mathsf{os}}, \mathsf{Verify}_{\mathsf{os}})$.

We define a non-uniform algorithm $\mathcal{B} = \{\mathcal{B}_\lambda\}_{\lambda \in \mathbb{N}}$ as follows. In order for $\mathcal{B}_\lambda$ to use $\mathcal{A}$ in the reduction, $\mathcal{B}_\lambda$ needs to simulate the oracle calls that $\mathcal{A}$ makes to $\mathcal{O}$. To do so, we need to make use of Lemma 3.6 of Unruh [Unr07]. Specifically, let $p(\lambda)$ be an upper bound on the size of the algorithm $\mathcal{A}'$ that runs $\mathcal{A}$, Sol_1, and Verify, where $\mathcal{A}$ is the adversary output by Z. Note that p is polynomially bounded by definition of Z and since T is polynomially bounded. For the polynomial function $f(\lambda) = 4 \cdot (p(\lambda))^2 \cdot (q(\lambda))^2$, we consider the inefficient algorithm Sam that on input $\mathcal{A}'$ outputs a partial assignment F of size $f(\lambda)$. By Lemma 3.6, it holds that the output distribution of $\mathcal{A}'$ with access to $\mathcal{P} \leftarrow \mathsf{RF}_{2\lambda+m}^{\lambda+m}[F]$ has statistical distance at most $\sqrt{p(\lambda)^2/f(\lambda)} = 1/(2 \cdot q(\lambda))$ from before. Thus, it holds that

$$\Pr\left[\begin{array}{l} \mathcal{O} \leftarrow \mathsf{RF}_{2\lambda+m}^{\lambda+m}; \quad \mathcal{A} = Z(\mathcal{O}) \\ F = \mathsf{Sam}(\mathcal{A}'); \quad \mathcal{P} \leftarrow \mathsf{RF}_{2\lambda+m}^{\lambda+m}[F] \\ \mathsf{crs}_i \leftarrow \{0,1\}^\lambda \\ (z, s', \pi', \mathsf{mcrs}_{-i}) \leftarrow \mathcal{A}^{\mathcal{P}}(1^\lambda, \mathsf{crs}_i, T(\lambda)) \\ \mathsf{mcrs} = (\mathsf{crs}_1, \ldots, \mathsf{crs}_n) \\ s = \mathsf{Sol}_1^{\mathcal{P}}(1^\lambda, \mathsf{mcrs}, T(\lambda), z) \end{array} : \begin{array}{l} \mathsf{Verify}^{\mathcal{P}}(1^\lambda, \mathsf{mcrs}, T(\lambda), z, (s', \pi')) = 1 \\ \wedge\ s' \neq s \end{array}\right] > \frac{1}{2 \cdot q(\lambda)}.$$

Now, for each $\lambda \in \mathbb{N}$, Z wins with at least $1/(2 \cdot q(\lambda))$ probability over a random $\mathcal{O} \leftarrow \mathsf{RF}_{2\lambda+m}^{\lambda+m}$. By a simple averaging argument, there must exist a fixed oracle $\mathcal{O}$ such that Z wins with at least $1/(2 \cdot q(\lambda))$ probability on that oracle. Let $\mathcal{A}$ be the output of Z on such an $\mathcal{O}$. We give $\mathcal{B}_\lambda$ the description of $\mathcal{A}$ hardcoded as non-uniform advice as well as the set of points F. As $|F|$ and $\mathcal{A}$ are polynomial-size, it holds that $\mathcal{B}_\lambda$ is also be polynomial-size.

We are now ready to define the behavior of $\mathcal{B}_\lambda$. On input $(1^\lambda, \mathsf{crs}_i, t)$, $\mathcal{B}_\lambda$ computes $(z, s', \pi', \mathsf{mcrs}_{-i}) \leftarrow \mathcal{A}^{\mathcal{P}}(1^\lambda, \mathsf{crs}_i, t)$. Whenever $\mathcal{A}$ queries the oracle $\mathcal{P}$, $\mathcal{B}_\lambda$ responds using F if possible, and otherwise responds with a uniformly random value (responding consistently if the same query is made multiple times). $\mathcal{B}_\lambda$ then parses π' as $(s'_{\mathsf{os}}, \pi'_{\mathsf{os}})$ and outputs $(z, s'_{\mathsf{os}}, \pi'_{\mathsf{os}}, \mathsf{crs}_i)$ if possible.

For correctness, we have already shown that when $\mathcal{A}$ wins, it must be the case that $s' = \bot$ and $s \neq \bot$. Since $s' = \bot$ and $\mathsf{Verify}^{\mathcal{P}}(1^\lambda, \mathsf{mcrs}, t, z, (s', \pi')) = 1$, it holds that $\pi' = (s'_{\mathsf{os}}, \pi'_{\mathsf{os}})$ and both (A) $\mathsf{Verify}_{\mathsf{os}}(1^\lambda, \mathsf{mcrs}, t, z, (s'_{\mathsf{os}}, \pi'_{\mathsf{os}})) = 1$ and (B) $z \neq \mathsf{Gen}^{\mathcal{P}}(1^\lambda, t, s''_{\mathsf{os}}; r''_{\mathsf{os}})$ where $s'_{\mathsf{os}} = s''_{\mathsf{os}} || r''_{\mathsf{os}}$. However, because $s \neq \bot$ and $\mathsf{Verify}^{\mathcal{P}}(1^\lambda, \mathsf{mcrs}, t, z, (s, \pi)) = 1$, it must be the case that (C) $z = \mathsf{Gen}^{\mathcal{P}}(1^\lambda, t, s; r)$ for $\pi = r$. This implies that that z is in the support of $\mathsf{Gen}_{\mathsf{os}}(1^\lambda, t, s_{\mathsf{os}})$ where $s_{\mathsf{os}} = s \| r$. By correctness of the underlying TLP, $s_{\mathsf{os}} = \mathsf{Sol}_{\mathsf{os},1}(1^\lambda, \mathsf{mcrs}, t, z)$. Finally, by (B) and (C), it holds that that $s||r \neq s''_{\mathsf{os}}||r''_{\mathsf{os}}$, so $s_{\mathsf{os}} \neq s'_{\mathsf{os}}$. Putting everything together, this implies that $\mathcal{B}_\lambda$ wins whenever $\mathcal{A}$ wins as $s'_{\mathsf{os}} \neq s_{\mathsf{os}}$, $\mathsf{Verify}_{\mathsf{os}}(1^\lambda, \mathsf{mcrs}, t, z, (s'_{\mathsf{os}}, \pi'_{\mathsf{os}})) = 1$, and $z \in \mathrm{Supp}\big(\mathsf{Gen}_{\mathsf{os}}(1^\lambda, t, \cdot)\big)$. More precisely, $\mathcal{B}_\lambda$ satisfies the following for infinitely many $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l} \mathsf{crs}_i \leftarrow \{0,1\}^\lambda \\ (z, s'_{\mathsf{os}}, \pi'_{\mathsf{os}}, \mathsf{mcrs}_{-i}) \\ \qquad \leftarrow \mathcal{B}_\lambda(1^\lambda, \mathsf{crs}_i, T(\lambda)) \\ \mathsf{mcrs} = (\mathsf{crs}_1, \ldots, \mathsf{crs}_n) \\ s_{\mathsf{os}} = \mathsf{Sol}_{\mathsf{os},1}(1^\lambda, \mathsf{mcrs}, T(\lambda), z) \end{array} : \begin{array}{l} \mathsf{Verify}_{\mathsf{os}}(1^\lambda, \mathsf{mcrs}, T(\lambda), z, (s'_{\mathsf{os}}, \pi'_{\mathsf{os}})) = 1 \\ \wedge\ s'_{\mathsf{os}} \neq s_{\mathsf{os}} \\ \wedge\ z \in \mathrm{Supp}\big(\mathsf{Gen}_{\mathsf{os}}(1^\lambda, t, \cdot)\big) \end{array}\right] > 1/(2 \cdot q(\lambda)),$$

which contradicts one-sided soundness of the underlying TLP. □

6 Applications to Multi-Party Coin Flipping and Auctions

In this section, we discuss our fair multi-party protocols. We focus on the case of multi-party coin flipping and address auctions in Remark 2 below.

Our multi-party coin flipping protocol is based on a publicly verifiable time-lock puzzle that satisfies concurrent functional non-malleability for the XOR function $f_\oplus$. Specifically, in order to produce L bits of randomness, we need concurrent function non-malleability for the function $f_\oplus \colon (\{0,1\}^L)^* \to \{0,1\}^L$ that on input $(r_1, \ldots, r_n)$ outputs $\bigoplus_{r_i \neq \bot} r_i$.

We describe our protocol in a public bulletin board model, where any party may "publish" a message that all other parties will see within some fixed time. Our protocol consists four phases: a commit phase, open phase, force open phase, and output phase. The commit and open phases consist of a single synchronous round of communication where all participating parties publish a message on the bulletin board. The force open phase can be computed by any party, and only needs to be computed by a single (honest) party. Once all puzzles have been opened (or force opened), any party can run the output phase to get the output of the protocol. When we refer to an *honest* participant, we mean a party that runs the protocol as specified, independent of all other participants.

For any $B, L \colon \mathbb{N} \to \mathbb{N}$, let $(\mathsf{Gen}, \mathsf{Sol}, \mathsf{Verify})$ be a B-secure publicly verifiable time-lock puzzle with message length $L(\lambda)$ that satisfies concurrent functional non-malleability for the function $f_\oplus$ (which has output length $L(\lambda)$). We additionally let $\alpha(\lambda)$ be the advantage of any attacker guaranteed by the hardness and functional non-malleability of the time-lock puzzle. The protocol takes as common input a security parameter λ and a time bound $t = T(\lambda)$ that satisfies the following requirements. First, we require that the commit phase takes time less than $T(\lambda)/\alpha(\lambda)$ such that hardness and functional non-malleability are preserved during the protocol. At the same time, the commit phase needs to be long enough so that all participants can generate and publish their puzzles.

- **Commit phase:** Each participant i samples $s_i \leftarrow \{0,1\}^{L(\lambda)}$ and $r_i, \mathsf{crs}_i \leftarrow \{0,1\}^\lambda$, computes $z_i \leftarrow \mathsf{Gen}(1^\lambda, t, s_i; r_i)$, and publishes z_i and crs_i. Let $\mathsf{mcrs} = (\mathsf{crs}_1, \ldots, \mathsf{crs}_n)$. We assume that only unique puzzle are published to the bulletin board.

- **Open phase:** Each participant i that published in the commit phase publishes the solution s_i and with an opening r_i.

- **Force open phase:** For each puzzle z_j, if either (a) there is no published solution s_j and opening r_j or (b) if $z_j \neq \mathsf{Gen}(1^\lambda, t, s_j; r_j)$, compute and publish $(s_j, \pi_j) \leftarrow \mathsf{Sol}(1^\lambda, \mathsf{mcrs}, t, z_j)$ (where s_j might be $\bot$).

- **Output phase:** If for every puzzle z_j and solution s_j, either (a) there is a published opening r_j such that $z_j = \mathsf{Gen}(1^\lambda, t, s_j; r_j)$ or (b) a published proof π_j such that $\mathsf{Verify}(1^\lambda, \mathsf{mcrs}, t, z_j, (s_j, \pi_j)) = 1$, then output $s = \bigoplus_{s_j \neq \bot} s_j$.

We note that the protocol above does not assume an a priori bound on the number of participants. Furthermore, there is no external setup needed by the protocol. All participants, however, do publish a random string $\mathsf{crs}_i \leftarrow \{0,1\}^\lambda$ to implement the ABO-string model for $(\mathsf{Gen}, \mathsf{Sol}, \mathsf{Verify})$.

Theorem 6.1. *Let $B \colon \mathbb{N} \to \mathbb{N}$, $L(\lambda) \in \lambda^{O(1)}$ where $B(\lambda) \in 2^{L(\lambda)} \cdot \mathrm{poly}(\lambda)$ Assume the existence of a B-secure publicly verifiable time-lock puzzle for $L(\lambda)$ bit messages in the ABO-string model that satisfies concurrent function non-malleability for $f_\oplus$ with $L(\lambda)$ bit output. Then, there exists a multi-party coin flipping protocol that satisfies optimistic efficiency, fairness, and public verifiability. The protocol supports an unbounded number of participants and requires no adversary-independent trusted setup.*

We note the following result by plugging Corollary 5.10 into Theorem 6.1.

Corollary 6.2. *Let $B, L\colon \mathbb{N} \to \mathbb{N}$ where $B(\lambda) \in 2^{L(\lambda)} \cdot \mathrm{poly}(\lambda)$. Assuming the B-repeated squaring assumption for* RSWGen*, there exists a multi-party coin flipping protocol that satisfies optimistic efficiency, fairness, and public verifiability. The protocol supports an unbounded number of participants and requires no adversary-indpendent trusted setup. Security is proven in the auxiliary-input random oracle.*

We remark how we can adapt our protocol to deal with auctions.

Remark 2 (Multi-Party Auctions)**.** *For our application to auctions, we consider a standard second-price, sealed-bid auction, in which the auctioned item is assigned to the highest bidder who pays the second highest bid for the item. We assume some form of authenticated channels so we can know the bidders' identities in order to distribute the auctioned items. We leave these as external implementation details for the protocol. The main protocol proceeds as follows.*

In the commit phase, each participant computes a time-lock puzzle to their bid. The open and force open phases are identical to the case of coin flipping. Then in the output phase, we need to determine the identity of the highest bidder and the value of the second highest bid.

The function that computes the output consists of finding the top two values in a set. This can be computed in low depth (doing a tree of comparisons in parallel) and has output length $\log n + \log M$ *where n is the number of participants and M is a bound on the largest valid bid. Thus, using our publicly verifiable time-lock puzzle that satisfies concurrent functional non-malleability for this function, the resulting protocol is secure assuming $n \cdot M \cdot \mathrm{poly}(\lambda)$ security for the repeated squaring assumption. Assuming n and M are polynomially bounded, we only need polynomial security assumptions.*

In the remainder of this section, we discuss the various properties satisfied by our construction of Theorem 6.1.

Complexity. We discuss the efficiency of each phase in the above protocol. As mentioned above, the commit and open phase consist of a single round of synchronous communication. For the commit phase, each participating party requires at most $\mathrm{poly}(\lambda, \log t)$ local computation time by the efficiency of Gen, and we require that all messages be posted within a specified time at most $t/\alpha(\lambda)$. After this specified time, all participating parties can publish their solutions and openings as part of the open phase, and if needed, unopened puzzles can be force opened. By the efficiency of Sol, force open requires time $t \cdot \mathrm{poly}(\lambda, \log t)$ per unopened puzzle and can be computed by a single party. The output phase can be computed by anyone and requires at most $n \cdot \mathrm{poly}(\lambda, \log t)$ time by the efficiency of Gen and Verify, where n is the number of puzzles submitted during the commit phase.

Optimistic efficiency. If all parties that publish a puzzle in the commit phase also publish a valid solution and opening in the open phase, then the output phase can be run immediately without running the force open phase.

Public verifiability. Let z_j be any puzzle which was not opened, or was opened incorrectly during the open phase. Suppose an honest party runs the force open phase for puzzle z_j and publishes $(s_j, \pi_j) \leftarrow \mathsf{Sol}(1^\lambda, \mathsf{mcrs}, t, z_j)$. By completeness of $(\mathsf{Gen}, \mathsf{Sol}, \mathsf{Verify})$, it follows that $\mathsf{Verify}(1^\lambda, \mathsf{mcrs}, t, z_j, (s_j, \pi_j)) = 1$ for any $z_j \in \{0,1\}^*$, so that check in the output phase will pass. Thus, if a single honest party runs the entire force open phase, then any party can run the output phase and all checks will pass.

Fairness. We formalize and prove fairness in the following lemma. At a high level, we show that as long as there is a single honest participant (who only needs to publish an honestly sampled puzzle independent of all other participants), the output of the protocol is statistically close to a uniformly random distribution over $L(\lambda)$ bits.

Lemma 6.3 (Fairness)**.** *For any distinguisher $\mathcal{D}$, there exists a negligible function* negl *such that for all $\lambda \in \mathbb{N}$, the following holds. Suppose that at most $n(\lambda) \in \mathrm{poly}(\lambda)$ participants for the commit phase, and at least one honest party runs the commit phase. Let s be the output of the open phase at the end of the protocol for security parameter λ, and $r \leftarrow \{0,1\}^{L(\lambda)}$. It holds that*

$$|\Pr[\mathcal{D}(s) = 1] - \Pr[D(r) = 1]| \leq \mathsf{negl}(\lambda).$$

Proof. Suppose by way of contradiction that there exists a polynomial q such that for infinitely many $\lambda \in \mathbb{N}$,

$$|\Pr[\mathcal{D}(s) = 1] - \Pr[\mathcal{D}(r) = 1]| > 1/q(\lambda).$$

Fix any $\lambda \in \mathbb{N}$ for which the above holds, and let $n = n(\lambda)$, $L = L(\lambda)$, and $t = T(\lambda)$. Let $z_1, \ldots, z_n$ be the (unique) puzzles published in the commit phase and let $\mathsf{mcrs} = (\mathsf{crs}_1, \ldots, \mathsf{crs}_n)$. Without loss of generality, suppose that participant "1" is honest, so z_1 and crs_1 are generated honestly and published to the bulletin board. More specifically, $s_1 \leftarrow \{0,1\}^L$, $r_1, \mathsf{crs}_1 \leftarrow \{0,1\}^\lambda$, $z_1 \leftarrow \mathsf{Gen}(1^\lambda, t, s_1; r_1)$, and only z_1 and crs_1 are published before the open phase begins. Note that since z_1 is generated honestly and independently of all other parties, it will be unique with all but negligible probability, so will be successfully published to the bulletin board.

Let $\mathcal{A} = \{\mathcal{A}_\lambda\}_{\lambda \in \mathbb{N}}$ be a non-uniform algorithm where $\mathcal{A}_\lambda$ consists of the actions of all parties in the protocol other than participant 1 that outputs puzzles $z_2, \ldots, z_n$ during the commit phase. By definition of the protocol and α, $\mathcal{A}_\lambda$ has polynomial size and depth bounded by $T(\lambda)/\alpha(\lambda)$. Thus, $\mathcal{A}$ is a valid $(1, n, B, \alpha, T)$-MIM adversary, and we can consider the corresponding distribution $\mathsf{mim}_{\mathcal{A}}(1^\lambda, t, v)$ for any $v \in \{0,1\}^L$.

Let $\mathcal{B} = \{\mathcal{B}_\lambda\}_{\lambda \in \mathbb{N}}$ be a non-uniform algorithm where $\mathcal{B}_\lambda$ receives as input $(1^\lambda, \mathsf{crs}_1, t)$, where crs_1 is the random string output by the honest party, and then simulates the actions of all parties in the protocol and eventually computes the output s. In particular, we assume that $\mathcal{B}_\lambda$ publishes all of the relevant values on the public bulletin board.

We consider the following hybrid distributions:

$$\begin{aligned}
\mathsf{Hyb}_0(\lambda) &= s,\\
\mathsf{Hyb}_1(\lambda) &= \mathcal{B}_\lambda(1^\lambda, \mathsf{crs}_1, t),\\
\mathsf{Hyb}_2(\lambda) &= f_\oplus(s_1, \mathsf{mim}_{\mathcal{A}}(1^\lambda, t, s_1)),\\
\mathsf{Hyb}_3(\lambda) &= f_\oplus(s_1, \mathsf{mim}_{\mathcal{A}}(1^\lambda, t, 0^L)),\\
\mathsf{Hyb}_4(\lambda) &= r.
\end{aligned}$$

By assumption, we have that

$$|\Pr[\mathcal{D}(\mathsf{Hyb}_0(\lambda)) = 1] - \Pr[\mathcal{D}(\mathsf{Hyb}_4(\lambda)) = 1]| > 1/q(\lambda).$$

Towards a contradiction, we will show that for each $i \in \{1,2,3,4\}$ that $|\Pr\left[\mathcal{D}(\mathsf{Hyb}_{i-1}(\lambda)) = 1\right] - \Pr[\mathcal{D}(\mathsf{Hyb}_i(\lambda)) = 1]| \leq 1/(4 \cdot q(\lambda))$.

- $\underline{|\Pr[\mathcal{D}(\mathsf{Hyb}_0(\lambda)) = 1] - \Pr[\mathcal{D}(\mathsf{Hyb}_1(\lambda)) = 1]| \leq 1/(4 \cdot q(\lambda))}$

  By definition of $\mathcal{B}$, $\mathsf{Hyb}_0(\lambda)$ and $\mathsf{Hyb}_1(\lambda)$ are identically distributed, so it holds that

$$|\Pr[\mathcal{D}(\mathsf{Hyb}_0(\lambda)) = 1] - \Pr[\mathcal{D}(\mathsf{Hyb}_1(\lambda)) = 1]| = 0.$$

- $|\Pr[\mathcal{D}(\mathsf{Hyb}_1(\lambda)) = 1] - \Pr[\mathcal{D}(\mathsf{Hyb}_2(\lambda)) = 1]| \le 1/(4 \cdot q(\lambda))$

Assume by way of contradiction that $|\Pr[\mathcal{D}(\mathsf{Hyb}_1(\lambda)) = 1] - \Pr[\mathcal{D}(\mathsf{Hyb}_2(\lambda)) = 1]| > 1/(4 \cdot q(\lambda))$. Under this assumption, we show how to break soundness of $(\mathsf{Gen}, \mathsf{Sol}, \mathsf{Verify})$.

Consider a non-uniform algorithm $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$ defined as follows. $\mathcal{C}_\lambda$ on input $(1^\lambda, \mathsf{crs}_1, t)$ simulates $\mathcal{B}_\lambda(1^\lambda, \mathsf{crs}_1, t)$. For $i \in [2, n]$, let crs_i be the random string posted, and s'_i be the solution published on the bulletin board for puzzle z_i with valid opening r'_i or proof π'_1. If z_i is equal to a puzzle z_j for $j < i$, set $s_i = \bot$, and otherwise let $(s_i, \pi_i) = \mathsf{Sol}(1^\lambda, \mathsf{mcrs}, t, z_i)$. If there exists an $i \in [n]$ such that $s_i \ne s'_i$, $\mathcal{C}_\lambda$ outputs $(z_i, s'_i, \pi'_i, \mathsf{crs}_{-1})$, and outputs $\bot$ otherwise. Since there are a polynomial n number of participants, each of which is polynomial time, $\mathcal{B}_\lambda$ and hence $\mathcal{C}_\lambda$ runs in polynomial time. We proceed to argue that $\mathcal{C}_\lambda$ breaks soundness with $1/(4 \cdot q(\lambda))$ probability.

Recall that s_i is the solution given by Sol. By full correctness of $(\mathsf{Gen}, \mathsf{Sol}, \mathsf{Verify})$, for every $i \in [n]$, s_i is either the unique value such that $z_i \in \operatorname{Supp}\big(\mathsf{Gen}(1^\lambda, t, s_i)\big)$ or s_i is $\bot$. By definition of $\mathsf{mim}_\mathcal{A}$, it follows that $(s_2, \ldots, s_n)$ is the output of $\mathsf{mim}_\mathcal{A}(1^\lambda, t, s_1)$ (as we can assume without loss of generality that duplicate puzzles are ignored by $\mathcal{B}_\lambda$). Thus, in the event where $s_i = s'_i$ for all $i \in [n]$, it follows that $\mathsf{Hyb}_1(\lambda) = \mathsf{Hyb}_2(\lambda)$ However, we assumed that with probability $\mathsf{Hyb}_1(\lambda) \ne \mathsf{Hyb}_2(\lambda)$ with at least $1/(4 \cdot q(\lambda))$ probability, so it follows that there exists an $i \in [n]$ such that $s_i \ne s'_i$ with the same probability.

Whenever $s_i \ne s'_i$, we claim that $\mathcal{C}_\lambda$ breaks soundness. We have that $\mathcal{B}_\lambda$ outputs $f_\oplus(s'_1, \ldots, s'_n)$. We know that the checks in the output phase pass for s'_i, so either (a) $z_i = \mathsf{Gen}(1^\lambda, t, s'_i; r'_i)$ or (b) $\mathsf{Verify}(1^\lambda, \mathsf{mcrs}, t, z_i, (s'_i, \pi'_i)) = 1$. We have that (a) is impossible since if z_i is in the support of $\mathsf{Gen}(1^\lambda, t, s'_i)$, this means that $s_i = s'_i$ by correctness. Thus, it must be the case that (b) holds with at least $1/(4 \cdot q(\lambda))$ probability. This implies that for infinitely many $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l} \mathsf{crs}_1 \leftarrow \{0,1\}^\lambda \\ (z, s'_i, \pi'_i, \mathsf{crs}_{-1}) \leftarrow \mathcal{C}_\lambda(1^\lambda, \mathsf{crs}_1, t) \\ \mathsf{mcrs} = (\mathsf{crs}_1, \ldots, \mathsf{crs}_n) \\ s_i = \mathsf{Sol}_1(1^\lambda, \mathsf{mcrs}, t, z_i) \end{array} : \begin{array}{l} \mathsf{Verify}(1^\lambda, \mathsf{mcrs}, t, z_i, (s'_i, \pi'_i)) = 1 \\ \wedge\ s_i \ne s'_i \end{array}\right] > 1/(4 \cdot q(\lambda)),$$

which contradicts soundness.

- $|\Pr[\mathcal{D}(\mathsf{Hyb}_2(\lambda)) = 1] - \Pr[\mathcal{D}(\mathsf{Hyb}_3(\lambda)) = 1]| \le 1/(4 \cdot q(\lambda))$

Assume by way of contradiction that $|\Pr[\mathcal{D}(\mathsf{Hyb}_2(\lambda)) = 1] - \Pr[\mathcal{D}(\mathsf{Hyb}_3(\lambda)) = 1]| > 1/(4 \cdot q(\lambda))$. Under this assumption, we show how to break concurrent functional non-malleability of $(\mathsf{Gen}, \mathsf{Sol}, \mathsf{Verify})$ for the function $f_\oplus$.

We already mentioned that $\mathcal{A}$ is a valid $(1, n, B, \alpha, T)$-MIM adversary. Thus, it suffices to come up with a distinguisher $\mathcal{D}_{\mathsf{nm}}$ for non-malleability that can distinguish $f_\oplus(\mathsf{mim}_\mathcal{A}(1^\lambda, t, v))$ for $v = s_1$ or $v = 0^L$.

$\mathcal{D}_{\mathsf{nm}}$ on input $f_\oplus(\mathsf{mim}_\mathcal{A}(1^\lambda, t, v))$ has s_1 hardcoded and simply outputs $\mathcal{D}(s_1 \oplus f_\oplus(\mathsf{mim}_\mathcal{A}(1^\lambda, t, v)))$. By definition of $f_\oplus$, it holds that

$$s_1 \oplus f_\oplus(\mathsf{mim}_\mathcal{A}(1^\lambda, t, v)) = f_\oplus(s_1, \mathsf{mim}_\mathcal{A}(1^\lambda, t, v)).$$

Thus, $\mathcal{D}_{\mathsf{nm}}$ distinguishes $v = s_1$ and $v = 0^L$ with the same probability as $\mathcal{D}$, so it holds that

$$|\Pr\Big[D_{\mathsf{nm}}(f_\oplus(\mathsf{mim}_\mathcal{A}(1^\lambda, t, s_1))) = 1\Big] - \Pr\Big[D_{\mathsf{nm}}(f_\oplus(\mathsf{mim}_\mathcal{A}(1^\lambda, t, 0^L))) = 1\Big]| > 1/(4 \cdot q(\lambda)),$$

in contradiction.

- $\underline{|\Pr[\mathcal{D}(\mathsf{Hyb}_3(\lambda)) = 1] - \Pr[\mathcal{D}(\mathsf{Hyb}_4(\lambda)) = 1]| \leq 1/(4 \cdot q(\lambda))}$

  By assumption that participant 1 is honest, it holds that $\mathsf{mim}_{\mathcal{A}}(1^\lambda, t, 0^L)$ is independent of s_1. Since $s_1 \leftarrow \{0,1\}^\lambda$, this implies that $f_\oplus(s_1, \mathsf{mim}_{\mathcal{A}}(1^\lambda, t, 0^L)$ is uniformly random. Thus, it is identically distributed to $r \leftarrow \{0,1\}^\lambda$. It follows that

$$|\Pr[\mathcal{D}(\mathsf{Hyb}_3(\lambda)) = 1] - \Pr[\mathcal{D}(\mathsf{Hyb}_4(\lambda)) = 1]| = 0.$$

This completes the proof of the lemma.

□

Acknowledgements. This work was supported in part by NSF Award SATC-1704788, NSF Award RI-1703846, NSF Award DGE-1650441, AFOSR Award FA9550-18-1-0267, and a JP Morgan Faculty Award. This research is based upon work supported in part by the Office of the Director of National Intelligence (ODNI), Intelligence Advanced Research Projects Activity (IARPA), via 2019-19-020700006. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of ODNI, IARPA, or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright annotation therein.

References

[Bar02] Boaz Barak. Constant-round coin-tossing with a man in the middle or realizing the shared random string model. In *43rd Symposium on Foundations of Computer Science FOCS*, pages 345–355, 2002.

[BBBF18] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In *Advances in Cryptology - CRYPTO*, pages 757–788, 2018.

[BBF18] Dan Boneh, Benedikt Bünz, and Ben Fisch. A survey of two verifiable delay functions. *IACR Cryptol. ePrint Arch.*, 2018:712, 2018.

[BDD$^+$20a] Carsten Baum, Bernardo David, Rafael Dowsley, Jesper Buus Nielsen, and Sabine Oechsner. Craft: Composable randomness and almost fairness from time. Cryptology ePrint Archive, Report 2020/784, 2020. https://eprint.iacr.org/2020/784.

[BDD$^+$20b] Carsten Baum, Bernardo David, Rafael Dowsley, Jesper Buus Nielsen, and Sabine Oechsner. Tardis: Time and relative delays in simulation. Cryptology ePrint Archive, Report 2020/537, 2020. https://eprint.iacr.org/2020/537.

[BGJ$^+$16] Nir Bitansky, Shafi Goldwasser, Abhishek Jain, Omer Paneth, Vinod Vaikuntanathan, and Brent Waters. Time-lock puzzles from randomized encodings. In *ITCS*, 2016.

[BN00] Dan Boneh and Moni Naor. Timed commitments. In *Advances in Cryptology - CRYPTO*, pages 236–254, 2000.

[BPS06] Boaz Barak, Manoj Prabhakaran, and Amit Sahai. Concurrent non-malleable zero knowledge. In *47th Annual IEEE Symposium on Foundations of Computer Science (FOCS*, pages 345–354, 2006.

[BPS16] Iddo Bentov, Rafael Pass, and Elaine Shi. Snow white: Provably secure proofs of stake. *IACR Cryptology ePrint Archive*, 2016:919, 2016.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM Conference on Computer and Communications Security*, pages 62–73. ACM, 1993.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–145. IEEE Computer Society, 2001.

[CDGS18] Sandro Coretti, Yevgeniy Dodis, Siyao Guo, and John P. Steinberger. Random oracles and non-uniformity. In *Advances in Cryptology - EUROCRYPT*, pages 227–258, 2018.

[Cle86] Richard Cleve. Limits on the security of coin flips when half the processors are faulty (extended abstract). In *Proceedings of the 18th Annual ACM Symposium on Theory of Computing, STOC*, pages 364–369, 1986.

[COSV16] Michele Ciampi, Rafail Ostrovsky, Luisa Siniscalchi, and Ivan Visconti. Concurrent non-malleable commitments (and more) in 3 rounds. In *Advances in Cryptology - CRYPTO*, pages 270–299, 2016.

[COSV17] Michele Ciampi, Rafail Ostrovsky, Luisa Siniscalchi, and Ivan Visconti. Four-round concurrent non-malleable commitments from one-way functions. In *Advances in Cryptology - CRYPTO*, pages 127–157, 2017.

[DDN91] Danny Dolev, Cynthia Dwork, and Moni Naor. Non-malleable cryptography (extended abstract). In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, STOC*, pages 542–552, 1991.

[DGKR18] Bernardo David, Peter Gazi, Aggelos Kiayias, and Alexander Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In *Advances in Cryptology - EUROCRYPT*, pages 66–98, 2018.

[DKP20] Dana Dachman-Soled, Ilan Komargodski, and Rafael Pass. Non-malleable codes for bounded polynomial depth tampering, 2020.

[DPS19] Phil Daian, Rafael Pass, and Elaine Shi. Snow white: Robustly reconfigurable consensus and applications to provably secure proof of stake. In *Financial Cryptography and Data Security - 23rd International Conference, FC*, pages 23–41, 2019.

[EFKP19] Naomi Ephraim, Cody Freitag, Ilan Komargodski, and Rafael Pass. Continuous verifiable delay functions. *IACR Cryptology ePrint Archive*, 2019:619, 2019.

[FKL18] Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology - CRYPTO 2018*, pages 33–62. Springer, 2018.

[FMPS19] Luca De Feo, Simon Masson, Christophe Petit, and Antonio Sanso. Verifiable delay functions from supersingular isogenies and pairings. In *Advances in Cryptology - ASIACRYPT*, pages 248–277, 2019.

[FO13] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. *J. Cryptology*, 26(1):80–101, 2013.

[GLOV12] Vipul Goyal, Chen-Kuei Lee, Rafail Ostrovsky, and Ivan Visconti. Constructing non-malleable commitments: A black-box approach. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS*, pages 51–60, 2012.

[GO14] Jens Groth and Rafail Ostrovsky. Cryptography in the multi-string model. *J. Cryptology*, 27(3):506–543, 2014.

[Goy11] Vipul Goyal. Constant round non-malleable protocols using one way functions. In Lance Fortnow and Salil P. Vadhan, editors, *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC*, pages 695–704, 2011.

[GPR16] Vipul Goyal, Omkant Pandey, and Silas Richelson. Textbook non-malleable commitments. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC*, pages 1128–1141, 2016.

[HMPS15] Susan Hohenberger, Steven Myers, Rafael Pass, and Abhi Shelat. An overview of ANONIZE: A large-scale anonymous survey system. *IEEE Security & Privacy*, 13(2):22–29, 2015.

[Khu17] Dakshita Khurana. Round optimal concurrent non-malleability from polynomial hardness. In *Theory of Cryptography - 15th International Conference, TCC*, pages 139–171, 2017.

[KK19] Yael Tauman Kalai and Dakshita Khurana. Non-interactive non-malleability from quantum supremacy. In *Advances in Cryptology - CRYPTO*, pages 552–582, 2019.

[KLX20] Jonathan Katz, Julian Loss, and Jiayu Xu. On the security of time-locked puzzles and timed commitments. *IACR Cryptol. ePrint Arch.*, 2020:730, 2020.

[KS17] Dakshita Khurana and Amit Sahai. How to achieve non-malleability in one or two rounds. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS*, pages 564–575, 2017.

[LP09] Huijia Lin and Rafael Pass. Non-malleability amplification. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC*, pages 189–198, 2009.

[LP11] Huijia Lin and Rafael Pass. Constant-round non-malleable commitments from any one-way function. In *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC*, pages 705–714, 2011.

[LPS17] Huijia Lin, Rafael Pass, and Pratik Soni. Two-round and non-interactive concurrent non-malleable commitments from time-lock puzzles. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS*, pages 576–587. IEEE Computer Society, 2017.

[LPTV10] Huijia Lin, Rafael Pass, Wei-Lung Dustin Tseng, and Muthuramakrishnan Venkitasubramaniam. Concurrent non-malleable zero knowledge proofs. In *Advances in Cryptology - CRYPTO*, 2010.

[LPV08] Huijia Lin, Rafael Pass, and Muthuramakrishnan Venkitasubramaniam. Concurrent non-malleable commitments from any one-way function. In *Theory of Cryptography, Fifth Theory of Cryptography Conference, TCC*, pages 571–588, 2008.

[MT19] Giulio Malavolta and Sri Aravinda Krishnan Thyagarajan. Homomorphic time-lock puzzles and applications. In *Advances in Cryptology - CRYPTO*, volume 11692 of *Lecture Notes in Computer Science*, pages 620–649. Springer, 2019.

[OPV10] Rafail Ostrovsky, Omkant Pandey, and Ivan Visconti. Efficiency preserving transformations for concurrent non-malleable zero knowledge. In *Theory of Cryptography, 7th Theory of Cryptography Conference, TCC*, pages 535–552, 2010.

[Pie19] Krzysztof Pietrzak. Simple verifiable delay functions. In *10th Innovations in Theoretical Computer Science Conference, ITCS*, pages 60:1–60:15, 2019.

[PPV08] Omkant Pandey, Rafael Pass, and Vinod Vaikuntanathan. Adaptive one-way functions and applications. In *Advances in Cryptology - CRYPTO*, pages 57–74, 2008.

[PR05a] Rafael Pass and Alon Rosen. Concurrent non-malleable commitments. In *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS*, pages 563–572, 2005.

[PR05b] Rafael Pass and Alon Rosen. New and improved constructions of non-malleable cryptographic protocols. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, STOC*, pages 533–542, 2005.

[PR08] Rafael Pass and Alon Rosen. Concurrent nonmalleable commitments. *SIAM J. Comput.*, 37(6):1891–1925, 2008.

[PW10] Rafael Pass and Hoeteck Wee. Constant-round non-malleable commitments from subexponential one-way functions. In *Advances in Cryptology - EUROCRYPT*, 2010.

[RS20] Lior Rotem and Gil Segev. Generically speeding-up repeated squaring is equivalent to factoring: Sharp thresholds for all generic-ring delay functions. *IACR Cryptol. ePrint Arch.*, 2020:812, 2020. To appear in CRYPTO 2020.

[RSW96] R. L. Rivest, A. Shamir, and D. A. Wagner. Time-lock puzzles and timed-release crypto, 1996. Technical report, Massachusetts Institute of Technology, Cambridge, MA, USA.

[Unr07] Dominique Unruh. Random oracles and auxiliary input. In *Advances in Cryptology - CRYPTO 2007, 27th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2007, Proceedings*, pages 205–223, 2007.

[VZGS13] Joachim Von Zur Gathen and Igor E Shparlinski. Generating safe primes. *Journal of Mathematical Cryptology*, 7(4):333–365, 2013.

[Wee10] Hoeteck Wee. Black-box, round-efficient secure computation via non-malleability amplification. In *FOCS*, 2010.

[Wes19] Benjamin Wesolowski. Efficient verifiable delay functions. In *Advances in Cryptology - EUROCRYPT*, pages 379–407, 2019.

A Non-malleability Against Depth-Bounded Distinguishers

In this section we consider the notion of non-malleability with a depth-bounded MIM attacker as well as a *depth-bounded distinguisher*. This section was added after posting the initial version and was motivated by the (concurrent and independent) works of [BDD+20b, KLX20, BDD+20a] who studied similar strenghtenings of plain timed primitives to ours yet their definitions are different. The definition of [BDD+20b, BDD+20a] is a UC-style definition and the definition of [KLX20] is a CCA-style one, where in both the attacker/distinguisher/environment are depth-bounded and cannot brute-force solve any puzzle. Here, we show that such a modification for our non-malleability definition gives a strictly weaker security guarantees which, in particular, may be insufficient for some applications.

We first define the notion of non-malleability where the distinguisher runs in bounded depth.

Definition A.1 (Depth-Bounded Distinguisher Non-malleability). *Let $n_L, n_R, B \colon \mathbb{N} \to \mathbb{N}$. A B-secure time-lock puzzle* $(\mathsf{Gen}, \mathsf{Sol})$ *is* (n_L, n_R)-concurrent non-malleable against depth-bounded distinguishers *if there exists a positive polynomial α such that for every function T with $\alpha(\lambda) \leq T(\lambda) \in$* $\mathrm{poly}(\lambda, B(\lambda))$ *for all $\lambda \in \mathbb{N}$ and every (n_L, n_R, B, α, T)-MIM adversary $\mathcal{A} = \{\mathcal{A}_\lambda\}_{\lambda\in\mathbb{N}}$, the following holds.*

For any non-uniform distinguisher $\mathcal{D} = \{\mathcal{D}_\lambda\}_{\lambda\in\mathbb{N}}$ satisfying $\mathsf{depth}(\mathcal{D}_\lambda) \leq T(\lambda)/\alpha(\lambda)$ *and* $\mathsf{size}(\mathcal{D}_\lambda) \in \mathrm{poly}(\lambda, B(\lambda))$ *for all $\lambda \in \mathbb{N}$, there exists a negligible function* negl *such that for all $\lambda \in \mathbb{N}$ and $\vec{s} = (s_1, \ldots, s_{n_L(\lambda)}) \in (\{0,1\}^\lambda)^{n_L(\lambda)}$,*

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

We note that the definition of non-malleability given in Definition 3.4 corresponds to when the distinguisher $\mathcal{D}$ is unbounded. We can also consider an in between notion of non-malleability where the distinguisher $\mathcal{D}_\lambda$ may run in polynomial time that depends on T, e.g. $\mathrm{poly}(\lambda, T(\lambda))$. When T is restricted to a polynomial, this corresponds to a more standard notion of computational non-malleability. As in Definition 3.5, we can similarly define the relevant extensions of non-malleability specifying the number of left and right puzzles in the MIM experiment.

A.1 Equivalence to Functional Non-malleability with One Bit Output

Recall that, at a high level, we defined the class of functions $\mathcal{F}_{B,m}$ as functions of the form $f \colon (\{0,1\}^\lambda)^* \to \{0,1\}^m$ computable in size $\mathrm{poly}(\lambda, B(\lambda))$ and "low depth" which depends only polynomially on λ and poly-logarithmically on the number of inputs it receives. We could have instead explicitly allowed the class of functions $\mathcal{F}_{B,m}$ to be low depth in a way that depends on the time bound T. Specifically, for B, T, α from the definition of non-malleability, we could consider the class of functions $\mathcal{D}_{B,\alpha,T,m}$ of the form $f \colon (\{0,1\}^\lambda)^* \to \{0,1\}^m$ computable in size $\mathrm{poly}(\lambda, B(\lambda))$ and depth $T(\lambda)/(\alpha(\lambda))$. When the output length is 1, functional non-malleability with respect to this class $\mathcal{D}_{B,\alpha,T,1}$ is definitionally equivalent to depth-bounded distinguisher non-malleability.

We briefly argue why the two notions are equivalent. We focus on the case of plain (non-concurrent) non-malleability, but the intuition extends to general concurrent setting as well. To see this, note that the functional non-malleability intuitively says that no depth-bounded MIM attacker $\mathcal{A}$ can statistically influence the distribution of $f(\mathsf{mim}_{\mathcal{A}}(s))$ for any $f \in \mathcal{D}_{B,\alpha,T,1}$ and $s \in \{0,1\}^\lambda$. More specifically, for any $s \in \{0,1\}^\lambda$,

$$f(\mathsf{mim}_{\mathcal{A}}(s)) \approx f(\mathsf{mim}_{\mathcal{A}}(0^\lambda)).$$

On the other hand, depth-bounded distinguisher non-malleability says that no depth-bounded distinguisher $\mathcal{D}$ can distinguish $\mathsf{mim}_{\mathcal{A}}(s)$ from $\mathsf{mim}_{\mathcal{A}}(0^\lambda)$, meaning that the output is statistically close. Because both f and $\mathcal{D}$ have output length 1, the definitions are equivalent.

As a consequence, it suffices to show that our construction nmTLP_1 given in Section 4 actually satisfies functional non-malleability for $\mathcal{D}_{B,\alpha,T,1}$ in addition to $\mathcal{F}_{B,1}$. The proof only requires slight modification where we use the fact that the function f is computable in low depth. Therefore, combining this with the above, nmTLP_1 actually satisfies concurrent non-malleability against depth-bounded distinguishers and only relies on polynomial security.

Lemma A.2. *Let $B\colon \mathbb{N} \to \mathbb{N}$, where $B(\lambda) \in \mathrm{poly}(\lambda)$. Assuming the existence of a B-secure time-lock puzzle, then there exists a B-secure time-lock puzzle that is concurrent non-malleable against depth-bounded distinguishers. Security is proven in the auxiliary-input random oracle model.*

A.2 Separating Depth-Bounded Distinguishers from Unbounded Ones

We next give our construction separating non-malleability with a depth-bounded distinguisher from non-malleability with a non-uniform $\mathrm{poly}(\lambda, T(\lambda))$-size distinguisher.

We first give a high-level overview of the separation. Our construction $\mathsf{TLP}^\star$ will be for messages of length $m(\lambda)$ and will rely on an underlying non-malleable time-lock puzzle $\mathsf{TLP}_{\mathsf{short}}$ for shorter messages whose output length is $m(\lambda)$. The main idea is that $\mathsf{TLP}^\star$ will split up a message $s \in \{0,1\}^m$ into two parts s_L and s_R and generate puzzles z_L and z_R for each of them. This leads to the following very natural MIM attack with an unbounded distinguisher. Let $\mathcal{A}$ be a MIM attacker that on input $z = z_L \| z_R$ simply outputs a puzzle $\tilde{z}$ for $\mathsf{TLP}^\star$ with solution z_L, which has the right length m by assumption. An unbounded distinguisher receives as input z_L, can solve the puzzle, and check if the solution corresponds to s_L. It remains to show how to generate the puzzles z_L and z_R such that $\mathsf{TLP}^\star$ *does* satisfy non-malleability against depth-bounded distinguishers.

If the solutions underlying the $\mathsf{TLP}_{\mathsf{short}}$ puzzles z_L and z_R are simply s_L and s_R, respectively, this will clearly not satisfy non-malleability as it allows for a "mix-and-match" style attack. Specifically, on input $z = z_L \| z_R$, a MIM attacker can output $\tilde{z} = z_R \| z_L$ or $\tilde{z} = z_L \| z^\star$ where $z^\star$ is any valid $\mathsf{TLP}_{\mathsf{short}}$ puzzle (other than z_R). The underlying solution in these attacks are clearly related to $s = s_L \| s_R$ in a way that can be easily checked in bounded depth. To prevent such attacks, we make two key modifications. First, we add a bit at the beginning of each solution indicating whether that part of the puzzle is intended to correspond to the left or right half of the solution s. This prevents the attack that "swaps" z_L and z_R. Second, we append a random string $r \leftarrow \{0,1\}^\lambda$ to the solutions and require that both parts for any valid puzzle for $\mathsf{TLP}^\star$ end in the same string. This prevents the attacker from replacing one of the puzzles with a new value, possibly unrelated to s. If it could do so, it would intuitively need to know what the underlying value for r is. With these two modifications, we can prove that no other MIM attacks succeed with a bounded distinguisher (assuming $\mathsf{TLP}_{\mathsf{short}}$ is concurrent non-malleable against depth-bounded distinguishers). We next formalize our construction as follows.

Let $m\colon \mathbb{N} \to \mathbb{N}$ be a function with even output, and $\mathsf{TLP}_{\mathsf{short}} = (\mathsf{Gen}_{\mathsf{short}}, \mathsf{Sol}_{\mathsf{short}})$ be a $(1,2)$-concurrent non-malleable TLP for $1 + m(\lambda)/2 + \lambda$ bit messages with output length $m(\lambda)$. We will construct a time-lock puzzle $\mathsf{TLP}^\star = (\mathsf{Gen}^\star, \mathsf{Sol}^\star)$ for $m(\lambda)$ bit messages. For simplicity, we write $m = m(\lambda)$ when the context is clear. For a string $s \in \{0,1\}^m$, we use $s[a : b]$ to denote the substring of length $b - a$ starting at the ath character in s.

- $\mathsf{Gen}^\star(1^\lambda, t, s)$:

    1. Sample $r \leftarrow \{0,1\}^\lambda$.

   2. Compute $z_L \leftarrow \mathsf{Gen}_{\text{short}}(1^\lambda, t, 0\|s[0:m/2]\|r)$.
   3. Compute $z_R \leftarrow \mathsf{Gen}_{\text{short}}(1^\lambda, t, 1\|s[m/2+1:m]\|r)$.
   4. Output $z = z_L\|z_R$.

- $\mathsf{Sol}^\star(1^\lambda, t, z)$**:**
   1. Parse $z = z_L\|z_R$.
   2. Compute $b_L\|s_L\|r_L = \mathsf{Sol}_{\text{short}}(1^\lambda, t, z_L)$.
   3. Compute $b_R\|s_R\|r_R = \mathsf{Sol}_{\text{short}}(1^\lambda, t, z_R)$.
   4. If $b_L = 0$, $b_R = 1$, and $r_L = r_R$, output $s_L\|s_R$.
   5. Otherwise, output $\perp$.

Theorem A.3. *Let $B, m\colon \mathbb{N} \to \mathbb{N}$ with $B(\lambda) \leq 2^\lambda$ and where $m(\lambda)$ is at least $4\lambda + 2$ and is even for all $\lambda \in \mathbb{N}$. Assuming the existence of a B-secure time-lock puzzle that is $(1,2)$-concurrent non-malleable against depth-bounded distinguishers for messages of length $1 + m(\lambda)/2 + \lambda$ and output length $m(\lambda)$, there exists a B-secure time-lock puzzle for messages of length $m(\lambda)$ that satisfies non-malleability against depth-bounded distinguisher but does not satisfy non-malleability against distinguishers with size and depth $t \cdot \text{poly}(\lambda, \log t)$.*

We note that, in order for $\mathsf{TLP}_{\text{short}}$ to provide at least λ bits of security, it must be the case that its output length is at least λ bits longer than its input length. This implies that $m \geq 1+m/2+2\lambda$, or $m \geq 4\lambda+2$. Additionally, we note that the, by Lemma A.2, our construction nmTLP_1 of Section 4 when instantiated for messages of length $1+m/2+\lambda$ gives a candidate for $\mathsf{TLP}_{\text{short}}$ in the auxiliary-input random oracle model assuming any polynomially-secure TLP.

The proof of the theorem follows by considering the construction $\mathsf{TLP}^\star$ based on $\mathsf{TLP}_{\text{short}}$. Correctness, efficiency, and B-hardness of $\mathsf{TLP}^\star$ are straightforward to show from the corresponding properties of $\mathsf{TLP}_{\text{short}}$, so we focus on the relevant notions of non-malleability below.

Lemma A.4. *Assuming $\mathsf{TLP}_{\text{short}}$ is a correct time-lock puzzle, $\mathsf{TLP}^\star$ is not $(1,1)$-concurrent non-malleable against distinguishers with size and depth $t \cdot \text{poly}(\lambda, \log t)$.*

Proof. Let $\beta(\lambda)$ be a bound on the running time $\mathsf{Gen}^\star(1^\lambda, T(\lambda), \cdot)$ for any $T(\lambda) \in \text{poly}(\lambda, B(\lambda))$, which is a polynomial by efficiency of $\mathsf{Gen}^\star$ and since $B(\lambda) \leq 2^\lambda$ by assumption. Let α be any positive polynomial, and consider the function $T(\lambda) = \alpha(\lambda) \cdot \beta(\lambda)$. We construct a MIM adversary $\mathcal{A} = \{\mathcal{A}_\lambda\}_{\lambda\in\mathbb{N}}$ that on input $z = z_L\|z_R \leftarrow \mathsf{Gen}^\star(1^\lambda, T(\lambda), s)$ outputs $\tilde{z} \leftarrow \mathsf{Gen}^\star(1^\lambda, t, z_L)$. By efficiency of $\mathsf{Gen}^\star$, $\mathcal{A}_\lambda$ runs in time $\beta(\lambda) = T(\lambda)/\alpha(\lambda)$ so is a valid $(1, 1, B, \alpha, T)$-MIM adversary.

Let $\mathcal{D}$ be the distinguishing algorithm that on input $\tilde{s}$ computes $\tilde{b}_L\|\tilde{s}_L\|\tilde{r}_L = \mathsf{Sol}_{\text{short}}(1^\lambda, T(\lambda), \tilde{s})$ and outputs $\tilde{s}_L[0]$. Since $\tilde{z}$ is in the support of $\mathsf{Gen}^\star(1^\lambda, t, z_L)$, it follows that $\tilde{s} = z_L$ as long as z_L is not equal to z (which is the case with high probability over the randomness of $\mathsf{Gen}^\star$ used to generate $z = z_L\|z_R$). By correctness of $\mathsf{TLP}_{\text{short}}$, it holds that $\tilde{s}_L = s[0:m/2]$, so $\mathcal{D}$ outputs $s[0]$ with high probability. This violates $(1,1)$-concurrent non-malleability by considering the string $s = 1^m$, which differs from 0^m in the first bit. Furthermore, $\mathcal{D}$ runs in time $t \cdot \text{poly}(\lambda, \log t)$ by efficiency of $\mathsf{Sol}_{\text{short}}$. □

Lemma A.5. *Assuming that $\mathsf{TLP}_{\text{short}}$ is a B-secure $(1,2)$-concurrent non-malleable time-lock puzzle against depth-bounded distinguishers, then $\mathsf{TLP}^\star$ is $(1,1)$-concurrent non-malleable against depth-bounded distinguishers.*

Proof. Suppose that $\mathsf{TLP}^\star$ is not non-malleable against depth-bounded distinguishers. Then, for any positive polynomial α, there is a function T with $\alpha(\lambda) \le T(\lambda) \in \mathrm{poly}(\lambda, B(\lambda))$ for all $\lambda \in \mathbb{N}$, a $(1, 1, B, \alpha, T)$-MIM adversary $\mathcal{A} = \{\mathcal{A}_\lambda\}_{\lambda\in\mathbb{N}}$, a distinguisher $\mathcal{D} = \{\mathcal{D}_\lambda\}_{\lambda\in\mathbb{N}}$ satisfying $\mathsf{depth}(\mathcal{D}_\lambda) \le T(\lambda)/(\alpha(\lambda))$ and $\mathsf{size}(\mathcal{D}_\lambda) \in \mathrm{poly}(\lambda, B(\lambda))$, a polynomial q, and a string $s \in \{0,1\}^m$ such that for infinitely many $\lambda \in \mathbb{N}$ it holds that

$$\left|\Pr\left[\mathcal{D}_\lambda(\mathsf{mim}_{\mathcal{A}}(1^\lambda, T(\lambda), s)) = 1\right] - \Pr\left[\mathcal{D}_\lambda(\mathsf{mim}_{\mathcal{A}}(1^\lambda, T(\lambda), 0^m)) = 1\right]\right| > 1/q(\lambda).$$

We show, by a hybrid argument, that this implies we can break $(2,2)$-concurrent non-malleability of $\mathsf{TLP}_{\text{short}}$ against a depth-bounded distinguisher. As $(1,2)$-concurrent non-malleability implies $(2,2)$-concurrent non-malleability (see Lemma C.1), this suffices to reach a contradiction. Since the above holds for any α, we will show it for the case where $\alpha(\lambda) = \alpha_{\text{short}}(\lambda) \cdot (1 + \beta_1(\lambda) + \beta_2(\lambda))$, where α_{short} is the polynomial specified for the $(2,2)$-concurrent non-malleability of $\mathsf{TLP}_{\text{short}}$ and β_1, β_2 are fixed polynomials, greater than 1, specified in the proofs of Claim A.6 and Claim A.7, respectively.

Throughout the proof, let $t = T(\lambda)$. For any $s \in \{0,1\}^m$ and each $\lambda \in \mathbb{N}$, we define the following hybrid experiments. In order to specify the MIM distribution, for any puzzle z under $\mathsf{TLP}^\star$, we let $\mathsf{UniqueSol}^\star(1^\lambda, t, z)$ denote the unique solution s underlying z if one exists, and $\perp$ otherwise. We similarly denote by $\mathsf{UniqueSol}_{\text{short}}$ the corresponding values when z is a puzzle using $\mathsf{TLP}_{\text{short}}$.

- $\mathsf{Hyb}_0^s(\lambda)$**:** The first hybrid is identical to $\mathsf{mim}_{\mathcal{A}}(1^\lambda, t, s)$ with $\mathsf{Gen}^\star$ written out explicitly.

$$\mathsf{Hyb}_0^s(\lambda) = \left\{\begin{array}{l} r \leftarrow \{0,1\}^\lambda \\ z_L \leftarrow \mathsf{Gen}_{\text{short}}(1^\lambda, t, 0\|s[0:m/2]\|r) \\ z_R \leftarrow \mathsf{Gen}_{\text{short}}(1^\lambda, t, 1\|s[m/2+1:m]\|r) \\ z = z_L\|z_R \\ \tilde{z} \leftarrow \mathcal{A}_\lambda(z) \\ \tilde{s} = \begin{cases} \perp \text{ if } \tilde{z} = z \\ \mathsf{UniqueSol}^\star(1^\lambda, t, \tilde{z}) \text{ otherwise} \end{cases} \end{array} : \tilde{s}\right\}$$

- $\mathsf{Hyb}_1^s(\lambda)(\lambda)$**:** The next hybrid sets $\tilde{s} = \perp$ if either $\tilde{z}_L$ or $\tilde{z}_R$ are copied, instead of only if both are copied.

$$\mathsf{Hyb}_1^s(\lambda) = \left\{\begin{array}{l} r \leftarrow \{0,1\}^\lambda \\ z_L \leftarrow \mathsf{Gen}_{\text{short}}(1^\lambda, t, 0\|s[0:m/2]\|r) \\ z_R \leftarrow \mathsf{Gen}_{\text{short}}(1^\lambda, t, 1\|s[m/2+1:m]\|r) \\ z = z_L\|z_R \\ \tilde{z} = \tilde{z}_L\|\tilde{z}_R \leftarrow \mathcal{A}_\lambda(z) \\ \tilde{s} = \begin{cases} \perp \text{ if } \tilde{z}_L \text{ or } \tilde{z}_R \text{ equal } z_L \text{ or } z_R \\ \mathsf{UniqueSol}^\star(1^\lambda, t, \tilde{z}) \text{ otherwise} \end{cases} \end{array} : \tilde{s}\right\}$$

- $\mathsf{Hyb}_2^s(\lambda)$**:** The next hybrid computes z_L and z_R using $0^{m/2}$ in place of $s[0:m/2]$ and $s[m/2+1:m]$.

$$\mathsf{Hyb}_2^s(\lambda) = \left\{\begin{array}{l} r \leftarrow \{0,1\}^\lambda \\ z_L \leftarrow \mathsf{Gen}_{\text{short}}(1^\lambda, t, 0\|0^{m/2}\|r) \\ z_R \leftarrow \mathsf{Gen}_{\text{short}}(1^\lambda, t, 1\|0^{m/2}\|r) \\ z = z_L\|z_R \\ \tilde{z} \leftarrow \mathcal{A}_\lambda(z) \\ \tilde{s} = \begin{cases} \perp \text{ if } \tilde{z}_L \text{ or } \tilde{z}_R \text{ equal } z_L \text{ or } z_R \\ \mathsf{UniqueSol}^\star(1^\lambda, t, \tilde{z}) \text{ otherwise} \end{cases} \end{array} : \tilde{s}\right\}$$

- $\mathsf{Hyb}_3^s(\lambda)$**:** The final hybrid set $\tilde{s}$ to $\perp$ only if $\tilde{z}$ is invalid or $\tilde{z} = z$ (equivalently, $\tilde{z}_L = z_L$ and $\tilde{z}_R = z_R$). Note that this hybrid is identical to $\mathsf{mim}_{\mathcal{A}}(1^\lambda, t, 0^m)$.

$$\mathsf{Hyb}_3^s(\lambda) = \left\{ \begin{array}{ll} r \leftarrow \{0,1\}^\lambda & \\ z_L \leftarrow \mathsf{Gen}_{\text{short}}(1^\lambda, t, 0\|0^{m/2}\|r) & \\ z_R \leftarrow \mathsf{Gen}_{\text{short}}(1^\lambda, t, 1\|0^{m/2}\|r) & \\ z = z_L\|z_R & : \tilde{s} \\ \tilde{z} \leftarrow \mathcal{A}_\lambda(z) & \\ \tilde{s} = \begin{cases} \perp \text{ if } \tilde{z} = z \\ \mathsf{UniqueSol}^\star(1^\lambda, t, \tilde{z}) \text{ otherwise} \end{cases} & \end{array} \right\}$$

It follows that, for infinitely many $\lambda \in \mathbb{N}$, $\mathcal{D}_\lambda$ distinguishes at least one pair of consecutive hybrids with at least $1/(3 \cdot q(\lambda))$ probability. We show that, if this is true for any pair of consecutive hybrids, $\mathcal{D}_\lambda$ can be used to violate $(2, 2)$-concurrent non-malleability of $\mathsf{TLP}_{\text{short}}$ with a depth-bounded distinguisher. At a high level, we show in the first claim that if $\mathcal{D}_\lambda$ distinguishes either $\mathsf{Hyb}_0^s(\lambda)$ from $\mathsf{Hyb}_1^s(\lambda)$ or $\mathsf{Hyb}_2^s(\lambda)$ from $\mathsf{Hyb}_3^s(\lambda)$, then it can be used to generate a new puzzle that depends on the randomness r. In the next claim, we make use of the fact that $\mathsf{Hyb}_1^s(\lambda)$ and $\mathsf{Hyb}_2^s(\lambda)$ now set $\tilde{s}$ to be $\perp$ as the MIM distribution would for $(2, 2)$-concurrent non-malleability of $\mathsf{TLP}_{\text{short}}$.

Claim A.6. *Let $s \in \{0,1\}^m$ and for any $\lambda \in \mathbb{N}$, define $\rho(\lambda)$ as*

$$\rho(\lambda) = \left|\Pr\left[\mathcal{D}_\lambda(\mathsf{Hyb}_0^s(\lambda)) = 1\right] - \Pr\left[\mathcal{D}_\lambda(\mathsf{Hyb}_1^s(\lambda)) = 1\right]\right|.$$

Then, there exists a reduction that violates $(2, 2)$-concurrent non-malleability against depth-bounded distinguishers for $\mathsf{TLP}_{\text{short}}$ with probability at least $\rho(\lambda) - 2^{-\lambda}$.

Proof. Fix any value of s. Throughout the proof, we denote by s_L the first $m/2$ bits of s, and by s_R the second $m/2$ bits.

We start by noting that $\mathsf{Hyb}_0^s(\lambda)$ sets $\tilde{s} = \perp$ if and only if either $\tilde{z}$ is an invalid puzzle, or $\tilde{z}_L = z_L$ and $\tilde{z}_R = z_R$. The difference between this and $\mathsf{Hyb}_1^s(\lambda)$ is that in the latter, the value of $\tilde{s}$ might additionally be set to $\perp$ if neither of the above cases occur (so $\tilde{z}$ is valid and $(\tilde{z}_L\|\tilde{z}_R) \neq (z_L\|z_R)$), yet $\tilde{z}_L$ or $\tilde{z}_R$ are equal to either z_L or z_R.

We have the following observations about this event. First, because $\tilde{z}$ is valid, then $\tilde{z}_L$ has the solution $0\|\tilde{s}_L\|\tilde{r}$ and $\tilde{z}_R$ has the solution $1\|\tilde{s}_R\|\tilde{r}$ for some values of $\tilde{s}_L, \tilde{s}_R, \tilde{r}$. Moreover, since one of the puzzles is copied, then $\tilde{r} = r$. Therefore, whenever the output of the hybrids differs, then the following event, denoted E, occurs:

> At least one of $\tilde{z}_L, \tilde{z}_R$ is a valid $\mathsf{TLP}_{\text{short}}$ puzzle that's not copied from z_L or z_R, and has solution $\tilde{b}\|\tilde{s}\|r$ for some $\tilde{b} \in \{0,1\}$ and $\tilde{s} \in \{0,1\}^{m/2}$.

It follows that E occurs with probability at least $\rho(\lambda)$ when $r, z_L, z_R, \tilde{z}_L, \tilde{z}_R$ are sampled as in the hybrid distributions. Formally, we have that

$$\Pr\left[\begin{array}{ll} r \leftarrow \{0,1\}^\lambda & \\ z_L \leftarrow \mathsf{Gen}_{\text{short}}(1^\lambda, t, 0\|s_L\|r) & \\ z_R \leftarrow \mathsf{Gen}_{\text{short}}(1^\lambda, t, 1\|s_R\|r) & : \mathsf{E} \\ \tilde{z}_L\|\tilde{z}_R \leftarrow \mathcal{A}_\lambda(z_L\|z_R) & \end{array} \right] \geq \rho(\lambda).$$

Looking ahead, we want to use this to break the $(2, 2)$-concurrent non-malleability of $\mathsf{TLP}_{\text{short}}$, where we receive puzzles (z_L, z_R) either corresponding to solutions $(0\|s_L\|r, 1\|s_R\|r)$ or $(0^{m/2+\lambda+1},$

$0^{m/2+\lambda+1}$). To do so, it will be helpful to consider the probability that E occurs when z_L and z_R are generated as puzzles for $0^{m/2+\lambda+1}$. Because r is chosen uniformly from $\{0,1\}^\lambda$ and z_L and z_R are independent of r, it follows that E occurs with at most $2^{-\lambda}$ probability over r. Namely,

$$\Pr\left[\begin{array}{l} r \leftarrow \{0,1\}^\lambda \\ z_L \leftarrow \mathsf{Gen}_{\text{short}}(1^\lambda, t, 0^{m/2+\lambda+1}) \\ z_R \leftarrow \mathsf{Gen}_{\text{short}}(1^\lambda, t, 0^{m/2+\lambda+1}) \\ \tilde{z}_L \| \tilde{z}_R \leftarrow \mathcal{A}_\lambda(z_L \| z_R) \end{array} : \mathsf{E}\right] \leq 2^{-\lambda}.$$

It follows that there exists a fixed value of r, denoted $r^\star$, such that the difference in the above two probabilities relative to $r^\star$ is at least $\rho(\lambda) - 2^{-\lambda}$. We will use $r^\star$ to give an adversary and distinguisher against the non-malleability of $\mathsf{TLP}_{\text{short}}$.

Consider the MIM adversary $\mathcal{A}_{\text{short}}$ against the $(2,2)$-non-malleability of $\mathsf{TLP}_{\text{short}}$ that on input (z_L, z_R), computes $\tilde{z}_L \| \tilde{z}_R \leftarrow \mathcal{A}_\lambda(z_L \| z_R)$ and outputs $(\tilde{z}_L, \tilde{z}_R)$. Let $\mathcal{D}_{\text{short}}$ be the distinguisher that has $r^\star$ hardcoded and receives two solutions as input. If at least one of them can be parsed as $\tilde{b} \| \tilde{s} \| r^\star$ for some $\tilde{b}$, $\tilde{s}$, then $\mathcal{D}_{\text{short}}$ outputs 1. Otherwise, $\mathcal{D}_{\text{short}}$ outputs a random bit.

We observe that $\mathcal{D}_{\text{short}}$ outputs 1 (rather than a random bit) if and only if E occurs, which follows by definition of E and $\mathcal{D}_{\text{short}}$. It therefore follows that

$$\begin{aligned} &\Big|\Pr\Big[\mathcal{D}_{\text{short}}(\mathsf{mim}_{\mathcal{A}_{\text{short}}}(1^\lambda, t, (0\|s_1\|r, 1\|s_2\|r)) = 1\Big] \\ &- \Pr\Big[\mathcal{D}_{\text{short}}(\mathsf{mim}_{\mathcal{A}_{\text{short}}}(1^\lambda, t, (0^{m/2+\lambda+1}, 0^{m/2+\lambda+1})) = 1\Big]\Big| \geq \rho(\lambda) - 2^{-\lambda}. \end{aligned}$$

To complete the reduction, we discuss the efficiency of $\mathcal{A}_{\text{short}}$ and $\mathcal{D}_{\text{short}}$ and the parameters used in the reduction. For $\mathcal{A}_{\text{short}}$, it only runs $\mathcal{A}_\lambda$ while formatting the inputs and outputs appropriately. For $\mathcal{D}_{\text{short}}$, it simply checks if its input can be parsed correctly based on $r^\star$, which takes time polynomial in its input length $m + 2\lambda + 1$, which is a fixed polynomial in λ. Let β_1 be a polynomial in λ upper bounding the overheads for each algorithm, so both algorithms have size $\operatorname{poly}(\lambda, B(\lambda))$ and depth at most $T(\lambda)/\alpha(\lambda) + \beta_1(\lambda)$. Recall that we set $\alpha(\lambda) \geq \alpha_{\text{short}}(1 + \beta_1(\lambda))$ where α_{short} is the polynomial given by the $(2,2)$-non-malleability of $\mathsf{TLP}_{\text{short}}$ against depth-bounded distinguishers. We can therefore bound $\mathsf{depth}(\mathcal{A}_{\text{short}})$ and $\mathsf{depth}(\mathcal{D}_{\text{short}})$ by

$$\frac{T(\lambda)}{\alpha(\lambda)} + \beta_1(\lambda) = \frac{T(\lambda) + \alpha(\lambda)\cdot\beta_1(\lambda)}{\alpha(\lambda)} \leq \frac{T(\lambda) + T(\lambda)\cdot\beta_1(\lambda)}{\alpha(\lambda)} \leq \frac{T(\lambda) + T(\lambda)\cdot\beta_1(\lambda)}{\alpha_{\text{short}}(\lambda)\cdot(1+\beta_1(\lambda))} = \frac{T(\lambda)}{\alpha_{\text{short}}(\lambda)},$$

where we used the fact that $\alpha(\lambda) \leq T(\lambda)$. Therefore, $\mathcal{A}_{\text{short}}$ is a valid $(2, 2, B, \alpha_{\text{short}}, T)$-MIM adversary and $\mathcal{D}_{\text{short}}$ has bounded depth, as required. Finally, we note that that since $\alpha(\lambda)$ is greater than $\alpha_{\text{short}}(\lambda)$, it follows that T satisfies $\alpha_{\text{short}}(\lambda) \leq T(\lambda) \in \operatorname{poly}(\lambda, B(\lambda))$, which completes the proof. ☐

Claim A.7. *Let $s \in \{0,1\}^m$ and for any $\lambda \in \mathbb{N}$, define $\rho(\lambda)$ as*

$$\rho(\lambda) = \left|\Pr\left[\mathcal{D}_\lambda(\mathsf{Hyb}_1^s(\lambda)) = 1\right] - \Pr\left[\mathcal{D}_\lambda(\mathsf{Hyb}_2^s(\lambda)) = 1\right]\right|.$$

Then, there exists a reduction that violates $(2,2)$-concurrent non-malleability against depth-bounded distinguishers for $\mathsf{TLP}_{\text{short}}$ with probability at least $\rho(\lambda)$.

Proof. Let T, $\mathcal{D}$, $\mathcal{A}$, and s be as above. As

$$\rho(\lambda) = \left|\Pr\left[\mathcal{D}_\lambda(\mathsf{Hyb}_1^s(\lambda)) = 1\right] - \Pr\left[\mathcal{D}_\lambda(\mathsf{Hyb}_2^s(\lambda)) = 1\right]\right|,$$

it follows by an averaging argument there exists a fixed choice of r in the experiment such that the distinguishing probability is at least $\rho(\lambda)$ with respect to r. We refer to the hybrid experiments above in this setting as $\mathsf{Hyb}_1^{\text{fixed}}(\lambda)$ and $\mathsf{Hyb}_2^{\text{fixed}}(\lambda)$.

We construct an adversary $\mathcal{A}_{\text{short}}$ and a distinguisher $\mathcal{D}_{\text{short}}$ that violates $(2,2)$-concurrent non-malleability of $\mathsf{TLP}_{\text{short}}$ for s_L and s_R equal to either (1) $0\|s[0:m/2]\|r$ and $0\|s[m/2+1:m]\|r$ or (2) $0\|0^{m/2}\|r$ and $0\|0^{m/2}\|r$, corresponding to hybrids 1 and 2, as follows. $\mathcal{A}_{\text{short}}$ on input $z_L \leftarrow \mathsf{Gen}_{\text{short}}(1^\lambda, t, s_L)$ and $z_L \leftarrow \mathsf{Gen}_{\text{short}}(1^\lambda, t, s_R)$ runs $\mathcal{A}_\lambda(z_L\|z_R)$ to obtain $\tilde{z} = \tilde{z}_L\|\tilde{z}_R$ and outputs the puzzles $\tilde{z}_L$ and $\tilde{z}_R$. The distinguisher $\mathcal{D}_{\text{short}}$ receives as input the underlying values $\tilde{b}_L\|\tilde{s}_L\|\tilde{r}_L$ and $\tilde{b}_R\|\tilde{s}_R\|\tilde{r}_R$ corresponding to the puzzles $\tilde{z}_L$ and $\tilde{z}_R$, respectively (unless either puzzle is invalid or copied, in which case it receives $\perp$ in place of the solution for that puzzle, which we deal with separately). If it holds that $\tilde{b}_L = 0$, $\tilde{b}_R = 1$, and $\tilde{r}_L = \tilde{r}_R$, $\mathcal{D}_{\text{short}}$ outputs $\mathcal{D}_\lambda(\tilde{s}_L\|\tilde{s}_R)$. Otherwise (or if either input is $\perp$), $\mathcal{D}_{\text{short}}$ outputs $\mathcal{D}_\lambda(\perp)$.

To prove the claim, we show that in case (1), the output of $\mathcal{D}_{\text{short}}$, when given as input the unique solution underlying the output of $\mathcal{A}_{\text{short}}(z)$, is distributed identically to $\mathcal{D}_\lambda(\mathsf{Hyb}_1^{\text{fixed}}(\lambda))$. The proof that the output of $\mathcal{D}_{\text{short}}$ is distributed according to $\mathcal{D}_\lambda(\mathsf{Hyb}_1^{\text{fixed}}(\lambda))$ in case (2) follows similarly. We note that the input to $\mathcal{A}_\lambda$ in $\mathsf{Hyb}_1^{\text{fixed}}(\lambda)$ is identically distributed to its input when run by $\mathcal{A}$, since both z_L and z_R are generated identically. Therefore, the output $\tilde{z}$ of $\mathcal{A}_\lambda(z\|z_r^\star)$ is identically distributed in both cases.

We next argue that the input to $\mathcal{D}_\lambda$ is identically distributed in each case. In particular, we want to show that $\mathsf{Hyb}_1^{\text{fixed}}$ sets the input to $\perp$ if and only if $\mathsf{mim}_{\mathcal{A}_{\text{short}}}$ or $\mathcal{D}_{\text{short}}$ cause the input of $\mathcal{D}_\lambda$ to be $\perp$. In the forward direction, $\mathsf{Hyb}_1^{\text{fixed}}$ sets the input to $\perp$ if (a) $\tilde{z}_L$ or $\tilde{z}_R$ are copied or (b) $\tilde{z}$ is an invalid puzzle, meaning either $\tilde{z}_L$ or $\tilde{z}_R$ are invalid or $\tilde{b}_L \neq 0$, $\tilde{b}_R \neq 1$, or $\tilde{r}_L \neq \tilde{r}_R$. In case (a) where either are copied, $\mathcal{D}_{\text{short}}$ receives $\perp$ for the corresponding value and outputs $\mathcal{D}_\lambda(\perp)$. In case (b), $\mathcal{D}_{\text{short}}$ directly checks if $\tilde{z}$ is invalid by checking if $\tilde{b}_L = 1$, $\tilde{b}_R = 1$, and $\tilde{r}_L = \tilde{r}_R$ if both $\tilde{z}_L$ and $\tilde{z}_R$ are valid. For the reverse direction $\mathcal{D}_{\text{short}}(\lambda)$ outputs $\mathcal{D}_\lambda(\perp)$ if either input it receives is $\perp$ or if the checks it makes don't pass. $\mathcal{D}_{\text{short}}$ receives an input of $\perp$ if either $\tilde{z}_L$ or $\tilde{z}_R$ are invalid or copied, in which case $\mathsf{Hyb}_1^{\text{fixed}}$ sets $\tilde{s}$ to $\perp$. Similarly, the checks $\mathcal{D}_{\text{short}}$ make exactly correspond to checking validity of $\tilde{z}$, in which case $\mathsf{Hyb}_1^{\text{fixed}}$ also sets $\tilde{s}$ to $\perp$. Thus, the output of $\mathcal{D}_{\text{short}}$ is identically distributed to the output of $\mathcal{D}_\lambda(\mathsf{Hyb}_1^{\text{fixed}}(\lambda))$.

Finally, we remark on the efficiency of $\mathcal{A}_{\text{short}}$ and $\mathcal{D}_{\text{short}}$ and the corresponding parameters used in the full reduction. $\mathcal{A}_{\text{short}}$ simply runs $\mathcal{A}_\lambda$ while formatting the inputs and outputs appropriately. $\mathcal{D}_{\text{short}}$ runs $\mathcal{D}_\lambda$ after making the necessary equality checks. Let β_2 be a polynomial in λ upper bounding the overhead for each algorithm, so both algorithms have size $\text{poly}(\lambda, B(\lambda))$ and depth at most $T(\lambda)/\alpha(\lambda) + \beta_2(\lambda)$, where we recall that $\alpha(\lambda) \geq \alpha_{\text{short}}(\lambda)(1+\beta_2(\lambda))$. It follows that $\mathsf{depth}(\mathcal{A}_{\text{short}})$ and $\mathsf{depth}(\mathcal{D}_{\text{short}})$ are bounded by

$$\frac{T(\lambda)}{\alpha(\lambda)} + \beta_2(\lambda) = \frac{T(\lambda) + \alpha(\lambda)\cdot\beta_2(\lambda)}{\alpha(\lambda)} \leq \frac{T(\lambda) + T(\lambda)\cdot\beta_2(\lambda)}{\alpha(\lambda)} \leq \frac{T(\lambda) + T(\lambda)\cdot\beta_2(\lambda)}{\alpha_{\text{short}}(\lambda)\cdot(1+\beta_2(\lambda))} = \frac{T(\lambda)}{\alpha_{\text{short}}(\lambda)},$$

where we used the fact that $\alpha(\lambda) \leq T(\lambda)$. This means that $\mathcal{A}_{\text{short}}$ is a valid $(2,2,B,\alpha,T)$-MIM adversary and $\mathcal{D}_{\text{short}}$ satisfies the necessary properties of depth-bounded distinguisher non-malleability. Finally, we note that that since $\alpha(\lambda)$ is greater than $\alpha_{\text{short}}(\lambda)$, it follows that T satisfies $\alpha_{\text{short}}(\lambda) \leq T(\lambda) \in \text{poly}(\lambda, B(\lambda))$, which completes the proof. □

This completes the proof of the Lemma. □

B Discussion of Non-Malleable Definitions

We briefly discuss the different notions of non-malleability studied in this work. Specifically, we compare standard non-malleability (Definition 3.4), non-malleability against depth-bounded distinguishers (Definition A.1), and functional non-malleability (Definition 4.1). In section 1.3, we also discuss the definitions considered in the concurrent works of [KLX20, BDD$^+$20a, BDD$^+$20b].

Common to all of our definitions, there is a depth-bounded man-in-the-middle (MIM) attacker, which we call $\mathcal{A}$, that on input a puzzle z with solution s tries to output a different puzzle $\tilde{z}$ to a related value $\tilde{s}$. Here, $\mathcal{A}$ is depth-bounded relative to the difficulty of the puzzle, so it should not be able to solve the puzzle. The definitions vary in what it means for $\tilde{s}$ to be "related" to s. For our standard notion of non-malleability, we require that no unbounded distinguisher $\mathcal{D}$ on input $\tilde{s}$ can tell if it came from the experiment starting with s or the all-zero string. In the definition of non-malleability against depth-bounded distinguishers, $\mathcal{D}$ is restricted to be depth-bounded in the same way as $\mathcal{A}$. In the case of functional non-malleability, the (unbounded) distinguisher $\mathcal{D}$ receives instead as input $f(\tilde{s})$ where f is a low-depth function. We parameterize functional non-malleability by an output length m. When $m = |s|$, this captures plain non-malleability by considering f to be the identity function. When $m = 1$, this captures depth-bounded distinguisher non-malleability as f essentially plays the role of the depth-bounded distinguisher $\mathcal{D}$. In Theorem 4.2, we show how to construct a time-lock puzzle satisfying functional non-malleability for any output length m assuming a time-lock puzzle that is $2^m \cdot \mathrm{poly}(\lambda)$ secure.

When considering concurrent non-malleability, the MIM attacker $\mathcal{A}$ receives possibly multiple puzzles $z_1, \ldots, z_{n_L}$ that have solutions $s_1, \ldots, s_{n_L}$ as input and tries to output multiple puzzles $\tilde{z}_1, \ldots, \tilde{z}_{n_R}$ (different from its inputs) corresponding to $\tilde{s}_1, \ldots, \tilde{s}_{n_R}$. In the most general form, we can consider some distinguisher $\mathcal{D}$ that receives as input $f(\tilde{s}_1, \ldots, \tilde{s}_{n_R})$ and tries to tell if it came from the experiment starting with $s_1, \ldots, s_{n_L}$ or with n_L all-zero strings. We show in Theorem 4.17 that if the MIM attacker can encode a time-lock puzzle into the value $f(\tilde{s}_1, \ldots, \tilde{s}_{n_R})$ (where f may be the identity), then the construction cannot be secure against an unbounded distinguisher. In particular, if the function's output length m is greater than the output length of the time-lock puzzle, the scheme may not be secure. On the other hand, our construction of Theorem 4.2 works for functional non-malleability even in the fully concurrent setting, as the output length of f is bounded. So, as long as the output length of the function f is sufficiently small, we can support unbounded concurrency.

Finally, our separation in Appendix A.2 gives a construction that satisfies plain (non-concurrent) non-malleability against depth-bounded distinguishers yet does not satisfy non-malleability against unbounded distinguishers. We remark that in the setting where the message length for the puzzle is 1 bit, these notions are equivalent by simply considering the depth-bounded distinguisher that outputs the bit it gets as input. Moreover, it can be shown that they are equivalent as long as the message length is in $O(\log \lambda)$. Therefore, this separation necessarily relies on the fact that the message length for the puzzle is in $\omega(\log \lambda)$.

We summarize the various relationships between our definitions in Figure 1. Specifically, an arrow from definition A to definition B indicates that any construction satisfying A also satisfies B. We let m be the message length of the TLP construction and n be the bound on concurrency. Unless otherwise specified, the arrows hold for all concurrency bounds n. First, the implications going from left to right for the top and bottoms rows hold directly since they only restrict the power of the distinguisher. Next, we note that functional NM for all functions f is equivalent to NM with an unbounded distinguisher $\mathcal{D}$. From FNM to NM, you can let f be the identity (as long as the output length $\ell \geq m \cdot n$), and in the other direction, you can construct a new distinguisher $\mathcal{D}'$ that simply runs $(\mathcal{D} \circ f)$ for any f. We argued in Appendix A.1 that FNM for depth-bounded functions

$$\begin{tikzcd}
\text{FNM for all functions} \arrow[r] \arrow[dr, leftrightarrow] & \text{FNM for } \mathcal{F}_\ell \arrow[r] \arrow[d, "m \cdot n \leq \ell"] & \text{FNM for } \mathcal{F}_1 \arrow[d, leftrightarrow] \\
& \text{NM} \arrow[r] & \text{Depth-bounded NM} \arrow[l, bend left, "/"{marking}, "n = 1"]
\end{tikzcd}$$

Figure 1: Relationship between functional non-malleability (FNM), standard non-malleability (NM), and depth-bounded non-malleability. Here, an arrow from definition A to definition B indicates that any construction satisfying A also satisfies B. For notation, we let $\mathcal{F}_\ell$ be class of depth-bounded functions with ℓ-bit output. We denote by m the message length of the TLP scheme, and by n the concurrency. We note that (except where indicated) the relationships hold for any setting of concurrency, namely, when all notions are non-concurrent, have bounded concurrency, or are fully concurrent.

with output length 1 is equivalent to depth-bounded NM since both f and $\mathcal{D}$ are depth-bounded and have output length 1. Finally, the negative arrow from depth-bounded NM to NM follows in the setting of $n = 1$ by Theorem A.3.

C One-Many Non-Malleability

We show that one-m functional non-malleable time-lock puzzles for any class of functions $\mathcal{F}$ are in fact m-concurrent functional non-malleable for $\mathcal{F}$. We note that this captures standard non-malleability whenever $\mathcal{F}$ contains the identity function. The structure of the proof follows from [LPV08], who show the claim for standard non-malleable commitments. We note that their proof requires relying on a definition of non-malleable commitments where the distinguisher also gets the view of the attacker. Similar to the setting of non-malleable codes, we cannot achieve such a notion for time-lock puzzles since a (even polynomial-time) distinguisher can trivially distinguish when receiving the view as input. However, because time-lock puzzles are non-interactive, we can still show the following composition theorem.

Lemma C.1 (One-many to concurrent)**.** *Let $B\colon \mathbb{N} \to \mathbb{N}$ with $B(\lambda) \in 2^{\text{poly}(\lambda)}$. If $(\mathsf{Gen}, \mathsf{Sol})$ is a B-secure time-lock puzzle that is one-many functional non-malleable for a class of functions $\mathcal{F}$, then it is also fully concurrent function non-malleable for $\mathcal{F}$.*

Proof. Let $(\mathsf{Gen}, \mathsf{Sol})$ be a B-secure one-many functional non-malleable time-lock puzzle for a class of functions $\mathcal{F}$.

To show the claim, suppose for contradiction that the time-lock puzzle is not fully-concurrent non-malleable, meaning that there exists a function $f \in \mathcal{F}$ such that for every positive polynomial α, there exists a function T with $\alpha(\lambda) \leq T(\lambda) \in \text{poly}(\lambda, B(\lambda))$ and polynomial n such that the following holds. Fix a probabilistic polynomial-time (n, n, B, α, T)-MIM adversary $\mathcal{A} = \{\mathcal{A}_\lambda\}_{\lambda \in \mathbb{N}}$, and suppose there exists a distinguisher $\mathcal{D} = \{\mathcal{D}_\lambda\}_{\lambda \in \mathbb{N}}$, a polynomial q, and a vector $\vec{s} = (s_1, \ldots, s_{n(\lambda)})$

such that for infinitely many $\lambda \in \mathbb{N}$ it holds that

$$\left|\Pr\left[\mathcal{D}_\lambda(f(\mathsf{mim}_{\mathcal{A}}(1^\lambda, T(\lambda), \vec{s})) = 1\right] - \Pr\left[\mathcal{D}_\lambda(f(\mathsf{mim}_{\mathcal{A}}(1^\lambda, T(\lambda), (0^\lambda)^{m(\lambda)})) = 1\right]\right| \geq \frac{1}{q(\lambda)}.$$

Looking ahead, let α_{tlp} be the positive polynomial associated with the functional non-malleability of $(\mathsf{Gen}, \mathsf{Sol})$ for f. We will show that this implies a $(1, n, B, \alpha_{\mathsf{tlp}}, T)$-MIM adversary $\mathcal{B}$ such that for infinitely many $\lambda \in \mathbb{N}$, there exists a value $s \in \{0,1\}^\lambda$ such that $\mathcal{D}_\lambda$ can be used to distinguish the output of $f(\mathsf{mim}_{\mathcal{B}}(1^\lambda, T(\lambda), s))$ from $f(\mathsf{mim}_{\mathcal{B}}(1^\lambda, T(\lambda), 0^\lambda))$, which will contradict the one-many functional non-malleability of the time-lock puzzle. As the above holds for any α, we will show a contradiction when $\alpha(\lambda) = \alpha_{\mathsf{tlp}}(\lambda)(1 + 2\beta(\lambda))$, for a polynomial β specified below.

To so do, fix any λ for which $\mathcal{D}_\lambda$ succeeds at distinguishing above, and for $i \in \{0, \dots, n(\lambda)\}$ define $\vec{v}^{(i)}$ to be the vector where the first i entries are $(s_1, \dots, s_i)$ and the remaining $n(\lambda) - i$ entries are set to 0^λ. As $\vec{v}^{(n(\lambda))} = \vec{s}$ and $\vec{v}^{(0)} = (0^\lambda)^{n(\lambda)}$, it follows by a hybrid argument that there exists an $i \in [n(\lambda)]$ such that

$$\begin{aligned}\Big|\Pr\left[\mathcal{D}_\lambda(f(\mathsf{mim}_{\mathcal{A}}(1^\lambda, T(\lambda), \vec{v}^{(i-1)})) = 1\right]& \\ - \Pr\left[\mathcal{D}_\lambda(f(\mathsf{mim}_{\mathcal{A}}(1^\lambda, T(\lambda), \vec{v}^{(i)})) = 1\right]\Big| &\geq \frac{1}{n(\lambda) \cdot p(\lambda)}\end{aligned} \tag{C.1}$$

We can now define the $(1, n, B, \alpha_{\mathsf{tlp}}, T)$-MIM adversary $\mathcal{B}_\lambda$, which we will use to contradict the one-many non-malleability of the time-lock puzzle relative to the value s_i. The adversary $\mathcal{B}_\lambda$ has $\vec{v}^{(i)}$ hardcoded, and receives as input a puzzle $z^\star$ either to s_i or 0^λ. It then does the following:

1. Sample puzzles $z_j \leftarrow \mathsf{Gen}(1^\lambda, T(\lambda), \vec{v}^{(i)}_j)$ in parallel for $j \in [n(\lambda)] \setminus \{i\}$.
2. Let $\vec{z} = (z_1, \dots, z_{i-1}, z^\star, z_{i+1}, \dots, z_{n(\lambda)})$, and run $\vec{\tilde{z}} \leftarrow \mathcal{A}_\lambda(\vec{z})$.
3. In parallel, check if any of the puzzles in $\vec{\tilde{z}}$ are equal to any puzzles in $\vec{z}$. If so, replace the matching puzzles with $\perp$, and output the resulting vector.

We will show that $\mathcal{B}_\lambda$ succeeds at breaking the one-n functional non-malleability of the time-lock puzzle, which suffices to break one-many functional non-malleability. We first analyze its success probability, and then its efficiency. To analyze its success probability, we claim that when $\mathcal{B}_\lambda$ receives a puzzle for s_i, then

$$\mathsf{mim}_{\mathcal{B}}(1^\lambda, T(\lambda), s_i) \equiv \mathsf{mim}_{\mathcal{A}}(1^\lambda, T(\lambda), \vec{v}^{(i)})$$

and when $\mathcal{B}_\lambda$ receives a puzzle for 0, it holds that

$$\mathsf{mim}_{\mathcal{B}}(1^\lambda, T(\lambda), 0^\lambda) \equiv \mathsf{mim}_{\mathcal{A}}(1^\lambda, T(\lambda), \vec{v}^{(i-1)}).$$

To see this, recall that for any vector $\vec{s}$, the distribution $\mathsf{mim}_{\mathcal{A}}(1^\lambda, T(\lambda), \vec{s})$ is given by (a) sampling a vector of puzzles for $\vec{s}$, (b) running $\mathcal{A}_\lambda$ on that vector to obtain its output vector, (c) replacing any puzzle that appear in both vectors with $\perp$, and (d) finding the unique solutions to the resulting puzzles (or $\perp$). Next, we compare this to the output of $\mathsf{mim}_{\mathcal{B}}$, both on input 0 and s_i:

- The distribution $\mathsf{mim}_{\mathcal{B}}$ first samples the puzzle $z^\star$, either to 0^λ or to s_i, and proceeds to run $\mathcal{B}_\lambda(z^\star)$, who computes the vector $\vec{z}$. When $\mathcal{B}_\lambda$ receives a puzzle for s_i, then it sets $\vec{z}$ to be a puzzle vector to $\vec{v}^{(i)}$, and when $\mathcal{B}_\lambda$ receives a puzzle for 0^λ, it sets the puzzle vector to correspond to $\vec{v}^{(i-1)}$. Therefore, $\vec{z}$ as computed by $\mathsf{mim}_{\mathcal{B}}(1^\lambda, T(\lambda), s_i)$ has the same distribution as the input to $\mathcal{A}$ computed by $\mathsf{mim}_{\mathcal{A}}(1^\lambda, T(\lambda), \vec{v}^{(i)})$, and $\vec{z}$ as computed by $\mathsf{mim}_{\mathcal{B}}(1^\lambda, T(\lambda), 0^\lambda)$ has the same distribution as the input to $\mathcal{A}$ computed by $\mathsf{mim}_{\mathcal{A}}(1^\lambda, T(\lambda), \vec{v}^{(i-1)})$.

- Next, $\mathcal{B}_\lambda$ computes $\vec{\tilde{z}} \leftarrow \mathcal{A}_\lambda(\vec{z})$, exactly as done in step (b) of $\mathsf{mim}_{\mathcal{A}}$.
- $\mathcal{B}_\lambda$ then replaces any puzzles in $\vec{\tilde{z}}$ that have been copied from $\vec{z}$ with $\perp$, exactly as in step (c) of $\mathsf{mim}_{\mathcal{A}}$. Note that at this point $\mathcal{B}_\lambda$ outputs $\vec{\tilde{z}}$, so $\mathsf{mim}_{\mathcal{B}}$ is done running $\mathcal{B}_\lambda$.
- Finally, $\mathsf{mim}_{\mathcal{B}}$ compares the puzzles in $\vec{\tilde{z}}$ with $z^\star$ and replaces any that have been copied, but since $z^\star$ is part of $\vec{z}$, this has already been done by $\mathcal{B}_\lambda$ and so doesn't change $\vec{\tilde{z}}$. Lastly, $\mathsf{mim}_{\mathcal{B}}$ uniquely solves these puzzles, exactly as in step (d) of $\mathsf{mim}_{\mathcal{A}}$.

We conclude that $\mathsf{mim}_{\mathcal{B}}(1^\lambda, T(\lambda), s_i) \equiv \mathsf{mim}_{\mathcal{A}}(1^\lambda, T(\lambda)), \vec{v}^{(i)})$ and $\mathsf{mim}_{\mathcal{B}}(1^\lambda, T(\lambda), 0^\lambda) \equiv \mathsf{mim}_{\mathcal{A}}(1^\lambda, T(\lambda), \vec{v}^{(i-1)})$. When coupling this with Equation C.1, it implies that for infinitely many $\lambda \in \mathbb{N}$, the distinguisher $\mathcal{D}_\lambda$ can distinguish $f(\mathsf{mim}_{\mathcal{B}}(1^\lambda, T(\lambda), s_i))$ and $f(\mathsf{mim}_{\mathcal{B}}(1^\lambda, T(\lambda), 0^\lambda))$ with inverse polynomial probability.

We next analyze the efficiency of $\mathcal{B}_\lambda$. Recall that $\mathcal{B}_\lambda$ samples $n(\lambda) - 1$ puzzles to form the vector $\vec{z}$, runs $\mathcal{A}_\lambda(\vec{z})$, and then compares the resulting puzzles given by $\mathcal{A}_\lambda$ to those in $\vec{z}$. As $\mathsf{Gen}(1^\lambda, T(\lambda), \cdot)$ runs in $\mathrm{poly}(\lambda, \log T(\lambda))$ time and $\mathsf{size}(\mathcal{A}_\lambda) \in \mathrm{poly}(\lambda, B(\lambda))$, it follows that

$$\mathsf{size}(\mathcal{B}_\lambda) \in \mathrm{poly}(\lambda, n(\lambda), \log T(\lambda), B(\lambda)) \in \mathrm{poly}(\lambda, B(\lambda)).$$

To analyze its depth, since the puzzles are sampled in parallel, this can be done in fixed polynomial depth in λ and $\log T(\lambda))$ by the efficiency of Gen. As $T(\lambda) \in 2^{\mathrm{poly}(\lambda)}$, this is bounded by a fixed polynomial $\beta(\lambda)$ (independent of n). Running $\mathcal{A}_\lambda$ requires at most $T(\lambda)/(\alpha(\lambda))$ depth since it is an (n, n, B, α, T)-MIM adversary. Finally, comparing the resulting puzzles to the original also requires $\beta(\lambda)$ depth, by comparing each of the $(n(\lambda))^2$ pairs in parallel, since the length of each puzzle is a priori bounded by $\beta(\lambda)$. Putting everything together, we have that

$$\begin{aligned}\mathsf{depth}(\mathcal{B}_\lambda) &= \frac{T(\lambda)}{\alpha(\lambda)} + 2\beta(\lambda) = \frac{T(\lambda) + 2\cdot\alpha(\lambda)\cdot\beta(\lambda)}{\alpha(\lambda)} \\ &\leq \frac{T(\lambda) + 2\cdot T(\lambda)\cdot\beta(\lambda)}{\alpha(\lambda)} = \frac{T(\lambda) + 2\cdot T(\lambda)\cdot\beta(\lambda)}{\alpha_{\mathsf{tlp}}(\lambda)\cdot(1+2\beta(\lambda))} = \frac{T(\lambda)}{\alpha_{\mathsf{tlp}}(\lambda)}\end{aligned}$$

where we used the fact that $\alpha(\lambda) \leq T(\lambda)$. Lastly, we need to show that $\alpha_{\mathsf{tlp}}(\lambda) \leq T(\lambda) \in \mathrm{poly}(\lambda, B(\lambda))$. We are given that $\alpha(\lambda) \leq T(\lambda) \in \mathrm{poly}(\lambda, B(\lambda))$ and $\alpha(\lambda) \geq \alpha_{\mathsf{tlp}}(\lambda)$. Therefore, this gives the desired bound on T, which gives the contradiction. $\square$

D Proofs from Section 5

In this section, we state and prove the theorems for our construction of publicly-verifiable, non-malleable time-lock puzzles of Section 5.

Lemma D.1 (Restatement of Lemma 5.6)**.** *For any polynomial T and unbounded algorithm Z that on input a random oracle $\mathcal{O}$ outputs polynomial-size circuits, there exists a negligible function* negl *such that for all $\lambda, n \in \mathbb{N}$, $i \in [n]$, it holds that*

$$\Pr\left[\begin{array}{l} \mathcal{O} \leftarrow \mathsf{RF}^{\lambda}_{\ell_{\mathsf{in}}} \\ \mathcal{A} = Z(\mathcal{O}) \\ \mathsf{crs}_i \leftarrow \{0,1\}^\lambda \\ (g, \mathbb{G}, y', \pi, \mathsf{mcrs}_{-i}) \\ \quad \leftarrow \mathcal{A}^{\mathcal{O}}(1^\lambda, \mathsf{crs}_i, T(\lambda), n) \\ \mathsf{mcrs} = (\mathsf{crs}_1, \ldots, \mathsf{crs}_n) \end{array} : \begin{array}{l} \mathsf{RSWVerify}^{\mathcal{O}}(1^\lambda, \mathsf{mcrs}, T(\lambda), (g, \mathbb{G}), y', \pi) = 1 \\ \wedge\ y' \neq g^{2^{T(\lambda)}} \\ \wedge\ (g, \mathbb{G}, *) \in \mathrm{Supp}\left(\mathsf{RSWGen}(1^\lambda, T(\lambda))\right) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

Proof. Let C be a checking algorithm that has hardcoded $\lambda, T(\lambda), n, i$ and on input $(g, \mathbb{G}, y', \pi, \mathsf{mcrs})$ outputs 1 if and only if the event in the experiment of the lemma statement holds. Then, we can rephrase what we want to show as $\Pr[\mathsf{Hyb}_0(\lambda))] \leq \mathsf{negl}(\lambda)$ where $\mathsf{Hyb}_0(\lambda)$ is defined as follows:

$$\mathsf{Hyb}_0(\lambda) = \left\{ \begin{array}{l} \mathcal{O} \leftarrow \mathsf{RF}^{\lambda}_{\ell_{\mathsf{in}}}; \quad \mathcal{A} = Z(\mathcal{O}) \\ \mathsf{crs}_i \leftarrow \{0,1\}^{\lambda} \\ (g, \mathbb{G}, y', \pi, \mathsf{mcrs}_{-i}) \leftarrow \mathcal{A}^{\mathcal{O}}(1^{\lambda}, \mathsf{crs}_i, T(\lambda), n) \\ \mathsf{mcrs} = (\mathsf{crs}_1, \dots, \mathsf{crs}_n) \end{array} : C(g, \mathbb{G}, y', \pi, \mathsf{mcrs}) = 1 \right\}.$$

Let $f(\lambda) = 2^{\lambda/2}$ and let Sam be the inefficient algorithm given in Lemma 3.6 that on input an adversary, outputs a partial assignment F on $f(\lambda)$ points. Consider the hybrid distribution $\mathsf{Hyb}_1(\lambda)$, defined as follows:

$$\mathsf{Hyb}_1(\lambda) = \left\{ \begin{array}{l} \mathcal{O} \leftarrow \mathsf{RF}^{\lambda}_{\ell_{\mathsf{in}}}; \quad \mathcal{A} = Z(\mathcal{O}) \\ F \leftarrow \mathsf{Sam}(\mathcal{A}); \quad \mathcal{P} \leftarrow \mathsf{RF}^{\lambda}_{\ell_{\mathsf{in}}}[F] \\ \mathsf{crs}_i \leftarrow \{0,1\}^{\lambda} \\ (g, \mathbb{G}, y', \pi, \mathsf{mcrs}_{-i}) \leftarrow \mathcal{A}^{\mathcal{P}}(1^{\lambda}, \mathsf{crs}_i, T(\lambda), n) \\ \mathsf{mcrs} = (\mathsf{crs}_1, \dots, \mathsf{crs}_n) \end{array} : C(g, \mathbb{G}, y', \pi, \mathsf{mcrs}) = 1 \right\}.$$

By Lemma 3.6, it follows that there exists a negligible function $\mathsf{negl}_1(\lambda) \in \sqrt{\mathrm{poly}(\lambda)/2^{\lambda/2}}$ such that $|\Pr[\mathsf{Hyb}_1(\lambda)] - \Pr[\mathsf{Hyb}_0(\lambda)]| \leq \mathsf{negl}_1(\lambda)$.

Next, we define an inefficient checking algorithm C' that additionally receives F as input and outputs 1 if and only if $C(g, \mathbb{G}, y', \pi, \mathsf{mcrs}) = 1$ and F doesn't contain an assignment for any inputs that contain crs_i. We next consider the hybrid $\mathsf{Hyb}_2(\lambda)$ defined as follows:

$$\mathsf{Hyb}_2(\lambda) = \left\{ \begin{array}{l} \mathcal{O} \leftarrow \mathsf{RF}^{\lambda}_{\ell_{\mathsf{in}}}; \quad \mathcal{A} = Z(\mathcal{O}) \\ F \leftarrow \mathsf{Sam}(\mathcal{A}); \quad \mathcal{P} \leftarrow \mathsf{RF}^{\lambda}_{\ell_{\mathsf{in}}}[F] \\ \mathsf{crs}_i \leftarrow \{0,1\}^{\lambda} \\ (g, \mathbb{G}, y', \pi, \mathsf{mcrs}_{-i}) \leftarrow \mathcal{A}^{\mathcal{P}}(1^{\lambda}, \mathsf{crs}_i, T(\lambda), n) \\ \mathsf{mcrs} = (\mathsf{crs}_1, \dots, \mathsf{crs}_n) \end{array} : C'(g, \mathbb{G}, y', \pi, \mathsf{mcrs}, F) = 1 \right\}.$$

Since $|F| = 2^{\lambda/2}$ and crs_i is uniform over 2^{λ} different values, this implies that $|\Pr[\mathsf{Hyb}_2(\lambda)] - \Pr[\mathsf{Hyb}_1(\lambda)]| \leq \mathsf{negl}_2(\lambda)$ for negligible function $\mathsf{negl}_2(\lambda) = 2^{\lambda/2}/2^{\lambda} = 2^{-\lambda/2}$.

$\Pr[\mathsf{Hyb}_2(\lambda)]$ is bounded by the probability that a non-uniform PPT algorithm can break the soundness of Pietrzak's VDF in the plain random oracle model for any group QR^+_N where N is the product of two safe primes (which is the case since $\mathbb{G}$ is in the support of RSWGen). By the analysis in [Pie19], it follows that any adversary that makes at most Q queries to the random oracle can break soundness with probability at most $Q \cdot (3/2^{\lambda}) \leq \mathsf{negl}_3(\lambda)$, which is negligible when Q is bounded by a polynomial.

Finally, we conclude that there exists a negligible function $\mathsf{negl}(\lambda) = \mathsf{negl}_1(\lambda) + \mathsf{negl}_2(\lambda) + \mathsf{negl}_3(\lambda)$ such that $\Pr[\mathsf{Hyb}_0(\lambda)] \leq \mathsf{negl}(\lambda)$, as required. □

Theorem D.2 (Restatement of Theorem 5.7)**.** *Let $B \colon \mathbb{N} \to \mathbb{N}$. Assuming the B-repeated squaring assumption holds for* RSWGen*, then there exists a B-secure strong trapdoor VDF in the ABO-string model. Soundness holds in the auxiliary-input random oracle model.*

Proof. Let $\mathsf{RSWGen}, \mathsf{RSWProve}, \mathsf{RSWVerify}$ be defined as above. The theorem follows by considering $\mathsf{VDF} = (\mathsf{Sample}_{\mathsf{vdf}}, \mathsf{Eval}_{\mathsf{vdf}}, \mathsf{TDEval}_{\mathsf{vdf}}, \mathsf{Verify}_{\mathsf{vdf}})$. In what follows, we separately argue completeness, soundness, trapdoor evaluation, honest evaluation, and sequentiality.

Completeness. Let $\lambda, t, n \in \mathbb{N}$, $g, \mathbb{G} \in \{0,1\}^*$, $\mathsf{mcrs} \in (\{0,1\}^\lambda)^n$, and $\mathcal{O} \in \mathsf{RF}^\lambda_{\ell_{\mathsf{in}}}$. If $(g, \mathbb{G})$ is invalid, then $\mathsf{Eval}_{\mathsf{vdf}}$ outputs $y = \pi = \bot$. In this case, $\mathsf{Verify}_{\mathsf{vdf}}$ outputs 1 as required. Otherwise, it must be the case that $g \in \mathbb{G}$ and $\mathbb{G} = \mathrm{QR}^+_N$ for some N. In this case, $\mathsf{Eval}^{\mathcal{O}}_{\mathsf{vdf}}$ computes $y = g^{2^t}$ and $\pi = \mathsf{RSWProve}^{\mathcal{O}}(1^\lambda, \mathsf{mcrs}, t, (g, \mathbb{G}), y)$. $\mathsf{Verify}^{\mathcal{O}}_{\mathsf{vdf}}$ outputs 1 if $\mathsf{RSWVerify}^{\mathcal{O}}(1^\lambda, \mathsf{mcrs}, t, (g, \mathbb{G}), y, \pi) = 1$, which holds by Lemma 5.5.

Soundness. Let $\mathcal{A} = \{\mathcal{A}_\lambda\}_{\lambda \in \mathbb{N}}$ be a non-uniform PPT adversary and T be a polynomial. We note that we only require soundness to hold for valid $(g, \mathbb{G})$ which are in the support of $\mathsf{Sample}_{\mathsf{vdf}}(1^\lambda, T(\lambda))$. By definition of $\mathsf{Sample}_{\mathsf{vdf}}$, this implies that $(g, \mathbb{G})$ are in the support of $\mathsf{RSWGen}(1^\lambda, T(\lambda))$. It follows by definition of $\mathsf{Eval}_{\mathsf{vdf}}$ and $\mathsf{Verify}_{\mathsf{vdf}}$ that if $\mathcal{A}$ violates soundness of VDF, then it also violates soundness for the proof of repeated squaring. By Lemma 5.6, this can happen with at most negligible probability in the auxiliary-input random oracle model.

Trapdoor evaluation. Let $\lambda, t, n \in \mathbb{N}$, $(g, \mathbb{G}, |\mathbb{G}|) \in \mathrm{Supp}\left(\mathsf{Sample}_{\mathsf{vdf}}(1^\lambda, t)\right)$, $\mathsf{mcrs} \in (\{0,1\}^\lambda)^n$, and $\mathcal{O} \in \mathsf{RF}^\lambda_{\ell_{\mathsf{in}}}$. Since $(g, \mathbb{G})$ are in the support of $\mathsf{Sample}_{\mathsf{vdf}}$, they must be valid. In this case, $\mathsf{TDEval}_{\mathsf{vdf}}$ outputs $y = g^{2^t \bmod |\mathbb{G}|}$ and $\mathsf{Eval}^{\mathcal{O}}_{\mathsf{vdf}}$ outputs $y' = g^{2^t}$. Since $|\mathbb{G}|$ is the order of the group, $y = y'$ by definition.

Honest evaluation. Computing $y = g^{2^t}$ takes time $t \cdot \mathrm{poly}(\lambda)$ to compute t sequential squares in $\mathbb{G}$. As discussed above, $\mathsf{RSWProve}$ takes time $t \cdot \mathrm{poly}(\lambda, \log t, n)$ to compute. It follows that there exists a polynomial p such that for all $\lambda, t, n \in \mathbb{N}$, $\mathsf{mcrs} \in (\{0,1\}^\lambda)^n$, and $\mathcal{O} \in \mathsf{RF}^\lambda_{\ell_{\mathsf{in}}}$, $\mathsf{Eval}^{\mathcal{O}}_{\mathsf{vdf}}(1^\lambda, \mathsf{mcrs}, t, \cdot)$ can be computed in time $t \cdot p(\lambda, \log t, n)$, as required.

Sequentiality. B-Sequentiality of VDF follows immediately from the B-repeated squaring assumption for RSWGen stated in Assumption 5.4. Namely, $\mathsf{Sample}_{\mathsf{vdf}}$ simply outputs $(g, \mathbb{G})$ given by RSWGen, and Eval_1 simply computes $g^{2^{T(\lambda)}}$. So sequentiality follows syntactically.

Encoding. The group $\mathbb{G} = \mathrm{QR}^+_N$ is a ring. The group can be encoded by the integer N represented by a string in $\{0,1\}^*$. Elements can be encoded by the string representation of integers in $[0, (N-1)/2]$. The natural bijective function $f_x(y)$ is simply addition $x + y$ in this group. $\square$

Theorem D.3 (Restatement of Theorem 5.8)**.** *Let $B\colon \mathbb{N} \to \mathbb{N}$. Suppose there exists a B-secure strong trapdoor VDF. Then, there exists a B-secure publicly verifiable time-lock puzzle with one-sided soundness.*

Proof. Let $\mathsf{VDF} = (\mathsf{Gen}_{\mathsf{vdf}}, \mathsf{Eval}_{\mathsf{vdf}}, \mathsf{TDEval}_{\mathsf{vdf}}, \mathsf{Verify}_{\mathsf{vdf}})$ be any one-time trapdoor VDF. The theorem follows by considering the construction $\mathsf{TLP} = (\mathsf{Gen}_{\mathsf{tlp}}, \mathsf{Sol}_{\mathsf{tlp}}, \mathsf{Verify}_{\mathsf{tlp}})$ given in Section 5.2. In what follows, we separately argue correctness, efficiency, hardness, completeness, and soundness for this transformation.

Correctness. Let $\lambda, t \in \mathbb{N}$, $s \in \{0,1\}^\lambda$, $z = (x, \mathcal{X}, c) \in \mathrm{Supp}\left(\mathsf{Gen}_{\mathsf{tlp}}(1^\lambda, t, s)\right)$, and $s' = \mathsf{Sol}_{\mathsf{tlp},1}(1^\lambda, \mathsf{mcrs}, t, z)$ for any $n \in \mathbb{N}$ and $\mathsf{mcrs} \in (\{0,1\}^\lambda)^n$. We need to show that $s' = s$. By definition of $\mathsf{Gen}_{\mathsf{tlp}}$, it holds that $(x, \mathcal{X}, \mathsf{td}) \in \mathrm{Supp}\left(\mathsf{Gen}_{\mathsf{vdf}}(1^\lambda, t)\right)$, $y = \mathsf{TDEval}_{\mathsf{vdf}}(1^\lambda, t, (x, \mathcal{X}), \mathsf{td})$, and $c = y \oplus x_s$ where x_s is the encoding of s. In particular, this implies that $x_s = y \oplus c$ since $\oplus$ is a bijection over $\mathcal{X}$ by the encoding property of VDF. By definition of $\mathsf{Sol}_{\mathsf{tlp}}$, it also holds that $y' = \mathsf{Eval}_{\mathsf{vdf},1}(1^\lambda, \mathsf{mcrs}, t, (x, \mathcal{X}))$ and $x_{s'} = y' \oplus c$ where $x_{s'}$ is the encoding of s'. By the trapdoor evaluation property of VDF, it holds that $y' = y$, so $x_{s'} = x_s$. Since the string encodings are unique, this implies that $s' = s$, as required.

Efficiency. We note that by honest evaulation property of VDF, there exists a polynomial p_{vdf} such that $\mathsf{Eval}_{\mathsf{vdf}}(1^\lambda, \mathsf{mcrs}, t, \cdot)$ is computable in time $t \cdot p_{\mathsf{vdf}}(\lambda, \log t, n)$. Computing $y \oplus c$ and encoding s as x_s takes fixed polynomial time $\beta(\lambda)$ independent of t. This implies that there exists a polynomial $p_{\mathsf{tlp}}(\lambda, \log t) \geq p_{\mathsf{vdf}}(\lambda, \log t, n) + \beta(\lambda)/t$ such that $\mathsf{Sol}_{\mathsf{tlp}}(1^\lambda, \mathsf{mcrs}, t, \cdot)$ can be computed in time $t \cdot p_{\mathsf{vdf}}(\lambda, \log t, n) + \beta(\lambda) \leq t \cdot p_{\mathsf{tlp}}(\lambda, \log t, n)$, as required.

Hardness. We show that B-hardness of TLP follows from the B-sequentiality of VDF. Suppose by way of contradiction that B-hardness of TLP does not hold. Specifically, for any positive polynomial α_{tlp} there exists a function $T \in \text{poly}(\lambda, B(\lambda))$ with $\alpha_{\mathsf{tlp}}(\lambda) \leq T(\lambda)$ for all $\lambda \in \mathbb{N}$, a non-uniform adversary $\mathcal{A} = \{\mathcal{A}_\lambda\}_{\lambda \in \mathbb{N}}$ where $\mathsf{size}(\mathcal{A}_\lambda) \in \text{poly}(\lambda, B(\lambda))$ and $\mathsf{depth}(\mathcal{A}_\lambda) \leq T(\lambda)/\alpha(\lambda)$ for all $\lambda \in \mathbb{N}$, a polynomial q, and strings $s, s' \in \{0,1\}^\lambda$ such that for infinitely many λ, it holds that

$$\left|\Pr\left[\mathcal{A}_\lambda(\mathsf{Gen}_{\mathsf{tlp}}(1^\lambda, T(\lambda), s)) = 1\right] - \Pr\left[\mathcal{A}_\lambda(\mathsf{Gen}_{\mathsf{tlp}}(1^\lambda, T(\lambda), s')) = 1\right]\right| > 1/q(\lambda),$$

where the probabilities are over the randomness of $\mathsf{Gen}_{\mathsf{tlp}}$ and $\mathcal{A}_\lambda$.

For any value $v \in \{0,1\}^\lambda$, let $\mathsf{Hyb}_v(\lambda)$ be the distribution $\mathsf{Gen}_{\mathsf{tlp}}(1^\lambda, T(\lambda), v)$. Let $\mathsf{Hyb}_v^{\mathsf{rand}}(\lambda)$ be the distribution that outputs $z = (x, \mathcal{X}, c)$ where $(x, \mathcal{X}, \mathsf{td}) \leftarrow \mathsf{Gen}_{\mathsf{vdf}}(1^\lambda, T(\lambda))$ and $c = r \oplus x_s$ where x_s is the encoding of s and $r \leftarrow \mathcal{X}$ is uniformly sampled. By assumption, it holds that

$$|\Pr[\mathcal{A}_\lambda(\mathsf{Hyb}_s(\lambda)) = 1] - \Pr[\mathcal{A}_\lambda(\mathsf{Hyb}_{s'}(\lambda))]| > 1/q(\lambda).$$

We next note that $\mathsf{Hyb}_s^{\mathsf{rand}}(\lambda)$ and $\mathsf{Hyb}_{s'}^{\mathsf{rand}}(\lambda)$ are identically distributed. This is because random element $r \leftarrow \mathcal{X}$ is uniformly random and, by the encoding property of VDF, $\oplus$ is a bijective function on $\mathcal{X}$. It follows that c is uniformly distributed over $\mathcal{X}$ in both $\mathsf{Hyb}_s^{\mathsf{rand}}(\lambda)$ and $\mathsf{Hyb}_{s'}^{\mathsf{rand}}(\lambda)$.

As a result, it must be the case that

$$|\Pr[\mathcal{A}_\lambda(\mathsf{Hyb}_s(\lambda)) = 1] - \Pr\left[\mathcal{A}_\lambda(\mathsf{Hyb}_s^{\mathsf{rand}}(\lambda))\right]| > 1/(2 \cdot q(\lambda))$$

for either s or s'. Assume that this holds for s without loss of generality. We show that this breaks the B-sequentiality of VDF.

We first define a reduction from adversaries against hardness of TLP to adversaries against sequentiality of VDF. Given any adversary $\mathcal{A} = \{\mathcal{A}_\lambda\}_{\lambda \in \mathbb{N}}$ for TLP and string $s \in \{0,1\}^\lambda$, we construct an adversary $\mathcal{B} = \{\mathcal{B}_\lambda\}_{\lambda \in \mathbb{N}}$ against the sequentiality of the one-time trapdoor VDF as follows. $\mathcal{B}_\lambda$ on input $(x, \mathcal{X}, a)$ computes $c = a \oplus x_s$ where x_s is the encoding of s and outputs $\mathcal{A}_\lambda(x, \mathcal{X}, c)$. Whenever a is equal to $y = \mathsf{TDEval}_{\mathsf{vdf}}(1^\lambda, t, x)$, we note that the distribution output by $\mathcal{B}_\lambda$ is indentically distributed to $\mathcal{A}_\lambda(\mathsf{Hyb}_s(\lambda))$. Whenever a is a random element $r \leftarrow \mathcal{X}$, it holds that the output of $\mathcal{B}_\lambda$ is indentically distributed to $\mathcal{A}_\lambda(\mathsf{Hyb}_s^{\mathsf{rand}}(\lambda))$. Thus, for any adversary $\mathcal{A}$ and string s, it holds that

$$|\Pr[\mathcal{B}_\lambda(x, y) = 1] - \Pr[\mathcal{B}_\lambda(x, r) = 1]| = \left|\Pr[\mathcal{A}_\lambda(\mathsf{Hyb}_s(\lambda)) = 1] - \Pr\left[\mathcal{A}_\lambda(\mathsf{Hyb}_s^{\mathsf{rand}}(\lambda)) = 1\right]\right|$$

We next argue that this reduction can be used to break the B-sequentiality of VDF. Let α_{vdf} be any positive polynomial, and let $p(\lambda)$ be the polynomial function representing the time to do an encoding and compute $\oplus$ for any domain $\mathcal{X}$ specified by VDF.

Consider any polynomial $\alpha_{\mathsf{tlp}}(\lambda) = \alpha_{\mathsf{vdf}}(\lambda) \cdot (1 + p(\lambda))$. Note that α_{tlp} is positive, polynomially bounded, and greater than α_{vdf} as α_{vdf}, p are polynomials and $1 + p(\lambda)$ is greater than 1. Thus, by assumption, there exists a function $T \in \text{poly}(\lambda, B(\lambda))$ with $\alpha_{\mathsf{tlp}}(\lambda) \leq T(\lambda)$, an adversary $\mathcal{A} =$

$\{\mathcal{A}_\lambda\}_{\lambda\in\mathbb{N}}$ with $\mathsf{size}(\mathcal{A}_\lambda) \in \mathrm{poly}(\lambda, B(\lambda))$ and $\mathsf{depth}(\mathcal{A}_\lambda) \leq T(\lambda)/\alpha_{\mathsf{tlp}}(\lambda)$, a polynomial q, and a string s satisfying

$$|\Pr[\mathcal{A}_\lambda(\mathsf{Hyb}_s(\lambda)) = 1] - \Pr\left[\mathcal{A}_\lambda(\mathsf{Hyb}_s^{\mathsf{rand}}(\lambda))\right]| > 1/(2\cdot q(\lambda))$$

for infinitely many $\lambda \in \mathbb{N}$, as defined above.

For the same function T, consider an adversary $\mathcal{B} = \{\mathcal{B}_\lambda\}_{\lambda\in\mathbb{N}}$ based on $\mathcal{A}$ and s given by the reduction above. Since $\alpha_{\mathsf{tlp}}(\lambda) \geq \alpha_{\mathsf{vdf}}(\lambda)$, it holds that T satisfies $\alpha_{\mathsf{vdf}}(\lambda) \leq T(\lambda)$ for all $\lambda \in \mathbb{N}$. Next, we note that $\mathsf{size}(\mathcal{B}_\lambda) \leq \mathsf{size}(\mathcal{A}_\lambda) + p(\lambda) \in \mathrm{poly}(\lambda, B(\lambda))$. Also, $\mathsf{depth}(\mathcal{B}_\lambda) \leq \mathsf{depth}(\mathcal{A}_\lambda) + p(\lambda) \leq T(\lambda)/\alpha_{\mathsf{tlp}}(\lambda) + p(\lambda)$. Given that $\alpha_{\mathsf{tlp}}(\lambda) = \alpha_{\mathsf{vdf}}(\lambda)\cdot(1+p(\lambda))$ and $T(\lambda) \geq \alpha_{\mathsf{tlp}}(\lambda)$, it holds that

$$p(\lambda) = \frac{\alpha_{\mathsf{tlp}}(\lambda)}{\alpha_{\mathsf{vdf}}(\lambda)} - 1 = \alpha_{\mathsf{tlp}}(\lambda)\cdot\left(\frac{1}{\alpha_{\mathsf{vdf}}(\lambda)} - \frac{1}{\alpha_{\mathsf{tlp}}(\lambda)}\right) \leq T(\lambda)\cdot\left(\frac{1}{\alpha_{\mathsf{vdf}}(\lambda)} - \frac{1}{\alpha_{\mathsf{tlp}}(\lambda)}\right).$$

This implies that

$$\begin{aligned}\mathsf{depth}(\mathcal{B}_\lambda) &\leq \frac{T(\lambda)}{\alpha_{\mathsf{tlp}}(\lambda)} + p(\lambda)\\ &\leq \frac{T(\lambda)}{\alpha_{\mathsf{tlp}}(\lambda)} + T(\lambda)\cdot\left(\frac{1}{\alpha_{\mathsf{vdf}}(\lambda)} - \frac{1}{\alpha_{\mathsf{tlp}}(\lambda)}\right)\\ &= \frac{T(\lambda)}{\alpha_{\mathsf{vdf}}(\lambda)},\end{aligned}$$

as required. Finally, we recall that, for $x \leftarrow \mathsf{Gen}_{\mathsf{vdf}}(1^\lambda, t)$, $y = \mathsf{TDEval}_{\mathsf{vdf}}(1^\lambda, t, x)$ and $r \leftarrow \{0,1\}^\lambda$,

$$\begin{aligned}&|\Pr[\mathcal{B}_\lambda(x,y) = 1] - \Pr[\mathcal{B}_\lambda(x,r) = 1]|\\ &= \left|\Pr[\mathcal{A}_\lambda(\mathsf{Hyb}_s(\lambda)) = 1] - \Pr\left[\mathcal{A}_\lambda(\mathsf{Hyb}_s^{\mathsf{rand}}(\lambda)) = 1\right]\right|\\ &> 1/(2\cdot q(\lambda)),\end{aligned}$$

for infinitely many $\lambda \in \mathbb{N}$ by assumption, in contradiction.

Completeness. Let $\lambda, t, n \in \mathbb{N}$, $z \in \{0,1\}^*$, and $\mathsf{mcrs} \in (\{0,1\}^\lambda)^n$. If z cannot be parsed as $(x, \mathcal{X}, c)$, $\mathsf{Sol}_{\mathsf{tlp}}$ outputs $(s,\pi) = (\bot,\bot)$. As $\mathsf{Verify}_{\mathsf{tlp}}$ can also check if z is parsed this way, it outputs 1 in this case, as required. Otherwise, z can be parsed as $(x, \mathcal{X}, c)$. Let $(y, \pi_{\mathsf{vdf}}) = \mathsf{Eval}_{\mathsf{vdf}}(1^\lambda, \mathsf{mcrs}, t, (x, \mathcal{X}))$ and $x_s = y \oplus c$ with string encoding s as computed by $\mathsf{Sol}_{\mathsf{tlp}}$. As $x, \mathcal{X} \in \{0,1\}^*$, it follows by completeness of VDF that $\mathsf{Verify}_{\mathsf{vdf}}(1^\lambda, \mathsf{mcrs}, t, (x, \mathcal{X}), (y, \pi_{\mathsf{vdf}})) = 1$. Since $x_s = y \oplus c$ by definition, it follows that $\mathsf{Verify}_{\mathsf{tlp}}$ outputs 1, as required.

One-sided soundness. Suppose there is a non-uniform PPT adversary $\mathcal{A} = \{\mathcal{A}_\lambda\}_{\lambda\in\mathbb{N}}$ that breaks one-sided soundness of TLP. Specifically, suppose there exist polynomials T, q, integers $n \in \mathbb{N}$ and $i \in [n]$ such that for infinitely many $\lambda \in \mathbb{N}$, it holds that

$$\Pr\left[\begin{array}{l}\mathsf{crs}_i \leftarrow \{0,1\}^\lambda\\ (z, s', \pi, \mathsf{mcrs}_{-i}) \leftarrow \mathcal{A}_\lambda(1^\lambda, \mathsf{crs}_i, T(\lambda), n)\\ \mathsf{mcrs} = (\mathsf{crs}_1, \ldots, \mathsf{crs}_n)\\ s = \mathsf{Sol}_1(1^\lambda, \mathsf{mcrs}, T(\lambda), z)\end{array} : \begin{array}{l}\mathsf{Verify}(1^\lambda, \mathsf{mcrs}, T(\lambda), z, (s', \pi)) = 1\\ \wedge\ s \neq s'\\ \wedge\ z \in \mathrm{Supp}\big(\mathsf{Gen}(1^\lambda, T(\lambda), \cdot)\big)\end{array}\right] > 1/q(\lambda).$$

We construct an adversary $\mathcal{B} = \{\mathcal{B}_\lambda\}_{\lambda\in\mathbb{N}}$ such that for the polynomial T, integers $n \in \mathbb{N}$ and $i \in [n]$, $\mathcal{B}$ breaks that soundness of VDF with probability $1/q(\lambda)$ for infinitely many $\lambda \in \mathbb{N}$. $\mathcal{B}_\lambda$ on input $(1^\lambda, \mathsf{crs}_i, T(\lambda), n)$ computes $(z, s', \pi, \mathsf{mcrs}_{-i}) = \mathcal{A}_\lambda(1^\lambda, T(\lambda), (x, \mathcal{X}, c))$, parses π as (y', π_{vdf}), and outputs $(x, \mathcal{X}, y', \pi_{\mathsf{vdf}}, \mathsf{mcrs}_{-i})$. First, we note that since $\mathcal{A}_\lambda$ runs in polynomial time, so does $\mathcal{B}_\lambda$. We next analyze the success probability of $\mathcal{B}$ for security parameter λ. First we note that when $\mathcal{A}_\lambda$ succeeds, the puzzle $z = (x, \mathcal{X}, c)$ output by $\mathcal{A}_\lambda$ is in the support of $\mathsf{Gen}(1^\lambda, T(\lambda), \cdot)$, so it also holds that $(x, \mathcal{X}, *) \in \operatorname{Supp}\left(\mathsf{Sample}_{\mathsf{vdf}}(1^\lambda, T(\lambda))\right)$. Next, whenever $\mathsf{Verify}_{\mathsf{tlp}}(1^\lambda, \mathsf{mcrs}, T(\lambda), (x, \mathcal{X}, c), (s', \pi))$ outputs 1, it holds that $x_{s'} = c \oplus y'$ and $\mathsf{Verify}_{\mathsf{vdf}}(1^\lambda, \mathsf{mcrs}, T(\lambda), (x, \mathcal{X}), (y', \pi_{\mathsf{vdf}})) = 1$. By completeness of TLP, it holds that $x_s = c \oplus y$. However, since $s \neq s'$, it holds that $x_s \neq x_{s'}$ so $y \neq y'$. It follows that for the polynomial T, integers $n \in \mathbb{N}$ and $i \in [n]$, for infinitely many $\lambda \in \mathbb{N}$, it holds that

$$\Pr\left[\begin{array}{l} \mathsf{crs}_i \leftarrow \{0,1\}^\lambda \\ (x, \mathcal{X}, y', \pi_{\mathsf{vdf}}, \mathsf{mcrs}_{-i}) \\ \qquad\qquad \leftarrow \mathcal{B}_\lambda(1^\lambda, \mathsf{crs}_i, T(\lambda), n) \\ \mathsf{mcrs} = (\mathsf{crs}_1, \ldots, \mathsf{crs}_n) \\ y = \mathsf{Eval}_{\mathsf{vdf},1}(1^\lambda, \mathsf{mcrs}, T(\lambda), (x, \mathcal{X})) \end{array} : \begin{array}{l} \mathsf{Verify}_{\mathsf{vdf}}(1^\lambda, \mathsf{mcrs}, T(\lambda), (x, \mathcal{X}), (y', \pi_{\mathsf{vdf}})) = 1 \\ \wedge\ y \neq y' \\ \wedge\ (x, \mathcal{X}, *) \in \operatorname{Supp}\left(\mathsf{Sample}_{\mathsf{vdf}}(1^\lambda, T(\lambda))\right) \end{array}\right]$$
$$> 1/q(\lambda),$$

in contradiction. □

Lemma D.4 (Correctness and completeness proofs)**.** *Let* $(\mathsf{Gen}, \mathsf{Sol}, \mathsf{Verify})$ *be the construction of a publicly verifiable time-lock puzzle from any one-sided publicly verifiable time-lock puzzle from Section 5.3.* $(\mathsf{Gen}, \mathsf{Sol}, \mathsf{Verify})$ *satisfy full correctness and completeness.*

Proof. We separately prove full correctness and completeness.

Full correctness. Let $\lambda, t \in \mathbb{N}$, $z \in \{0,1\}^*$, and $s' = \mathsf{Sol}_{\mathsf{tlp},1}(1^\lambda, \mathsf{mcrs}, t, z)$ for any $n \in \mathbb{N}$ and $\mathsf{mcrs} \in (\{0,1\}^\lambda)^n$.

In the case that $z \in \operatorname{Supp}\left(\mathsf{Gen}^{\mathcal{O}}(1^\lambda, t, s)\right)$ for some $s \in \{0,1\}^\lambda$, we need to show that $s' = s$. Let r be a value such that $z = \mathsf{Gen}^{\mathcal{O}}(1^\lambda, t, s; r)$ First note that $z \in \operatorname{Supp}\left(\mathsf{Gen}_{\mathsf{os}}(1^\lambda, t, (s \| r))\right)$. Next, Sol computes $(\hat{s}_{\mathsf{os}}, \pi_{\mathsf{os}}) = \mathsf{Sol}_{\mathsf{os}}(1^\lambda, \mathsf{mcrs}, t, z)$ and parses $\hat{s}_{\mathsf{os}} = \hat{s} \| \hat{r}$. By correctness of $\mathsf{Sol}_{\mathsf{os}}$, it holds that $\hat{s} = s$ and $\hat{r} = r$. Then, by assumption, it holds that $z = \mathsf{Gen}^{\mathcal{O}}(1^\lambda, t, s; r)$, so Sol outputs $s' = s$, as required.

In the case that $z \notin \operatorname{Supp}\left(\mathsf{Gen}^{\mathcal{O}}(1^\lambda, t, s)\right)$ for any $s \in \{0,1\}^\lambda$, we need to show that $s' = \bot$. Note that Sol only outputs a value $s' \neq \bot$ if $z = \mathsf{Gen}^{\mathcal{O}}(1^\lambda, t, s; r)$ for some s. By definition, this can only be the case for $z \in \operatorname{Supp}\left(\mathsf{Gen}(1^\lambda, t, s)\right)$ for some s. Thus, s' must be equal to $\bot$ in this case.

Completeness. Let $\lambda, t, n \in \mathbb{N}$, $z \in \{0,1\}^*$, $\mathsf{mcrs} \in (\{0,1\}^\lambda)^n$. Let $(s, \pi) = \mathsf{Sol}^{\mathcal{O}}(1^\lambda, \mathsf{mcrs}, t, z)$. We need to show that $\mathsf{Verify}^{\mathcal{O}}(1^\lambda, \mathsf{mcrs}, t, z, (s, \pi)) = 1$.

First, we consider the case where $z \in \operatorname{Supp}\left(\mathsf{Gen}^{\mathcal{O}}(1^\lambda, t, s)\right)$ for some $s \in \{0,1\}^\lambda$. This means that $z = \mathsf{Gen}^{\mathcal{O}}(1^\lambda, t, s; r)$ for some $r \in \{0,1\}^*$. In the proof of correctness above, we showed that $\mathsf{Sol}^{\mathcal{O}}(1^\lambda, \mathsf{mcrs}, t, z)$ output (s, r) in this case. Since $s \neq \bot$, verify outputs 1 if any only if $z = \mathsf{Gen}^{\mathcal{O}}(1^\lambda, t, s; r)$, which it does by assumption.

Next, we consider the case where $z \notin \operatorname{Supp}\left(\mathsf{Gen}^{\mathcal{O}}(1^\lambda, t, s)\right)$ for any $s \in \{0,1\}^\lambda$. In this case, we argued in correctness that $s = \bot$, so it must be the case that $\pi = (s_{\mathsf{os}}, \pi_{\mathsf{os}}) = \mathsf{Sol}_{\mathsf{os}}(1^\lambda, \mathsf{mcrs}, t, z)$. By completeness of the underlying TLP, we know that $\mathsf{Verify}_{\mathsf{os}}(1^\lambda, \mathsf{mcrs}, t, z, (s_{\mathsf{os}}, \pi_{\mathsf{os}})) = 1$. Thus, Verify outputs 1 as long as $z \neq \mathsf{Gen}^{\mathcal{O}}(1^\lambda, t, s; r)$, but this must be the case as $z \notin \operatorname{Supp}\left(\mathsf{Gen}^{\mathcal{O}}(1^\lambda, t, \cdot)\right)$ by assumption. □