Publicly Verifiable Proofs of Sequential Work

Mohammad Mahmoody* Tal Moran† Salil Vadhan‡

December 7, 2012

Abstract

We construct a publicly verifiable protocol for proving computational work based on collision-resistant hash functions and a new plausible complexity assumption regarding the existence of "inherently sequential" hash functions. Our protocol is based on a novel construction of time-lock puzzles. Given a sampled "puzzle" $\mathcal{P} \xleftarrow{\$} \mathbf{D}_n$, where n is the security parameter and $\mathbf{D}_n$ is the distribution of the puzzles, a corresponding "solution" can be generated using N evaluations of the sequential hash function, where $N > n$ is another parameter, while any feasible adversarial strategy for generating valid solutions must take at least as much time as $\Omega(N)$ *sequential* evaluations of the hash function after receiving $\mathcal{P}$. Thus, valid solutions constitute a "proof" that $\Omega(N)$ parallel time elapsed since $\mathcal{P}$ was received. Solutions can be publicly and efficiently verified in time $\text{poly}(n) \cdot \text{polylog}(N)$. Applications of these "time-lock puzzles" include noninteractive timestamping of documents (when the distribution over the possible documents corresponds to the puzzle distribution $\mathbf{D}_n$) and universally verifiable CPU benchmarks.

Our construction is secure in the standard model under complexity assumptions (collision-resistant hash functions and inherently sequential hash functions), and makes black-box use of the underlying primitives. Consequently, the corresponding construction in the random oracle model is secure unconditionally. Moreover, as it is a public-coin protocol, it can be made non-interactive in the random oracle model using the Fiat-Shamir Heuristic.

Our construction makes a novel use of "depth-robust" directed acyclic graphs—ones whose depth remains large even after removing a constant fraction of vertices—which were previously studied for the purpose of complexity lower bounds. The construction bypasses a recent negative result of Mahmoody, Moran, and Vadhan (CRYPTO '11) for time-lock puzzles in the random oracle model, which showed that it is impossible to have time-lock puzzles like ours in the random oracle model if the puzzle generator also computes a solution together with the puzzle.

*Cornell, mohammad@cs.cornell.edu. Research supported in part by NSF Awards CNS-1217821 and CCF-0746990, AFOSR Award FA9550-10-1-0093, and DARPA and AFRL under contract FA8750-11-2-0211.

†Interdisciplinary Center (IDC) Herzliya, talm@idc.ac.il. Work done in part while at the Harvard Center for Research on Computation and Society.

‡School of Engineering & Applied Sciences and Center for Research on Computation and Society, Harvard University, Cambridge, MA salil@seas.harvard.edu. Supported in part by NSF grant CCF-1116616. Work done in part while on leave as a Visiting Researcher at Microsoft Research SVC and as a Visiting Scholar at Stanford University.

1 Introduction

A *timestamping scheme* is a mechanism for proving that a document was created before a certain time in the past. Timestamping schemes have variety of applications, including the resolution of intellectual property disputes (e.g., an inventor may timestamp her invention to prevent future patent challenges) and providing evidence of predictive powers (e.g., a stock analyst could prove that she correctly predicted stock price changes before they occurred).

We say a timestamping scheme is *noninteractive* if generating a timestamp does not require communication with a third party. This is a desirable property, both because it makes timestamping easily scalable (multiple parties generating timestamps do not interfere with each other), and it allows parties to hide the fact that they are generating a timestamp; this might be crucial in some scenarios (e.g., an inventor may not wish to reveal the fact that she has a new invention).

A natural cryptographic approach to timestamping is via *proofs of work*—use computational effort invested as a measure of time elapsed. For example, if a party wants to be able to issue future proofs that she knows a document D at time t_0, then starting at time t_0, she starts to evaluate a "moderately hard" function g on document D. If we know that g takes time $\approx N$ to evaluate, then the value $g(D)$ can be considered a "proof" that D was known N time units in the past.

Note that here we need both an upper bound and a lower bound on the complexity of computing g—an adversary should not be able to evaluate g on D much more quickly than an honest party following the specified algorithm for g. In addition, we would like verifying $y = g(D)$ (given D and y) to be done much more efficiently than evaluating g on D from scratch.

These (initial) goals can be achieved by taking $g = f^{-1}$ for a very strong one-way permutation $f : \{0,1\}^n \to \{0,1\}^n$, where we take the security parameter to be $n = \log N$. Given a document $D \in \{0,1\}^n$, the proof of work $f^{-1}(D)$ can be computed by brute force in time approximately $2^n = N$. Such a proof can be verified very quickly (e.g., in time $\mathrm{poly}(n) = \mathrm{poly}(\log N)$). Moreover, it is a plausible assumption that any efficient[1] algorithm for inverting f will require time $\Omega(2^n) = \Omega(N)$, at least on a uniformly random document $D \xleftarrow{\$} \{0,1\}^n$. (If D is not uniformly distributed, then we can heuristically apply this construction to a hash of D, or even apply a publicly known deterministic randomness extractor tailored to the distribution $\mathbf{D}_n$ from which D is sampled.)

One deficiency of the aforementioned construction is that, while it certifies that N units of computational effort were invested after receiving D, this need not correspond to clock time, because an adversary could parallelize its computational efforts (e.g., by using a bot-net to try many preimages at once). Thus, we would like to have proofs of work that are inherently sequential, i.e., even a massively parallel effort to evaluate $g(D)$ would still take time close to N. (Of course, "time" is still relative to single-core CPU speed, which may differ between the honest party and the adversary, but this gap should be easier to gauge and control than what can be achieved by massive parallelism.)

Based on ideas from [CLSY93, RSW96], Jerschow and Mauve [JM10] proposed the following timestamping function which is conjectured to be secure against parallel attack: $g(D) = 2^{2^D} \pmod N$, for an RSA integer N whose factors are kept secret. A verifier

[1] Here the adversary is assumed to be uniform, because a non-uniform attacker can in fact invert a one-way permutation in time 2^{cn} for some constant $c < 1$ [Hel80, FN99, DTT10].

who *already knows* the secret factorization of N can check the computation efficiently using the "shortcut" $2^{2^D} \equiv 2^{(2^D \bmod \varphi(N))} \pmod N$; if $|N| \approx |D|$, this shortcut gives an exponential speed-up. The security of this scheme is based on the conjecture that modular exponentiation is an inherently sequential task without knowing the factorization of N.

Time-Lock Puzzles. The idea of using modular exponentiation as a proof of sequential work was first proposed by Cai, Lipton, Sedgewick and Yao [CLSY93], in the context of CPU benchmarks, and by Rivest, Shamir and Wagner [RSW96] in the context of *time-lock puzzles*. In a time-lock puzzle protocol, solving a puzzle should take approximately N time (even for a massively parallel solver), while generating the puzzle and verifying the solution should take considerably less. Thus, one can think of a time-lock puzzle as an interactive proof of sequential work and view a solution to the puzzle as a proof that at least roughly N time units have elapsed since the prover received the puzzle.

When used as a timestamping scheme, however, modular exponentiation has a serious drawback: the verifier must know and keep secret the factorization of the modulus. In practice, this means that the timestamper must decide in advance which verifiers to target. Moreover, if a verifier's secret key ever leaks, all timestamps using the corresponding public key can no longer be trusted.

To construct *publicly verifiable* timestamping schemes from time-lock puzzles, we employ *public-coin* time-lock puzzles and interpret the document D as the coin tosses generating the puzzle $\mathcal{P}$.

Generic Proofs of Work via Efficient Arguments. Suppose we are given an inherently sequential function family: $\{f_{\mathcal{P}} : \{0,1\}^n \mapsto \{0,1\}^n\}_{\mathcal{P} \in \{0,1\}^{\mathrm{poly}(n)}}$, meaning that the t-fold composition $f^t_{\mathcal{P}}(0^n)$ takes parallel time $\Omega(t)$ for a uniformly random $\mathcal{P} \xleftarrow{\$} \{0,1\}^{\mathrm{poly}(n)}$. A natural attempt to construct a proof of work is to use simple iteration of f: the prover chooses a function $f_{\mathcal{P}}$ from the family (determined by the puzzle $\mathcal{P}$) and begins with an initial fixed value $x_0 = 0^n$. In iteration i, the prover computes $x_i = f_{\mathcal{P}}(x_{i-1}) = f^i_{\mathcal{P}}(x_0)$. Assuming that every adversary that outputs $f^t_{\mathcal{P}}(x)$ must run in time $\Omega(t)$, sending x_t to the verifier constitutes a proof of $\Omega(t)$ work. We can make the verification time polylogarithmic in t by using an efficient (public-coin) argument system to prove that x_t was computed correctly. Efficient argument systems can be constructed based on collision-resistant hash functions [Kil92, Mic00, BG08] and can be made noninteractive in the random oracle model.

This approach appears conceptually simple, but hides complexity in the construction of the argument system: existing schemes all make use of complex Probabilistically Checkable Proofs (PCPs), and this appears to be an inherent property of efficient argument systems [RV10]. In contrast, the constants hidden in the asymptotic notation of our constructions are very small (see Theorem 3.7 for the parameters). A second drawback of the generic scheme is that it is non-black-box (a proof that x_t was "computed correctly" necessarily uses the code of the algorithm computing f). As well as being of theoretical interest, a black-box construction has practical advantages: the implementation can be made in a modular way, changing the underlying sequential function can be done easily and it may even be replaced with a hardware module (or a corresponding physical assumption).

1.1 Our Results

In this paper, following Mahmoody et. al [MMV11], we study proofs of work (in the spirit of Dwork, Goldberg, Naor, and Wee [DN92, DGN03, DNW05]) and noninteractive timestamping.

Time-Lock Puzzles and Noninteractive Timestamping. Our main result is the first black-box construction of a time-lock puzzle (secure against parallel attack) with a public-coin puzzle generator. As described above, this implies a publicly verifiable, noninteractive timestamping scheme. Our construction relies on the existence of collision-resistant hash functions and a new assumption that *sequential* hash functions exist; these are functions where it is infeasible to find any "hash chain" of length N in time $\ll N$, even if the adversary enjoys massive parallel computing power (see Definition 3.2 for a formalization). Note that both assumptions hold in the random oracle model.

The following theorem states our main result informally; see Theorem 3.7 for a formal statement.

Theorem 1.1 (Main Result—Informal)**.** *Given any family of collision-resistant hash functions $\mathcal{CH}$ and any family of sequential hash functions $\mathcal{SH}$ we construct a time-lock puzzle scheme that is secure for a puzzle distribution* $\mathbf{D}$ *assuming that $\mathcal{CH}$ and $\mathcal{SH}$ are both secure when their index s is sampled from the same distribution* $\mathbf{D}$*.*

The verification time for our time-lock puzzle (and the corresponding timestamping scheme) can be poly-logarithmic in the time it takes to solve the puzzle, and the soundness time-gap (the ratio between the running time of the honest solver and that of a successful adversarial solver) is bounded by a constant. That is, our honest solver evaluates the hash function N times, and any adversary whose solution is accepted by an honest verifier must have used time proportional to $c \cdot N$ *sequential* evaluations of the hash function for a constant c. Our construction is fully black box (in the terminology of [RTV04]). After the puzzle is generated noninteractively, the verification process for both the time-lock puzzle and corresponding timestamping scheme are interactive protocols (a noninteractive timestamping scheme may still have an interactive verification stage). However, since our construction is fully black box, it is unconditionally secure in the random oracle model, and in this model we can make the verification noninteractive using the Fiat-Shamir Heuristic.

Universally Verifiable Benchmarks. Cai et al. [CLSY93] suggested using proofs of work for running "uncheatable" benchmarks. Their idea is that a vendor can prove a supercomputer's performance by having it run a proof of work that is timed by the verifier. The soundness of the proof-of-work protocol would guarantee that vendors couldn't cheat by optimizing their code or modifying it in some other way. Cai et al. proposed using exponentiation modulo an RSA integer as the candidate function. This has the drawback, however, of being verifier-specific (since only the verifier who generated the modulus and knows the secret factorization can trust the results). Using our time-lock puzzle construction, combined with a public randomness beacon, this benchmark can be made "universally verifiable": the randomness beacon would be used as the puzzle generator (assuming that the beacon's output in the next time period cannot be predicted by the vendor), and the vendor would publish the solution. Since there is no secret information, anyone can verify the results of the benchmark.

Combinatorial Tools. Our construction involves a novel use of *depth-robust* graphs: these are directed acyclic graphs on N vertices with low degree (e.g. $\text{polylog}\, N$) whose depth remains $\Omega(N)$ even after removing any constant fraction of vertices. To prove that it has done a lot of computational work, our prover constructs a labeling of the vertices of a depth-robust graph G where each vertex v should be labeled with $u_v = H_{\mathcal{P}}(u_{v_1}, \ldots, u_{v_d})$, where $v_1, \ldots, v_d$ are all vertices that have edges pointing to v, $\mathcal{P}$ is the puzzle, and H is a family of sequential hash-functions. (This is well-defined and can be computed with $O(N)$ hash evaluations due to the acyclicity of G.) It then sends the verifier a short commitment to this labeling through a Merkle tree (see Section B) computed using a collision-resistant hash function. During the verification, the prover is then asked to reveal the labels of a few randomly chosen vertices v along with their in-neighbors, and the verifier checks that $u_v = H_{\mathcal{P}}(u_{v_1}, \ldots, u_{v_d})$ holds for each such vertex v. Intuitively, if the prover can pass this check for a large fraction of vertices v, due to depth-robustness of G, the labeling constructed by the prover must have a hash chain of length $\Omega(N)$, and the prover must have used time proportional to $\Omega(N)$ sequential evaluations of the hash function.

Depth-robust graphs were investigated in the 70's and 80's, motivated by efforts to prove lower-bounds on circuit complexity and Turing machine time [EGS75, Val77, PR80, Sch82, Sch83]. We use a construction of Erdős, Graham and Szemerédi [EGS75], which involves a recursive use of constant-degree expander graphs. For different settings of parameters, depth-robust graphs are related to matrix rigidity and the "grate" property of graphs, defined by Valiant [Val77]. As far as we know, our work is the first "positive" use of depth-robust graphs.

In the *random oracle model* (ROM), we assume that all parties have oracle access to a public random function $\mathcal{O} : \{0,1\}^* \to \{0,1\}^n$ (where n is the security parameter). The ROM provides us with a clean and convenient way of lower-bounding both the total work invested by a party (namely the number of oracle queries) and the parallel time invested (the number of rounds of adaptivity in oracle queries). For this measure of computation time, it can be easily shown that a random oracle is sequential (a proof appears in Section 4). Since our constructions make only black-box use of the hash functions, they are *unconditionally* secure in the ROM.

Although random oracles do not exist in the real world, a common heuristic for instantiating protocols in the random-oracle model is to replace the oracle with a cryptographic hash function (e.g., SHA-256) [BR93, FS86a]. While no concrete hash-function can satisfy all the properties of an ideal random oracle (in fact, there are examples of schemes that are provably insecure for *any* instantiation of the oracle by a concrete hash function [CGH04, GT03]), it seams reasonable to conjecture that these functions are highly sequential.

1.2 Related Work

Timestamping. While physical timestamps have been in use for many years (e.g., having a notary public physically stamp a document) their introduction to the digital realm, by Haber and Stornetta [HS91], was more recent. Haber and Stornetta's main idea relies on a *Timestamping Service* (TSS): a trusted third party that is responsible for generating and managing the timestamps. Further work in this direction has improved communication and computational complexity, and allowed the use of an *untrusted* TSS (for examples, see [ABSW01, BHS92, BdM91, BdM93, BL98, BLLV98, BLS00]). The state-

of-the-art schemes using third-party Timestamping Services are efficient, can give very precise timestamps and even hide the contents of a document that is stamped, but necessarily reveal the fact that a party is stamping *some* document.

The first construction of a *noninteractive* timestamping scheme was given by Moran, Shaltiel and Ta-Shma [MSTS09], in the Bounded Storage Model (BSM). In the BSM, parties have limited storage space, and there exists a source that periodically broadcasts huge random strings to all parties. (The strings are large enough that no party can store more than a constant fraction of a string in every time period.) To generate a timestamp on a document at time t, the stamper uses the document to select a subset of the random string at time t and stores that subset. At every time period, verifiers store a small random subset of the random string. To prove that a timestamp is valid, the stamper proves that her stored subset is consistent with the values stored by the verifier.

Proofs of Work. Dwork and Naor [DN92] originally suggested using proofs of work as a "pricing mechanism" to fight SPAM and other denial-of-service (DoS) attacks. (They proposed that a sender of an email message would provide a proof of work related to the message, making mass emailing more expensive.) For fighting SPAM, requiring the proof of work to be sequential is pointless: an attacker who is trying to generate multiple messages can generate the proofs for each message in parallel. On the other hand, they do care about preventing amortization attacks: computing the proof for n messages in a batch should require approximately n times the work as computing the proof for a single message. Dwork, Naor and Wee [DNW05] later considered a proof of work that is memory-bound rather than CPU-bound; this is preferable as the variance between CPU speeds is much larger than the variance between memory access times.

Hash Graphs. The work of [DNW05] also makes use of "hash graphs". Dwork et al. use their hash graph G to have the adversary (who provides the proof of work) to generate a random-looking table T (which is larger than the cache size and will be used as a pointer-jumping table in the proof-of-work protocol) by computing the hash labels of G using a random oracle and taking the label of the output nodes as T. The crucial property of the hash graph G is that the adversary should not be able to compute T by accessing the main memory only "a few" times and is forced to encounter many cache misses. The hash graph G of [DNW05] is constructed by concatenating two subgraph DAGs: G_1 and G_2 (where inputs of G_2 are the outputs of G_1) as follows. The DAG G_1 is a super-concentrator (from [PTC77]) which is a sparse graph with high connectivity: for any set S of s input nodes and any set T of s outputs nodes there are s vertex disjoint paths from S to T. The graph G_2 is a shallow DAG (of logarithmic depth) which is robust in the following sense: by removing any "small fraction" of the nodes of G_2 (together with their incident edges), still a "large fraction" of the inputs remain connected to a "large fraction" of the outputs. Thus, at a high level, [DNW05] also uses some robust DAGs as we do in this work but with a different notion of robustness; their graphs are shallow and are robust against losing connectivity of input and outputs, while our graphs have large depth and preserve (most of) this depth even after removing some fraction of the nodes.

Non-Parallelizable Proofs of Work. Although the inherent sequentiality property is not very useful against SPAM, inherently sequential proofs of work *can* be useful as a countermeasure against DoS attacks in other cases. With this use in mind, several

proof-of-work constructions (called "client puzzles" in this context) have been suggested. These include an improvement in the efficiency of Rivest et al.'s modular-exponentiation-based construction [RSW96] (by Karame and Capkun [KC10]), but also constructions based on different "structural" assumptions with conjectured security against parallel attacks: Tritilanunt et al. [TBFN07] constructed proofs of work based on the hardness of solving subset-sum problems and assuming that the LLL algorithm [LLL82] for finding the shortest vectors in lattices is *optimal* and *inherently sequential*. Jerschow and Mauve [JM11] constructed proofs of work under the assumption that computing square roots modulo a prime is an inherently sequential task. Both constructions are public coin, but they achieve only a polynomial time-gap between puzzle generation and solution.

To the best of our knowledge, the only construction based on general unstructured assumptions is the recent construction of Mahmoody, Moran and Vadhan [MMV11] in the random oracle model. However, that construction only achieves a linear time-gap between puzzle generation and solution and, similar to the modular-exponentiation-based constructions, is not public coin.

Bypassing a Lower-Bound. Mahmoody, Moran and Vadhan considered time-lock puzzles in the random oracle model [MMV11] and proved that for a large class of time-lock puzzles, a large gap between generation/verification time and time to solve the puzzle is not possible. This class of time-lock puzzles include those in which the verifier does *not* query the oracle, which includes as a special case puzzles in which the generator produces a solution (sent to the verifier) together with the puzzle itself. For any such puzzle, they show that if it requires t oracle queries to generate the puzzle, the puzzle can be solved in t adaptive rounds of queries (with only a polynomial overhead in the total number of queries compared to the honest solver). We bypass the lower-bound of [MMV11] by constructing a verifier that does indeed query the oracle, and in fact our construction gives a time-lock puzzle with a super-polynomial gap between solution and verification times.

2 Time-Lock Puzzles and Timestamping, Formal Definitions

In this section we formalize the notion of time-lock puzzles and its relation to proofs of work.

Definition 2.1. A *time-lock puzzle* is a game between three parties $(\mathsf{Gen}, \mathsf{Sol}, \mathsf{Ver})$ who receive the common input 1^n for security parameter n and $N = \text{poly}(n) > n$ as the "complexity of the puzzle" and act as follows.

1. The puzzle generator Gen generates a "puzzle" $\mathcal{P}$.
2. The puzzle solver Sol receives the puzzle $\mathcal{P}$ and outputs some "solution" $\mathcal{S}$ in time $N \cdot \text{poly}(n)$ (where $\text{poly}(n)$ here is independent of N).
3. The verifier Ver receives the puzzle $\mathcal{P}$ and a solution $\mathcal{S}$ and either accepts or rejects. (This step could be noninteractive or through interaction with Sol—see the discussion below).

We require the following properties. The exact definition of the running time function $\mathbf{Time}(\cdot)$ and its parallel variant $\mathbf{ParTime}(\cdot)$ depend on the underlying computational model.

- **Completeness.** In an honest execution Ver accepts with probability $1 - \mathrm{negl}(n)$.
- **(Parallel) η-Soundness.** The (parallel) η-soundness property asserts that every non-uniform adversary $\widehat{\mathsf{Sol}}$ that runs in *parallel* time that is slightly smaller than the time of the honest solver ($\mathbf{ParTime}(\widehat{\mathsf{Sol}}) < \eta \cdot \mathbf{Time}(\mathsf{Sol})$) will fail to convince Ver with more than negligible probability (i.e., the probability that the output of $\widehat{\mathsf{Sol}}$ is accepted by Ver is negligible). We allow the total work $\mathbf{Time}(\widehat{\mathsf{Sol}})$ to be much larger than $\mathbf{Time}(\mathsf{Sol})$ (as long as it is at most polynomial in the security parameter).

Time Gap. For our applications, we are interested in time-lock puzzles where there is a time-gap between solving a puzzle (honestly or maliciously) and time it takes to verify a solution.

On Parallel Soundness. Intuitively, this guarantees that a successful $\widehat{\mathsf{Sol}}$, after receiving the puzzle, must have invested almost as much computational effort in terms of parallel-time complexity, as the honest solver Sol does in terms of its running time (η measures the "slowdown factor" of the honest solver compared to the parallel-time of the adversary—when $\eta = 1$ the adversary cannot solve any faster than the honest solver even with massive parallelism, while when $\eta = 1/2$ the adversary can run in half the time of the honest solver).

Note that we allow the adversary to perform arbitrary polynomial-time preprocessing before receiving the puzzle. For simplicity, we omit this from the definition above. However, our proof of security holds for a *non-uniform* adversary; in particular, this means we allow the adversary to receive arbitrary "advice" (that does not depend on the input, $\mathcal{P}$); any preprocessing done before receiving $\mathcal{P}$ can simply be treated as advice.

Interactive Verification. As we mentioned in Definition 2.1, the verification of the puzzle solution may be interactive (i.e., consist of an interactive protocol between the verifier and a "prover" who has access to the secret state of the puzzle solver Sol). The time-lock puzzles we construct in this paper have interactive verifiers. However, since their verification protocols are all public-coin, they can be made noninteractive in the random-oracle model by applying the Fiat-Shamir transformation.

Public vs. Secret Verification. Since the puzzle generator and the verifier in Definition 2.1 do not share any secret information, this definition guarantees that puzzles are "publicly verifiable": the verification protocol can be executed by any interested party. This property is especially useful for noninteractive time-lock puzzles: the solution and proof are both strings and can be easily published. The public verifiability means that anyone who receives the proof can verify it, without needing to trust any third parties. One can also consider an alternative definition of time-lock puzzles in which the puzzle generator and verifier do share secret information. This is the case, for example, in the repeated-squaring based puzzle of Rivest et al. [RSW96] (the secret information is the factorization of the modulus). The alternative definition is strictly weaker, and limits verification to parties that know the secret.

Timestamping Documents. Now we discuss how our results in the context of time-lock puzzles can be used for noninteractively timestamping documents. The type of timestamps we deal with are *relative* timestamps: that is, if Alice produces a d-timestamp of a document D at time T then she is claiming that she "knew" the document at time $T - d$. While relative timestamps can be used to construct absolute timestamps (e.g., by continuously computing d-timestamps for larger and larger d), they may also have direct applications. For example, suppose Alice believes she has solved a hard research problem, but is hesitant to publish before completely verifying her result. She can compute a one-week relative timestamp of her solution and store it; if, at some later time, Bob claims to have solved the problem, Alice has one week in which she can prove that she had a solution first. Our definition of security for timestamping is based on the following intuition: if Bob receives a previously unknown document D' at time T_0, then at time $T_0 + d$ he should not be able to produce a d'-timestamp of D' for any d' such that $d' \gg d$.

Definition 2.2. A *noninteractive timestamping scheme* is a protocol between a stamper, Stamp and a verifier, Ver. The protocol has two phases: In the *stamping* phase, the stamper receives an input document D and a duration d and computes the timestamp $\mathcal{S} = \mathsf{Stamp}(D, d)$. In the *verification* phase, the stamper communicates with the verifier. The stamper sends $(D, d, \mathcal{S})$ to the verifier and proves that $\mathcal{S}$ is a valid d-timestamp of D (this may be done using an interactive protocol). We also demand the following properties (where n is the security parameter).

- **Completeness.** We require a timestamping scheme to satisfy two completeness properties:
  - When parties execute the game honestly Ver accepts with probability $1 - \mathrm{negl}(n)$.
  - The honest stamper can compute $\mathsf{Stamp}(D, d)$ in d time.
- **η-Soundness** The soundness of the scheme is parameterized by $\eta \leq 1$ (measuring how much the adversary can cheat on the claimed timestamp duration) and by $\mathbf{D}$, the distribution of documents to be stamped. Consider the following game between an adversary $\mathbf{Adv}$ and Ver:
  1. $\mathbf{Adv}$ receives a document $D \xleftarrow{\$} \mathbf{D}$.
  2. $\mathbf{Adv}$ generates $(\mathcal{S}, d)$ and sends $(D, d, \mathcal{S})$ to the verifier Ver.
  3. Ver and $\mathbf{Adv}$ engage in the verification phase, after which Ver either accepts or rejects.

  We say the timestamp scheme is η-sound with respect to document distribution $\mathbf{D}$ if for every non-uniform adversary $\mathbf{Adv}$ of size $\mathrm{poly}(n)$ and whose parallel time (including the verification phase) is $\mathbf{ParTime}(\mathbf{Adv}) < \eta \cdot d$, the probability that $\mathbf{Adv}$ wins in the security game (by causing the verifier to accept) is negligible. (Note that, as in Definition 2.1, we allow the the adversary to perform arbitrary preprocessing before receiving the document to timestamp.)

Timestamping Using Time-Lock Puzzles. For any time-lock puzzle $(\mathsf{Gen}, \mathsf{Sol}, \mathsf{Ver})$ that is η-sound we can envision $(\mathsf{Sol}, \mathsf{Ver})$ as defining a timestamping scheme that is η-sound against documents in the distribution generated by Gen. That is, when Bob receives a document D at time T and wants to timestamp it for duration d, he treats D as the puzzle and "solves" it for parameter N such that $\mathbf{Time}(\mathcal{S}) = N \cdot \mathrm{poly}(n) = d$ to obtain the timestamp $\mathcal{S}$. The η-soundness of the time-lock puzzle guarantees that no adversary running in parallel time less than $\eta \cdot \mathbf{ParTime}(\mathsf{Sol})$ can convince the verifier to accept a valid timestamp.

The distribution $\mathbf{D}$ models the uncertainty about the documents to be timestamped. Any document D such that $\Pr[D \xleftarrow{\$} \mathbf{D}] \geq 1/\mathrm{poly}(n)$ (where n is the security parameter) can be thought of as a "typical" document that the adversary can "guess" in advance. Therefore, it is reasonable to restrict ourselves to distributions that satisfy $\Pr[D \xleftarrow{\$} \mathbf{D}] \leq 1/\mathrm{poly}(n)$ for every D and every polynomial $\mathrm{poly}(n)$; namely, the distribution $\mathbf{D}$ has super-logarithmic min-entropy. It is plausible to assume we have a family of hash functions $\mathcal{CH}$ and $\mathcal{SH}$ that are simultaneously secure for all index distributions with super-logarithmic min-entropy (they exist in the random oracle model). Our timestamping scheme will then inherit this property.

Security in the Presence of Auxiliary Information. Note that in Theorem 1.1, the distribution $\mathbf{D}$ of the documents is the same as the index distribution of the used hash functions. The distribution $\mathbf{D}$ could capture the uncertainty of the adversary about the document D *conditioned on* the auxiliary information of the adversary about the document. Thus, if we want to guarantee the security of our scheme under *any* auxiliary information (e.g., knowing only half of the document), as long as there is $\omega(\log n)$ bits of uncertainty about the final version of the document, we can construct secure timestamping schemes under the assumption that the used hash functions are also secure under any index distribution with super-logarithmic min-entropy.

Timestamping for Unknown Duration. A desirable feature of a timestamping scheme is to allow the stamper to perform its job *without* knowing the time period d in advance. A simple trick (that decreases η by a small constant factor) is as follows: The stamper iteratively generates timestamps for $d = 1, 2, \ldots, 2^i, \ldots$ and always keeps the last generated timestamp as the current timestamp. It is easy to see that this scheme achieves $\eta' \geq \eta/4$. Our time-lock puzzle (using the specific construction of depth-robust graphs from Section A) construction satisfies an even stronger property, making it particularly suited for timestamping documents without knowing the duration in advance. In our construction, for $d_2 > d_1$, the computation $\mathsf{Sol}_{d_1}(P)$ is a prefix of the computation $\mathsf{Sol}_{d_2}(P)$; thus, it is possible to run the stamper continuously, generating timestamps of increasing duration and without paying the constant factor in η.

3 Constructing Time-Lock Puzzles

In this section we prove our main result which is a new construction of publicly verifiable time-lock puzzle based on sequential hash functions and collision-resistant hash functions.

3.1 Outline and High-Level Ideas

The high-level idea is to force the solver to compute a labeling of a DAG G as a "hash-graph" in which each node is labeled with the hash of its in-neighbors' labels. The soundness of of our construction will rely on a computational assumption about the hashes—namely that they are inherently *sequential* (see Definition 3.2 below for a formalization). The sequentiality of the hash functions ensures that any adversary that can output the label of an endpoint of a correct path in the graph must spend time at least proportional to the length of the path (i.e., there are no "shortcuts"). To check a proposed solution to the puzzle, the verifier will test random nodes in the graph and check if they were computed correctly. We use an underlying DAG that has a special combinatorial property called *depth-robustness*: in any large subset of the nodes of the hash-graph there is a "long" path (see Definition 3.6). To achieve η-soundness with η arbitrarily close to one we employ a modified version of the depth-robust DAGs constructed by Erdős, Graham, and Szemerédi [EGS75] (see Section A).

If too many nodes in the graph are badly labeled (i.e., their labels are not the hash of their in-neighbors' labels) the verifier will reject the solution with high probability. Thus, if the verifier accepts we can use the combinatorial property of the graph to conclude that there must be a long path consisting of nodes that were "correctly" computed, and from the sequential property of the hash it will follow that the solver must have spent a "long" time performing the computation.

Finally, to reduce the communication and verification time, instead of sending the labeling of the entire graph, the solver will commit to the graph labeling and send only the commitment to the verifier. When the verifier challenges the solver on a specific node in the graph, the solver will send the label of that node and the labels of its in-neighbors, and will prove that these are consistent with the commitment. An efficient commitment scheme with all of the required properties is the Merkle tree, which can be constructed based on any collision-resistant hash function.

3.2 Formal Definitions

In this subsection we provide the formal definitions required for stating and proving our theorems. First, we formally define the notion of a sequential function. We first describe the security game.

Construction 3.1. We define a sequential function family by means of a game between an adversary and a challenger. Let n denote the security parameter. For every n and $m = m(n) \geq n$, let $H = \{h_s \colon \{0,1\}^m \mapsto \{0,1\}^n\}$ be a family of functions mapping m bits to n bits. The game is defined with respect to a distribution $\mathbf{D}_n$ defined over the indices of H. Both parties receive as inputs n, m and a "sequence-length" parameter N.

1. The challenger samples $s \xleftarrow{\$} \mathbf{D}_n$ as the index of the hash function and sends s to the adversary.
2. The adversary sends back some string y.
3. After y is received by challenger, the adversary sends back a sequence $y_0, \ldots, y_N = y$.

The adversary wins if for all $i \in [N]$ it holds that $h_s(y_{i-1})$ is a contiguous substring of y_i.

Definition 3.2 (Sequential Functions)**.** Let $\tau(n)$ be $0 < \tau(n) < \text{poly}(n)$, and suppose $H = \{h_s \colon \{0,1\}^{m(n)} \mapsto \{0,1\}^n\}$ is a family of (non-expanding) functions, and $D = \{\mathbf{D}_n\}_{n\in\mathbb{N}}$ be a family of distributions over the indexes $s \in \{0,1\}^{\text{poly}(n)}$ of H. The family H is called τ-*sequential* against $\mathbf{D}$ iff for every $N = \text{poly}(n)$ and every adversary $\mathbf{Adv}$ of the following form, the probability that $\mathbf{Adv}$ wins in the game of Construction 3.1 over common inputs n is $\text{negl}(n)$: the adversary $\mathbf{Adv}$ is a circuit of total size $\text{poly}(n, N)$, but the depth of this circuit till sending over y (in Step 2) is at most $\tau \cdot N$.

The sequentiality parameter τ determines the level of confidence one has in the sequential nature of H and is always at most $t(n)$, where $t(n)$ is the time it takes to call $H(\cdot)$ on an input to get an n-bit output. On an extreme point, for a particular function H, one might believe that getting a sequence of length N really needs N sequential evaluations of the function resulting in $\tau(n) = t(n)$.

Our construction also employs a family of collision-resistant hash functions. We use the following generalized definition that holds with respect to particular distributions over the index:

Definition 3.3 (Collision-Resistant Hashing)**.** Let $H = \{h_s \colon \{0,1\}^{2n} \mapsto \{0,1\}^n\}$ be a family of shrinking functions and let $\mathbf{D}_n$ be a distribution over the indexes of H for security parameter n. We call H *collision resistant* against $\mathbf{D} = \{\mathbf{D}_n\}$ iff for every non-uniform adversary $\mathbf{Adv}$ of size $\text{poly}(n)$, if we choose $s \xleftarrow{\$} \mathbf{D}_n$ and let $(x_1, x_2) = \mathbf{Adv}(s)$, then it holds that $\Pr[h_s(x_1) = h_s(x_2) \wedge x_1 \neq x_2] \leq \text{negl}(n)$.

Definition 3.4 (Directed Acyclic Graphs)**.** A *directed acyclic graph* (or DAG for short) $G = (V_G, E_G)$ is a directed graph whose vertices V_G can be renamed as $V_G = \{1, \ldots, N\}$—called the *topological order*—such that for every directed edge $(i, j) \in E_G$ it holds that $i < j$. We assume that our DAGs are always given in topological order. For any vertex $j \in [N]$ we call $\text{IN}(j) = \{i \mid (i, j) \in E_G\}$ the *in-neighbors* of the node j and call $d_{\text{in}}(j) = |\text{IN}(j)|$ the *in-degree* of the vertex j. We say G is of *in-degree* d if $d_{\text{in}}(j) \leq d$ for all $j \in V_G$. We call a family $\{G_N\}$ of DAGs where G_N has N vertices and in-degree d_N *explicit* if for any given $i \in [N]$ and $j \in [d_N]$ one can compute in time $\text{polylog}(N)$ the index of the j-th in-neighbor of the node i.

Remark 3.5. For simplicity we always assume that there is an extra redundant node 0 (not counted in the number of vertices) such that for every $j \in [N]$ there are $d - d_{\text{in}}(j)$ multiple edges from the node 0 to the node j to make the in-degrees of every $j \in [N]$ exactly equal to d.

Definition 3.6 (Depth Robustness)**.** For $\alpha \in [0, 1]$ and $\beta \in [0, \alpha]$, we call a DAG $G = (V_G, E_G)$ an (α, β)-*depth-robust* graph iff every induced subgraph H of G whose number of vertices is at least $|V_H| \geq \alpha \cdot |V_G|$ includes a path with at least $\beta \cdot |V_G|$ many vertices.

3.3 The Main Theorem

In this section we describe and prove our main result about the existence of time-lock puzzles based on collision-resistant and sequential hash functions. As discussed above, this implies the existence of timestamping protocols (a formal proof appears in Section 2).

Used Hash Functions. Let $\mathcal{CH} = \{ch\colon \{0,1\}^{2n} \mapsto \{0,1\}^n\}$ be a family of collision resistant hash functions against index distribution $\mathbf{D}$ and let $\mathcal{SH} = \{sh\colon \{0,1\}^m \mapsto \{0,1\}^n\}$ for $m = \omega(n \cdot \log^3 n)$ be a τ-sequential function against the same index distribution $\mathbf{D}$. Let $t_{ch}(n)$ denote the time required for an honest user to evaluate $ch(x)$ for $ch \in \mathcal{CH}$ when $ch(x) \in \{0,1\}^n$, and similarly, let $t_{sh}(n)$ be the time required for an honest user to evaluate an input for function $sh \in \mathcal{SH}$ with output in $\{0,1\}^n$. When clear from the context, we drop the index n and use names t_{sh}, t_{ch}, etc.

Theorem 3.7 (Main Result)**.** *Let n be the security parameter, $N = \text{poly}(n)$, and $k = \omega(\log n)$. Suppose $\mathcal{CH}$ and $\mathcal{SH}$ as described above are sequential and collision-resistant hash functions with respect to index distribution $\mathbf{D}$ and let $\beta < 1$ be any constant. Then there exists a time-lock puzzle as follows:*

- **Generation.** *The puzzle generator* Gen *simply outputs a sample $\mathcal{P} \xleftarrow{\$} \mathbf{D}$.*
- **Solving.** *The honest solver* Sol *runs in time $(t_{ch} + t_{sh} + \text{polylog}(N)) \cdot N$ to generate the puzzle solution.*
- **Verification.** *To answer the verifier's challenges, the solver is only required to lookup the answers from a table generated during solving the puzzle (no need for additional computation). This only takes linear time over the answer size which is $k \cdot (m + n)$. The* interactive *verifier* Ver *only asks k public-coin challenges, and to verify the received answer it runs in time at most $O(k) \cdot (t_{sh} + t_{ch} \cdot \log^3 N + \text{polylog}(N))$.*
- **Completeness.** *When the honest solver* Sol *interacts with the verifier* Ver*,* Ver *accepts with probability* 1*.*
- **Soundness.** *No malicious solver* $\widehat{\mathsf{Sol}}$ *computed by a circuit of size* $\text{poly}(n, N)$ *is able to make* Ver *accept with probability more than* $\text{negl}(n)$ *if* $\widehat{\mathsf{Sol}}$*'s circuit has depth $\tau \cdot N \cdot \beta$ till sending the puzzle solution. (Recall that $\mathcal{SH}$ is τ-sequential.)*

Remark 3.8 (Balancing Hashing Times)**.** It is easy to see that the soundness of Theorem 3.7 as stated implies η-parallel-soundness for $\eta \geq \tau\beta/(t_{ch} + t_{sh})$Thus we obtain $\eta = \tau\beta/(t_{ch} + t_{sh})$.Thus, on this extreme the advantage of the adversary relative to the honest solver, depends on the ratio between τ, the lower bound on the (average) time it takes the adversary to compute the sequential hash function sh, and $t_{sh} + t_{ch}$, the total time an honest party needs to evaluate sh once and ch once (where ch is the collision-resistant hash function). Since the adversary can always use the honest algorithm to compute sh, therefore it holds that $\tau \leq t_{sh}$. Thus, if $t_{ch} \approx t_{sh}$ (as would be the case, for example, if we used SHA-256 for both purposes), the adversary potentially could get a factor ≈ 2 advantage—even if the function is perfectly sequential ($\tau = t_{sh}$). A possible way to deal with this problem is by using the iterated function sh^k as the sequential function (where $sh^i(x) = sh(sh^{i-1}(x))$). Since any length-$N$ hash-chain of sh^k contains a length-$(k \cdot N)$ hash-chain for sh, it follows that $\tau(sh^k) \geq k \cdot \tau(sh)$. At the same time, $t_{sh^k} = k \cdot t_{sh}$. By increasing k, we can make the ratio $\tau/(t_{sh} + t_{ch})$ arbitrarily close to τ/t_{sh} (which is the best we can hope for). However, we pay for this increase by also increasing the computation time for the honest verifier; thus as k gets larger, the time-gap for the puzzle (the difference between the work done by the honest solver and the honest verifier) becomes smaller.

We describe our construction using an interactive verifier in a hybrid model with access to an ideal commitment functionality Com that remains hiding against selective opening (the commitment scheme allows a short commitment to a set of blocks $(u_1, \ldots, u_n)$, and later allows the prover to selectively open block u_j for every $j \in [N]$). We then replace the ideal commitment with a commitment scheme using Merkle tree based on the collision-resistant hash function $\mathcal{CH}$. If a single evaluation of the hash function takes time t_{ch}, then commitment to N blocks using Com can be done in time $t_{ch} \cdot N$ and verification in time $t_{ch} \cdot \log N$ (see Section B for details). Algorithms 1 and 2 describes the honest solver and Algorithm 3 describes the corresponding verifier.

Algorithm 1 Honest solver Sol solving puzzle $\mathcal{P}$ using sequential hash family sh indexed with $\mathcal{P}$ and output length n, and a DAG G of in-degree d and N vertices given in topological order.

1: Initially assign the hash label $u_0 = 0^n$ to the extra redundant node (see Remark 3.5).

2: **for** $v \in \{1, \ldots, N\}$ **do** {Compute the hash-labels corresponding to the nodes of G}
3: Suppose $v_1 \leq \cdots \leq v_d$ are the in-neighbors of v, and let u_{v_i} be the hash-label of v_i. Set $u_v = sh_{\mathcal{P}}(u_{v_1}, \ldots, u_{v_d})$.

Algorithm 2 Honest solver Sol answering a challenge for the generated solution c.

1: Receive a set of challenge nodes $\{v_1 \leq \ldots \leq v_k\}$ from the verifier.
2: **for** $i \in \{1, \ldots, k\}$ **do**
3: Open the commitments to u_{v_i} and u_v for all $v \in \mathrm{IN}(v_i)$.

Algorithm 3 Verifier of a solution for the puzzle $\mathcal{P}$ using sequential hash family sh and the DAG G of N vertices in topological order and in-degree d.

1: Receive the commitment c (supposedly for the hash labels $(u_1, u_2, \ldots, u_N)$) from the solver.
2: Randomly choose k nodes $v_1, \ldots, v_k$ from $[N] = V_G$ and send them to the solver.
3: **for** $i \in \{1, \ldots, k\}$ **do**
4: Verify the commitment openings of u_{v_i} and u_v for all $v \in \mathrm{IN}(v_i)$.
5: Verify $u_{v_i} = sh_{\mathcal{P}}(u_{v_{(1,i)}}, \ldots, u_{v_{(d,i)}})$ where $v_{(1,i)} \leq \cdots \leq v_{(d,i)}$ are the in-neighbors of v_i.

Construction 3.9 (Time-Lock Puzzle)**.** Given $n, N \in \mathbb{N}$ an explicit DAG G of N vertices and in-degree d, the time-lock puzzle $\Pi_{n,N}$ is as follows.

- The puzzle generator Gen outputs $\mathcal{P} \xleftarrow{\$} \mathbf{D}_n$.
- The puzzle solver Sol executes Algorithms 1 and 2, and the puzzle verifier Ver executes Algorithm 3.

3.4 Proving Theorem 3.7

To prove Theorem 3.7, we use the following depth-robust graphs in Construction 3.9.

Lemma 3.10 (Explicit Depth-Robust Graphs)**.** *For all constants $\beta < \alpha$ and all N there is an explicit family of (α, β)-depth robust graphs with N vertices and in-degree $O(\log^3 N)$.*

The proof of Lemma 3.10 is based on the ideas from [EGS75] and is presented in Section A. We now prove Theorem 3.7 by analyzing the properties of Construction 3.9 when instantiated by depth-robust graphs specified in Lemma 3.10.

Completeness. Clearly if the puzzle solver and the verifier follow Algorithms 1, 2, and 3, then the verifier accepts with probability one.

Running Times. The following lemma can be verified by inspecting Algorithms 1, 2, and 3, and the Merkle commitment algorithms of Section B. These bounds imply the bounds of Theorem 3.7 when substituting $d = O(\log^3 N)$ which is the degree of the DAG G we will use (according to Lemma 3.10).

Lemma 3.11. *The running time of the parties in Construction 3.9 is as follows.*

- **Generation.** *The puzzle generator* Gen *simply outputs a sample string from* $\mathbf{D}_n$*.*
- **Honest Solver.** *The solver, in its first round of action simply calls* $\mathbf{Adv}_1$ *and returns its output directly as the puzzle solution. To generate the hash labels, the solver* Sol *first constructs the DAG G in time $N \cdot \mathrm{polylog}(N)$ and then evaluates the sequential hash $sh(\cdot)$ N times for the vertices of G (over inputs of length $d \cdot n$ and outputs of length n). To commit to the hash labels, the solver evaluates the collision-resistant hash function $ch(\cdot)$ at most $N/2+N/4+\cdots \leq N$ times over inputs of length $2n$ and outputs of length n. Therefore, if calling $sh(\cdot)$ takes time t_{sh} and calling $ch(\cdot)$ takes time t_{ch}, the total running time of* Sol *will be $(t_{sh} + t_{ch} + \mathrm{polylog}\, N) \cdot N$.*
- **Verifier.** *To answer each of the k challenges asked by the verifier, the solver (now playing as a prover) needs to send $2 \cdot (d+1) \cdot \log N$ strings of length n (which are computed already) to the verifier. Also, for each of these k challenges, the verifier first constructs the relevant part of G in time $\mathrm{polylog}(N)$, and then it evaluates the sequential hash $sh(\cdot)$ once (in time t_{sh}) and evaluates $ch(\cdot)$ $2 \cdot (d+1) \cdot \log N$ times all in parallel time t_{ch} (see Algorithm 7).*

Parallel Soundness of Construction 3.9. The following lemma shows that by using an explicit (α, β)-depth robust graphs G for constants $\beta < \alpha$ from Lemma 3.10 in Construction 3.9, we can derive the soundness property stated in Theorem 3.7.

Lemma 3.12. *Suppose the DAG G used in Construction 3.9 is (α, β)-depth-robust for some constants $0 < \beta < \alpha < 1$. Then any malicious solver who is a circuit of size $\mathrm{poly}(n, N)$ and depth at most $\tau \cdot N \cdot \beta$ till returning the puzzle solution is able to make the verifier accept with at most $\mathrm{negl}(n)$ probability.*

We first prove Theorem 3.7 using Lemma 3.12 and then will prove Lemma 3.12.

Concluding Theorem 3.7. Since we have $k = \omega(\log n)$ Theorem 3.7 follows as a corollary from Lemmas 3.10, 3.11, and 3.12, because **(1)** it holds that $d = O(\log^3 N)$ and **(2)** for $k = \omega(\log n)$ and $\alpha = 1 - \Omega(1)$ it holds that $\alpha^k = \mathrm{negl}(n)$.

3.5 Proof of Lemma 3.12

Outline. Roughly speaking, for the committed labeling $u_1, \ldots, u_N$ by the adversary $\mathbf{Adv}$, we call a node $i \in [N]$ a good node if its hash label is indeed equal to the hash of the labels of its in-neighbors. If the number of good nodes is at most $\alpha \cdot N$, then the probability that the adversary can convince the verifier is at most $\alpha^k + \text{negl}(n)$. On the other hand, if the number of good nodes are more than $\alpha \cdot N$ then there should be path consisting of at least $\beta \cdot N$ many good nodes. The latter path, however, corresponds to a "chain" of queries, and the sequential property of $sh(\cdot)$ ensures that to generate such a chain the adversary must run in time proportional to the length of the chain.

More formally, the proof is by a reduction showing how to use an adversary that breaks η-soundness (i.e., can convince a verifier to accept a solution while working in "small" parallel time) to break the sequential soundness of sh (i.e., produce a "long" hash chain in "small" parallel time) as follows. Given an adversary $\mathbf{Adv}$ who breaks soundness of the time-lock puzzle scheme, we run $\mathbf{Adv}$ until it outputs the commitment c. We then execute $O(N)$ verification sessions using $O(N)$ copies of $\mathbf{Adv}$. In each of these verification sessions we emulate a verifier that queries different vertices of the DAG, with the hope that after this stage we gather the labeling of "many" vertices of the DAG. The main challenges of the proof of soundness lie in the analysis of this reduction. Since $\mathbf{Adv}$ succeeds in convincing the verifier to accept with non-negligible probability, at most a constant fraction of the extracted labels can be "bad" (i.e., don't correspond to a hash of their in-neighbors' labels). Thus, by the depth-robustness property of the graph G, there must be a long path of good vertices in G. This path is exactly the hash-chain we are looking for.

3.5.1 The Formal Proof

In the following we start by assuming (for sake of contradiction) that there is an adversary $\mathbf{Adv}_1$ who breaks the parallel-soundness of Construction 3.9 with probability at least $\varepsilon_1 \geq 1/\operatorname{poly}(n)$ (when using a $(\alpha, \beta)-$depth-robust graph of degree $O(\log^3 N)$), has circuit size $\operatorname{poly}(n, N)$ and depth $\tau \cdot N \cdot \beta$ till sending the puzzle solution. Then we will show how to turn this adversary into another adversary that either breaks the sequential property of $sh(\cdot)$ or the binding property of the commitment (which by Lemma B.1 implies breaking the collision resistance of ch).

Lemma 3.13 (Extracting a Chain)**.** *Suppose $\mathbf{Adv}_1$ is an adversary who convinces the verifier of Algorithm 3 with probability at least $\varepsilon_1 = \varepsilon_1' + \alpha^k$. Then there is an algorithm $\mathbf{Adv}_2$ (described in Algorithm 4) who executes $\ell = 2nN/\varepsilon_1'$ copies of $\mathbf{Adv}_1$, and then evaluates $sh(\cdot)$ $O(\ell \cdot k)$ times (and makes an additional $O(\ell \cdot k \cdot d)$ executions of the commitment scheme's verification algorithm) and $\Pr[E] \geq \varepsilon_1'/3$ where E is the event that: $\mathbf{Adv}_2$ breaks the binding of $\mathsf{Com}(\cdot)$ or its view has a chain of length βN in $sh(\cdot)$. Note that assuming $\varepsilon_1' > 1/\operatorname{poly}(n)$, the running time of $\mathbf{Adv}_2$ is only $\operatorname{poly}(n, N)$ times that of $\mathbf{Adv}_1$.*

Using Lemma 3.13 to Conclude Lemma 3.12. After obtaining the adversary $\mathbf{Adv}_2$ we can break the assumption that $sh(\cdot)$ is τ-sequential (which is a contradiction) as follows. The adversary $\mathbf{Adv}_2$ will simply forward the puzzle solution c of $\mathbf{Adv}_1$ (i.e., the Merkle-commitment string) as the label of the last node of the chain. Then, later on, suppose

$\mathbf{Adv}_2$ has access to a set of hash labels $u_1, \ldots, u_N$ such that there is a chain of length at least $\beta \cdot N$ planted in them. Then, by Lemma 3.13, $\mathbf{Adv}_2$ is able to find a path $\overrightarrow{\mathsf{PT}}$ of length at least $\beta \cdot N$ in G whose labels are all accepted during the verification phase. This means that the revealed labels along the paths that connect the nodes of $\overrightarrow{\mathsf{PT}}$ to the root of the Merkle-tree are all extracted successfully. $\mathbf{Adv}_2$ will simply return the labels of $\overrightarrow{\mathsf{PT}}$ followed by the labels of the nodes connecting the last node of $\overrightarrow{\mathsf{PT}}$ to the root (which is c). Therefore, $\mathbf{Adv}$ is able to win in the security game of Construction 3.1 with non-negligible probability by finding a chain of depth $\beta \cdot N$ in depth $< \tau \cdot \beta \cdot N$ till sending c. This violates the assumption that $sh(\cdot)$ is τ-sequential.

Algorithm 4 Either find a collision or extract a $\beta \cdot N$-chain, by using oracle access to an adversary $\mathbf{Adv}_1$ who convinces the interactive verifier with probability $\varepsilon_1 = \varepsilon_1' + \alpha^k$.

1: Run $\mathbf{Adv}_1$ over a random puzzle $\mathcal{P} \xleftarrow{\$} \{0,1\}^n$ and random coins rand_2 and receive some commitment $c \in \{0,1\}^n$. At this moment save the state of the adversary $\mathbf{Adv}_1$ since we are going to execute $\mathbf{Adv}_1$ in many "different branches" *in parallel* by feeding many different challenge messages to $\mathbf{Adv}_1$ and asking it to open those commitments. We cannot afford to use standard "rewinding" since we want to keep the depth of the circuit of $\mathbf{Adv}_2$ close to that of $\mathbf{Adv}_1$.
2: Let $\delta = \varepsilon_1'/(2N)$ and $\ell = n/\delta$.
3: For all $j \in [\ell]$ choose a random subset $S_j \subset [N]$ of size $|S_j| = k$ (possibly with repetitions).
4: **for** all $j \in [\ell]$ **do**
5: Ask $\mathbf{Adv}_1$ to open the nodes in the challenge set S_j with respect to the commitment c.
6: After receiving decommitments for $(d+1) \cdot k \cdot \ell$ many nodes in $[N]$ (possibly with repetitions), run Merkle verification algorithm for all of them.
7: If two different openings for the same label are both accepted by the verifier, return a collision in ch according to Lemma B.1.
8: Let H be an empty graph with the same vertex set as that of G. For every challenge node v (from the set of all k challenges) that passes the Merkle verification, let $v_1, \ldots, v_d$ be the in-neighbors of v_i in G. Add the edges $(v_1, v), \ldots, (v_d, v)$ to the graph H.
9: Search for a the longest path $\overrightarrow{\mathsf{PT}}$ in the graph H and return the labels of the nodes of $\overrightarrow{\mathsf{PT}}$ continued with the $\log(N)$ labels of the nodes of the Merkle-tree decommitment connecting the last node of $\overrightarrow{\mathsf{PT}}$ to the root.

Proof of Lemma 3.13. The running time of $\mathbf{Adv}_2$ is at most $O(Nn/\varepsilon_1') = \mathrm{poly}(N, n)$ times more that that of $\mathbf{Adv}_2$ (without considering the final verification). So we only need to prove the existence of the long chain in $sh(\cdot)$ in the view of $\mathbf{Adv}_2$, assuming that $\mathbf{Adv}_2$ did not break the binding property of $\mathsf{Com}(\cdot)$.

Since $\mathbf{Adv}_1$ succeeds in convincing the verifier with probability at least ε_1, by an averaging argument, with probability at least $\varepsilon_1'/2$ over the choices of the puzzle $\mathcal{P}$ and the randomness of $\mathbf{Adv}_1$ (i.e., rand_2), $\mathbf{Adv}_1$ will have at least a chance of $\alpha^k + \varepsilon_1'/2$ (over the randomness of the challenge message) to convince the verifier. In the following we assume that the sampled $\mathcal{P}$ and rand_2 in Step 1 of Algorithm 4 have this property. We will

show that in this case, $\mathbf{Adv}_2$ succeeds in finding a (long enough) chain with probability at least 9/10, leading to a total probability of success at least $(\varepsilon'_1/2) \cdot (9/10) > \varepsilon'_1/3$.

Suppose W is the event that $\mathbf{Adv}_1$ succeeds answering a random challenge set S of k nodes. Call a node $i \in [N]$ a *heavy* node if $\Pr[i \in S \text{ and } W] \geq \delta = \varepsilon'_1/(2N)$ for a random challenge set S of size k. Call a node $i \in [N]$ *light* if it is not heavy. Let $\mathcal{HV}$ be the set of heavy nodes and $\mathcal{LT}$ be the set of light nodes. We claim that the number of heavy nodes is at least $\alpha \cdot N$. Otherwise $\mathbf{Adv}_1$ is able to answer a random challenge $S \subset [N]$ of k nodes correctly only with probability:

$$\begin{aligned}\Pr_S[W] &\leq \Pr_S[W \text{and } S \subset \mathcal{HV}] + \Pr_S[W \text{and } S \cap \mathcal{LT} \neq \varnothing] \\ &< \alpha^k + \sum_{i \in \mathcal{LT}} \Pr[W \text{and } i \in S] \leq \alpha^k + N \cdot \delta\end{aligned}$$

which is at most $\alpha^k + \varepsilon'_1/2$ as opposed to our assumption. On the other hand, since $\mathbf{Adv}_2$ chooses ℓ random challenge sets S_j of size k, for every heavy node $i \in [N]$, the probability that for some $j \in [\ell]$ $\mathbf{Adv}_1$ can successfully decommit to all of the nodes in S_j while it includes $i \in S_j$ is at least $1 - (1-\delta)^\ell > 1 - e^{-n} > 1 - 2^{-n}$. Therefore, by a union bound, with probability at least $1 - 2^{-n} \cdot N > 1 - 2^{-\Omega(n)}$ for every heavy node v, the adversary will decommit successfully (at some point) into some hash label for v and also some hash labels for the in-neighbors of v. When the latter holds we call v a good node, and call the (successfully opened) hash labels of v and its in-neighbors some *extracted* hash labels (note that potentially we might extract different hash labels for v in different branches of executing $\mathbf{Adv}_1$ over some challenge set S).

Note that we can safely assume that for all $i \in [N]$ all the extracted hash labels for the node $i \in [N]$ (either extracted as the label of a sampled node in a challenge set, or as the label of an in-neighbor of a sampled node) are identical. The reason is that otherwise we would have broken the binding property of $\mathsf{Com}(\cdot)$.

Therefore with probability at least $1 - 2^{-\Omega(n)}$, we get at least $\alpha \cdot N$ good nodes (with some extracted hash label for them and also for their in-neighbors) and also it holds that all the extracted hash labels are consistent (i.e., equal for the same node). By the (α, β)-depth-robustness of G, the set of good nodes will have an induced path $\overrightarrow{\mathsf{PT}}$ of size at least $\beta \cdot N$. For every node $v \in \overrightarrow{\mathsf{PT}}$, let w_v be equal to the string $(u_{v_1}, \ldots, u_{v_d})$ where $v_1 \leq \cdots \leq v_d$ are the in-neighbors of v. Since $\overrightarrow{\mathsf{PT}}$ includes only good nodes that have passed the verification of the verifier, it holds that $sh_{\mathcal{P}}(w_v) = u_v$ where u_v is the extracted hash label of v. Hence, the sequence $(w_v)_{v \in \overrightarrow{\mathsf{PT}}}$ makes a chain of size $\beta \cdot N$. Thus, conditioned on the quality of the sampled $(\mathcal{P}, \mathsf{rand}_2)$ as discussed above, with probability $(1 - 2^{-\Omega(n)}) > 9/10$ the adversary $\mathbf{Adv}_2$ gets a chain of size at least $\beta \cdot N + \log N > \beta \cdot N$ (including the nodes connecting $\overrightarrow{\mathsf{PT}}$ to the root of the Merkle tree). □

4 Time-Lock Puzzles in the Random Oracle Model

Our construction for Theorem 3.7 makes *black-box* use of sequential hash functions and collision-resistant hash functions. Both primitives can be easily constructed with unconditional security in the random oracle model (when time-complexity is measured by the number of oracle queries rather than computational effort) for every index distribution

with super-logarithmic min-entropy. As a consequence, in this model we get unconditionally secure time-lock puzzles and time stamping schemes for any puzzle and/or document distribution with super-logarithmic min-entropy. Moreover, we can obtain time stamping protocols for any document distribution of super-logarithmic min-entropy.

More formally, in the random oracle model we model $\mathbf{Time}(\cdot)$ as the number of oracle queries and $\mathbf{ParTime}(\cdot)$ as the number of *rounds* of oracle queries. However, in the random oracle model (ROM), we can get stronger results: our construction is noninteractive and unconditionally secure. We prove the following theorem.

Theorem 4.1 (Main Result in ROM)**.** *Theorem 3.7 holds relative to any random oracle mapping $\{0,1\}^*$ to $\{0,1\}^n$ unconditionally for any distribution $\mathbf{D}$ of the messages with min-entropy $\omega(\log n)$. Moreover the verification of the scheme is noninteractive (i.e., no challenge is sent from the verifier to the puzzle solver).*

We prove Theorem 4.1 by proving the following lemma, and using Fiat-Shamir transformation [FS86b] in the random oracle model.

Lemma 4.2. *Suppose $\mathbf{D} = \mathbf{D}_n$ is a distribution with min-entropy at least $\omega(\log n)$. Then relative to a random oracle from $\{0,1\}^*$ to $\{0,1\}^n$, there are efficient hash functions $\mathcal{CH}, \mathcal{SH}$ such that $\mathcal{CH}$ is collision resistant according to Definition 3.3 and $\mathcal{SH}$ is sequential according to Definition 3.2.*

4.1 Unconditional Interactive Construction in the ROM

In this subsection we prove Lemma 4.2.

Implementing Hash Functions $sh(\cdot)$ and $ch(\cdot)$. For puzzle distributions of super-logarithmic mi-entropy, we pad the puzzle to become of size $|\mathcal{P}| \geq n$, and we use a random oracle of the same security parameter $\mathcal{O}\colon \{0,1\}^* \mapsto \{0,1\}^n$ as follows. The sequential hash function indexed by the puzzle $\mathcal{P}$, denoted $sh_{\mathcal{P}}(\cdot)$ is defined as follows: $sh_{\mathcal{P}}(x) = \mathcal{O}(\mathcal{P}, 0, x)$. To get the collision resistant hash function, we use $ch_{\mathcal{P}}(x) = \mathcal{O}(\mathcal{P}, 1, x)$ to map $2n$ bit strings to n bits.

Note that since our adversaries in this work always ask $\text{poly}(n) < 2^{o(n)}$ queries to $\mathcal{O}$, at the time $\mathcal{P} \xleftarrow{\$} \mathbf{D}$ is sampled, (except with $\text{negl}(n)$ probability) no query with prefix $\mathcal{P}$ is asked to $\mathcal{O}$ and therefore we can safely assume that both oracles $sh(\cdot)$ and $ch(\cdot)$ are completely random even conditioned on the $\text{poly}(n)$ preprocessing queries asked by the adversary. Therefore, to prove Lemma 4.2 we just need to prove that random oracles can be *directly* used to get sequential and collision-resistant hash functions.

First we recall the well-known fact that a random oracle is collision-resistant.

Lemma 4.3 (Random Oracle is Collision-Resistant)**.** *Suppose $ch(\cdot)$ is a random oracle from $\{0,1\}^*$ to $\{0,1\}^n$. Then any adversary who asks at most $2^{o(n)}$ queries to $ch(\cdot)$ is able to find a collision $x \neq x', ch(x) = ch(x')$ with probability at most $2^{-\Omega(n)}$.*

Second we prove that the $sh(\cdot)$ is indeed a sequential function.

Definition 4.4. A *chain* of length r relative to the oracle $sh(\cdot)$ is a sequence of strings $w_0, w_1 \ldots, w_r$ such that $sh(w_{i-1})$ is a (contiguous) substring of w_i for all $i \in [r]$.

Lemma 4.5 (Random Oracle is Sequential)**.** *Suppose $sh(\cdot)$ mapping $\{0,1\}^*$ to $\{0,1\}^n$. Then, no adversary with oracle access to $sh(\cdot)$ can break the η-sequential of $sh(\cdot)$ according to Definition 3.2 for any $\eta < 1$ and any $N = 2^{o(n)}$.*

Proof. We use the following two claims to prove the lemma.

Claim 4.6. *Suppose A is an oracle algorithm who asks $2^{o(n)}$ queries of length at most $2^{o(n)}$ to $sh(\cdot)$ in $r-1$ adaptive rounds. The probability that A's queries include a chain of length r is at most $2^{-\Omega(n)}$.*

Claim 4.7. *Suppose $sh\colon \{0,1\}^* \mapsto \{0,1\}^n$ is distributed uniformly at random conditioned on a set of $2^{o(n)}$ input/output pairs S. Namely, S contains pairs (x,y) (where $|y| = n$) for which we know $sh(x) = y$ and $sh(\cdot)$ is randomly mapped to $\{0,1\}^n$ otherwise. Suppose T is another set of size at most $2^{o(n)}$ containing some possible outputs (i.e., $|y| = n$ if $y \in T$). Suppose a computationally unbounded adversary A who knows the set S asks at most $2^{o(n)}$ queries to $sh(\cdot)$ and wins the game if it could output some (x,y) such that $y \in T$ and $(x,y) \notin S$. Then the probability of A winning is at most $2^{-\Omega(n)}$.*

We first prove Lemma 4.5 using Claims 4.6 and 4.7, and then will prove these claims.

Proving Lemma 4.5. Suppose **Adv** is an adversary with oracle access to $sh(\cdot)$ who can break the sequential property of $sh(\cdot)$ according to Definition 3.2. Let S be the set of oracle query/answer pairs known to **Adv** till it sends y as the last node the chain (in Construction 3.1). By Claim 4.6, except with probability $2^{-\Omega(n)}$, S has no chain of length at least N. Since S does not have any change of length at least N, if **Adv**, at the end of the game (of Construction 3.1) outputs any chain of length at least N ending at y, it implies that **Adv** has managed to find some (x', y') such that:

1. $sh(x') = y'$.
2. $(x', y') \notin S$.
3. y' is either equal to y or $(x, y') \in S$ for some x.

Since **Adv** asks only $2^{o(n)}$ queries, by defining $T = \{y\} \cup \{y' \mid (x', y) \in S\}$ and applying Claim 4.7 we conclude that the adversary can not win the security game of Construction 3.1 except with probability $2^{-\Omega(n)}$.

Proving Claim 4.6. Suppose A has asked the queries $x_1, \ldots, x_\ell$ so far and is about to ask a new round of queries $y_1 \ldots, y_q$. We claim that with probability $1 - 2^{-\Omega(n)}$ the new queries $y_1, \ldots, y_q$ can only be the *last* nodes in any new chain in the view of A. Since the total number of queries of A is $2^{o(n)}$ we only need to prove the latter claim for one of the new queries, $y_i \in \{y_1, \ldots, y_q\}$ and the claim follows by a union bound. Let $X = \{x_1, \ldots, x_\ell, y_1, \ldots, y_{i-1}, y_{i+1}, \ldots, y_q\}$ be all the queries of A asked (or about to be asked) so far other than y_i. The total number of (contiguous) substrings of length n among all the elements of X is at most $2^{o(n)} \cdot 2^{o(n)} \leq 2^{o(n)}$ (because all those queries are of length $2^{o(n)}$ any substring is determined by choosing two points in the string). Since the answer to $sh(y_i)$ is a random string of length n, this answer will be different from all the contiguous substrings of the elements of X with probability at least $1 - 2^{-n}2^{o(n)} = 1 - 2^{-\Omega(n)}$.

Therefore, with probability at least $1 - 2^{-\Omega(n)}$ the length of the longest chain in the view of A can increase in each rounds of adaptive queries of A only by one (except with

probability $2^{-\Omega(n)}$). Thus (by induction) the probability that A can output a chain of length r in $r-1$ rounds of queries is at most $2^{o(n)} \cdot 2^{-\Omega(n)} = 2^{-\Omega(n)}$.

Proving Claim 4.7. The adversary A has to find some x' such that $sh(x') \in T$ while the answer to x' is not previously fixed by S. Any new query x' (i.e., that there is no y such that $(x', y) \notin S$) asked by A will be mapped by $sh(\cdot)$ to a point in T with probability at most $|T| \cdot 2^{-n} = 2^{-\Omega(n)}$. Since the total number of queries asked by A is at most $2^{o(n)}$, the chance of A winning will be bounded by $2^{o(n)} \cdot 2^{-\Omega(n)} \leq 2^{-\Omega(n)}$. □

Lemmas 4.5 and 4.3 together Lemma 4.2, which in turn proves our Theorem 4.1 for the case of interactive verification.

4.2 Noninteractive Verification Using the Fiat-Shamir Transformation

In the random oracle model, by using the Fiat-Shamir transformation of Lemma 4.8 we can remove the challenge message of the verifier in Construction 3.9 to make the verification completely noninteractive and obtain Theorem 4.1.

Similarly to the previous subsection, here also we assume w.l.o.g. that the oracle $\mathcal{O}$ is completely random at the time the puzzle $\mathcal{P} \xleftarrow{\$} \mathbf{D}$ is chosen and given to the adversary. This is because we can pad all queries to $\mathcal{O}$ with $(\mathcal{P}, 3)$ and apply a similar argument to that of the sequential and collision hash functions $\mathcal{SH}, \mathcal{CH}$ above based on the $\omega(\log n)$ min-entropy of $\mathbf{D}$.

The following transformation is due to Fiat and Shamir [FS86a] and shows how to remove interaction from public-coin protocols in the random oracle model. For completeness, here we prove a special case in which there is only four messages exchanged, while we are interested in (almost) preserving the adaptivity of the adversary.

Lemma 4.8 (Fiat-Shamir Transformation)**.** *Suppose (P, V) is two party protocol using a random oracle $\mathcal{O}$ of output length n as follows.*

- *The protocol has only 4 messages: v_1, p_1, v_2, p_2 where v_1 and v_2 are public-coin messages of V and V does not use any private randomness to make her final decision.*
- *We have $|v_1| = n$ and $|v_2| \leq \ell \cdot n$.*
- *The verifier rejects its interaction with probability $1 - \varepsilon$ against* any *prover $\widehat{P}$ who:*
  1. *$\widehat{P}$ asks a total of q queries to the random oracle $\mathcal{O}$.*
  2. *$\widehat{P}$ has at most r rounds of adaptivity in its queries.*

Suppose (P', V') is a two-message *protocol defined based on (P, V) as follows: The second message v_2 of V is removed from the protocol and instead the oracle answers to the following queries are used $\mathcal{O}(v_1, p_1, 1), \mathcal{O}(v_1, p_1, 2), \ldots, \mathcal{O}(v_1, p_1, \ell)$. (Note that the number of obtained random bits this way will be exactly $n \cdot \ell = |v_2|$.) This randomness is used by the parties and the two messages of the prover (p_1, p_2) are sent together. Then it holds that any adversary $\widehat{P'}$ who interacts with V' and asks $q' = q/\ell$ number of queries to $\mathcal{O}$ and has at most $r' = r - 1$ rounds of adaptivity is able to convince V' with probability at most $\varepsilon' = \varepsilon \cdot q$.*

Proof. For sake of contradiction suppose $\widehat{P'}$ is an adversary who interacts with V', asks at most $q' = q/\ell$ queries to $\mathcal{O}$, has at most $r' = r - 1$ rounds of adaptivity, and is able to convince V' with probability $\varepsilon' > \varepsilon \cdot q$. We show how to get an adversary $\widehat{P}$ who interacts with V, asks at most $q' \cdot \ell = q$ oracle queries, has at most r rounds of adaptivity and is able to make V accept with probability at least $\varepsilon'/q = \varepsilon$, which is a contradiction.

First we modify $\widehat{P'}$ as follows.

- $\widehat{P'}$ never asks any query twice.
- $\widehat{P'}$ always asks the queries $\mathcal{O}(v_1, p_1, 1), \mathcal{O}(v_1, p_1, 2), \ldots, \mathcal{O}(v_1, p_1, \ell)$ before sending (p_1, p_2) in one round of adaptivity, if not asked already.
- If $\widehat{P'}$ makes any query of the form $\mathcal{O}(v'_1, p'_1, j)$ for any p'_1 and $j \in [\ell]$, it also asks all the other queries $\{\mathcal{O}(v'_1, p'_1, i) \mid i \in [\ell], i \neq j\}$ in the same round of adaptivity. (Note that $\mathcal{O}(v'_1, p'_1, j)$ might be asked when v_1 is not known or p_1 is not decided yet).

The above changes might increase the total queries of $\widehat{P'}$ by a factor of ℓ and might add one round of adaptive queries to $\widehat{P'}$. The adversary $\widehat{P}$ works as follows:

1. When $\widehat{P'}$ asks for v_1, receive v_1 from V and forward it to $\widehat{P'}$.
2. Choose $i \xleftarrow{\$} [q]$ at random.
3. Emulate the execution of $\widehat{P'}$ by preserving the adaptivity of the queries as follows.
   (a) When $\widehat{P'}$ asks its i-th query $\mathcal{O}(y)$, if y is *not* of the form (v_1, p'_1, j) for some $(p'_1, j \in [\ell])$ then abort. Otherwise do the following:
      i. Send p'_1 back to V as the first message of the prover and receive v_2.
      ii. Use v_2 to answer the query $\mathcal{O}(y)$ as well as all of $\{\mathcal{O}(v'_1, p'_1, i) \mid i \in [\ell], i \neq j\}$ that are going to be asked in the same round of adaptivity.
   (b) When the emulation of $\widehat{P'}$ is finished, suppose (p_1, p_2) is the generated message. If p_1 is different from p'_1 (which was part of y) abort, otherwise send p_2 to V.

We claim that with probability $1/q$ over the random choice of $i \xleftarrow{\$} [q]$ the game above is a perfect emulation of the game in which $\widehat{P'}$ interacts with V'. The reason is that with probability $1/q$ the emulating adversary $\widehat{P}$ guesses the actual query $\mathcal{O}(v_1, p_1, 1)$ of $\widehat{P'}$ correctly, in which case since v_2 is completely random, we get a perfect emulation of the game of interaction between $\widehat{P'}$ and V' in the random oracle model (where a particular oracle query is answered using fresh randomness). Thus the emulation above leads to the accept of V with probability at least $\varepsilon' \cdot 1/q = \varepsilon$, while the number of rounds of oracle queries asked by $\widehat{P}$ is at most r. □

5 Open Questions

Space Complexity of the Solver. In our construction of time stamping and time-lock puzzles for time N, the solver keeps the hash labels of a graph of N vertices. Is there any other solution that uses $o(N)$ storage? Or is there any inherent reason that $\Omega(N)$ storage is necessary?

Necessity of Depth-Robust Graphs. The efficiency and security of our construction is tightly tied to the parameters of depth-robust graph constructions: graphs with lower degree give more efficient solutions, while graphs with higher robustness (the lower bound on the length of the longest path remaining after some of the vertices are removed) give us puzzles with smaller adversarial advantage. An interesting open question is whether the converse also holds: do time-lock puzzles with better parameters also imply the existence of depth-robust graphs with better parameters?

Using Time-Lock Puzzles to Achieve Fairness. One motivation for studying time-lock puzzles, and timed assumptions in general, is that they can be used to solve problems that are provably impossible in the standard model. For example, Boneh and Naor showed that *timed commitments* (ones that can be opened without the key in certain amount of time, but not faster) can be used to perform fair coin flipping [BN00], which was previously shown by Cleve to be impossible in the standard model [Cle86]. Boneh and Naor construct timed commitments based on the same assumption used by Rivest et al. [RSW96] to construct time-lock puzzles (the inherent sequentiality of exponentiation). Briefly, their coin-flipping protocol is as follows: Alice chooses a random bit b_a and sends $\mathsf{Com}(b_a)$ to Bob, where Com is a timed commitment. Bob chooses a random bit b_b and sends b_b to Alice. Alice verifies that b_b arrived fast enough (so Bob could not have forced open her commitment), and then opens the commitment. The result of the coin-flip is $b_a \oplus b_b$. The "timed" part of the commitment is used if Alice aborts before opening her commitment. In that case, Bob can spend a moderate amount of time to force-open the commitment and recover b_a without Alice's help. For this protocol to work, the time it takes to force open a commitment must be more than the maximum network latency. On the other hand, for efficiency, the time it takes to honestly open a commitment should be short as possible.

Mahmoody et al. [MMV11] showed that, in the random oracle model, timed commitments with a large time-gap (between the forced opening and an honest opening) cannot be constructed, hence we cannot use black-box constructions to implement them in the standard model. A natural approach would be to replace the timed commitment in the coin-flipping protocol with our proof of work: Alice sends a puzzle P to Bob, Bob sends a random bit-vector b_2, then Alice sends a bit-vector b_1 and proves it is the solution to the puzzle P. The result is taken to be $b_1 \cdot b_2$. This would be less efficient than using timed commitments, since Alice has to solve the puzzle even in an honest execution, but she can do the work offline, leaving the online phase of the protocol efficient. Unfortunately, this protocol is insecure: the soundness of our proof of work ensures that Alice spends time proportional to the honest solver, but still she may convince Bob to accept an incorrect solution b_1. An interesting open question is whether fair secure computation (and in particular fair coin flipping) is possible based on black-box sequentiality assumptions.

Acknowledgments. We thank the anonymous CRYPTO 2011 reviewers of our previous paper [MMV11], whose comments led us to investigate the topics in this paper. We also thank Moni Naor, Leslie Valiant, and Avi Wigderson for pointers to relevant work and Yuval Ishai for helpful discussions. Finally, we thank the anonymous ITCS 2013 reviewers for their helpful comments.

References

[ABSW01] A. Ansper, A. Buldas, M. Saarepera, and J. Willemson, *Improving the availability of time-stamping services*, Information Security and Privacy, 6th Australasian Conference, ACISP, 2001, pp. 360–375. 4

[BdM91] J. Benaloh and M. de Mare, *Efficient broadcast time-stamping*, Tech. Report 1, Clarkson University Department of Mathematics and Computer Science, August 1991. 4

[BdM93] J. C. Benaloh and M. de Mare, *One-way accumulators: A decentralized alternative to digital sinatures (extended abstract)*, EUROCRYPT, 1993, pp. 274–285. 4

[BG08] Boaz Barak and Oded Goldreich, *Universal arguments and their applications*, SIAM J. Comput. **38** (2008), no. 5, 1661–1694. 2

[BHS92] D. Bayer, S. Haber, and W. S. Stornetta, *Improving the efficiency and reliability of digital time-stamping*, Sequences II: Methods in Communication, Security and Computer Science (R. M. Capocelli et al., ed.), Springer-Verlag, 1992, pp. 329–334. 4

[BL98] A. Buldas and P. Laud, *New linking schemes for digital time-stamping*, Information Security and Cryptology, 1998, pp. 3–13. 4

[BLLV98] A. Buldas, P. Laud, H. Lipmaa, and J. Villemson, *Time-stamping with binary linking schemes*, CRYPTO, 1998, pp. 486–501. 4

[BLS00] A. Buldas, H. Lipmaa, and B. Schoenmakers, *Optimally efficient accountable time-stamping*, Public Key Cryptography, 2000, pp. 293–305. 4

[BN00] Dan Boneh and Moni Naor, *Timed commitments*, CRYPTO (Mihir Bellare, ed.), Lecture Notes in Computer Science, vol. 1880, Springer, 2000, pp. 236–254. 22

[BR93] Mihir Bellare and Phillip Rogaway, *Random Oracles are Practical: A Paradigm for Designing Efficient Protocols*, ACM Conference on Computer and Communications Security, 1993, pp. 62–73. 4

[CGH04] Ran Canetti, Oded Goldreich, and Shai Halevi, *The random oracle methodology, revisited*, J. ACM **51** (2004), no. 4, 557–594. 4

[Cle86] Richard Cleve, *Limits on the security of coin flips when half the processors are faulty (extended abstract)*, STOC, ACM, 1986, pp. 364–369. 22

[CLSY93] Jin-yi Cai, Richard J. Lipton, Robert Sedgewick, and Andrew Chi-Chih Yao, *Towards uncheatable benchmarks*, Structure in Complexity Theory Conference, 1993, pp. 2–11. 1, 2, 3

[DGN03] Cynthia Dwork, Andrew Goldberg, and Moni Naor, *On memory-bound functions for fighting spam*, CRYPTO (Dan Boneh, ed.), Lecture Notes in Computer Science, vol. 2729, Springer, 2003, pp. 426–444. 3

[DN92] Cynthia Dwork and Moni Naor, *Pricing via processing or combatting junk mail*, Crypto '92, 1992, LNCS No. 740, pp. 139–147. 3, 5

[DNW05] Cynthia Dwork, Moni Naor, and Hoeteck Wee, *Pebbling and proofs of work*, CRYPTO (Victor Shoup, ed.), Lecture Notes in Computer Science, vol. 3621, Springer, 2005, pp. 37–54. 3, 5

[DTT10] Anindya De, Luca Trevisan, and Madhur Tulsiani, *Time space tradeoffs for attacks against one-way functions and PRGs*, Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings (Tal Rabin, ed.), Lecture Notes in Computer Science, vol. 6223, Springer, 2010, pp. 649–665. 1

[EGS75] Paul Erdős, Ronald L. Graham, and Endre Szemerédi, *On sparse graphs with dense long paths*, Computers & Mathematics with Applications **1** (1975), 365–369. 4, 10, 14, 26

[FN99] Fiat and Naor, *Rigorous time/space trade-offs for inverting functions*, SICOMP: SIAM Journal on Computing **29** (1999). 1

[FS86a] Fiat and Shamir, *How to prove yourself: Practical solutions to identification and signature problems*, CRYPTO: Proceedings of Crypto, 1986. 4, 20

[FS86b] A. Fiat and A. Shamir, *How to Prove Yourself: Practical Solutions to Identification and Signature Problems*, Crypto '86, 1986, LNCS No. 263, pp. 186–194. 18

[GT03] Shafi Goldwasser and Yael Tauman, *On the (in)security of the fiat-shamir paradigm*, Proc. 44th FOCS, IEEE, 2003. 4

[Hel80] Martin E. Hellman, *A cryptanalytic time-memory trade-off*, IEEE Transactions on Information Theory **26** (1980), no. 4, 401–406. 1

[HS91] Stuart Haber and W. Scott Stornetta, *How to time-stamp a digital document*, J. Cryptology **3** (1991), no. 2, 99–111. 4

[JM10] Y.I. Jerschow and M. Mauve, *Offline submission with rsa time-lock puzzles*, Computer and Information Technology (CIT), 2010 IEEE 10th International Conference on, 29 2010-july 1 2010, pp. 1058–1064. 1

[JM11] Yves Igor Jerschow and Martin Mauve, *Non-parallelizable and non-interactive client puzzles from modular square roots*, ARES 2011: Proceedings of the 6th International Conference on Availability, Reliability and Security, August 2011. 6

[KC10] Ghassan Karame and Srdjan Capkun, *Low-cost client puzzles based on modular exponentiation*, ESORICS (Dimitris Gritzalis, Bart Preneel, and Marianthi Theoharidou, eds.), Lecture Notes in Computer Science, vol. 6345, Springer, 2010, pp. 679–697. 6

[Kil92] Joe Kilian, *A note on efficient zero-knowledge proofs and arguments (extended abstract)*, STOC, ACM, 1992, pp. 723–732. 2

[LLL82] Lenstra, Lenstra, and Lovasz, *Factoring polynomials with rational coefficients*, MATHANN: Mathematische Annalen **261** (1982). 6

[Mic00] Silvio Micali, *Computationally sound proofs*, SIAM J. Comput. **30** (2000), no. 4, 1253–1298. 2

[MMV11] Mohammad Mahmoody, Tal Moran, and Salil P. Vadhan, *Time-lock puzzles in the random oracle model*, CRYPTO (Phillip Rogaway, ed.), Lecture Notes in Computer Science, vol. 6841, Springer, 2011, pp. 39–50. 3, 6, 22

[MSTS09] Tal Moran, Ronen Shaltiel, and Amnon Ta-Shma, *Non-interactive timestamping in the bounded storage model*, J. Cryptology **22** (2009), no. 2, 189–226. 5

[PR80] Wolfgang J. Paul and Rüdiger Reischuk, *On alternation ii. a graph theoretic approach to determinism versus nondeterminism*, Acta Inf. **14** (1980), 391–403. 4

[PTC77] Wolfgang J. Paul, Robert Endre Tarjan, and James R. Celoni, *Space bounds for a game on graphs*, Mathematical Systems Theory **10** (1977), 239–251. 5

[RSW96] Ronald L. Rivest, Adi Shamir, and David A. Wagner, *Time-lock puzzles and timed-release crypto*, Tech. Report MIT/LCS/TR-684, MIT, February 1996. 1, 2, 6, 7, 22

[RTV04] Omer Reingold, Luca Trevisan, and Salil P. Vadhan, *Notions of reducibility between cryptographic primitives.*, Theory of Cryptography, First Theory of Cryptography Conference, TCC 2004, Lecture Notes in Computer Science, vol. 2951, Springer, 2004, pp. 1–20. 3

[RV10] Guy N. Rothblum and Salil P. Vadhan, *Are PCPs inherent in efficient arguments?*, Computational Complexity **19** (2010), no. 2, 265–304. 2

[RVW00] Reingold, Vadhan, and Wigderson, *Entropy waves, the zig-zag graph product, and new constant-degree expanders and extractors*, FOCS: IEEE Symposium on Foundations of Computer Science (FOCS), 2000. 26

[RVW01] ———, *Entropy waves, the zig-zag graph product, and new constant-degree expanders and extractors*, ECCCTR: Electronic Colloquium on Computational Complexity, technical reports, 2001. 26

[Sch82] Georg Schnitger, *A family of graphs with expensive depth reduction*, Theor. Comput. Sci. **18** (1982), 89–93. 4

[Sch83] ———, *On depth-reduction and grates*, FOCS, IEEE, 1983, pp. 323–328. 4

[TBFN07] Suratose Tritilanunt, Colin Boyd, Ernest Foo, and Juan Manuel González Nieto, *Toward non-parallelizable client puzzles*, CANS (Feng Bao, San Ling, Tatsuaki Okamoto, Huaxiong Wang, and Chaoping Xing, eds.), Lecture Notes in Computer Science, vol. 4856, Springer, 2007, pp. 247–264. 6

[Val77] Leslie G. Valiant, *Graph-theoretic arguments in low-level complexity*, MFCS (Jozef Gruska, ed.), Lecture Notes in Computer Science, vol. 53, Springer, 1977, pp. 162–176. 4

A Explicit Constructions of Depth Robust Graphs

In this section we prove Lemma 3.10 by showing how to obtain explicit (α, β)-depth robust graphs for constants $\alpha, \beta < 1$ that can be arbitrarily close to 1. Erdős, Graham and Szemerédi [EGS75] constructed (α, β)-depth robust graphs for some constants $0 < \beta < \alpha < 1$ based on a recursive use of constant-degree expanders. This construction can be made explicit using any explicit family of such expanders.

Theorem A.1 ([EGS75]). *There exists an explicit family $\{G_N\}$ of DAGs with N vertices and in-degree $d = O(\log N)$ that is (α, β)-depth-robust for some* constants $0 < \beta < \alpha < 1$.

Using the proof of [EGS75] one can obtain, e.g., $\alpha = 99/100, \beta = 1/100$, but by minor modifications to the construction of [EGS75] one can obtain constants (α, β) that are arbitrarily close to 1 at the cost of larger degrees $\log^2 N$. For sake of completeness, in Section A we describe this construction. Our construction follows that of [EGS75] closely, with the difference that we use denser expanding graphs in the recursive construction.

Definition A.2 (Expanding Graphs)**.** A bipartite graph $G = (V_1, V_2, E)$, $|V_1| = |V_2| = M$ is *A-expanding* if for every $S_1 \subseteq V_1$ and $S_2 \subseteq V_2$ such that $|S_1| = |S_2| = \lceil M/A \rceil$ there is an edge from S_1 to S_2.

We use explicit constructions of A-expanding graphs of [RVW00, RVW01].

Construction A.3. For simplicity suppose the number of vertices of our graph G is a power of two $N = 2^t$, and let $\gamma = \varepsilon / \log N = \varepsilon / t$ for arbitrarily small constant ε. We use the following recursive construction to get $G = G_t$. Let G_1 be a two-vertex graph with an edge between them. Informally, G_{i+1} consists of two identical copies of G_i connected with the edges of a bipartite $(1/\gamma)$-expanding graph. Formally, $G_{i+1} = ((L_{i+1}, R_{i+1}), E_{i+1})$, where $L_{i+1} = \{1, \ldots, 2^i\}$, $R_{i+1} = \{2^i + 1, \ldots, 2^{i+1}\}$ and the edges are

$$E_{i+1} := E_i \cup \left\{(u + 2^i, v + 2^i) \mid (u, v) \in E_i\right\} \cup E'_{i+1} \ ,$$

where $G' = ((L, R), E'_{i+1})$ is an explicit $(1/\gamma)$-expanding graph for $L = \{1, 2, \ldots, 2^i\}$ and $R = \{2^i + 1, \ldots, 2^{i+1}\}$.

Degrees and Explicitness. The explicit expanding graphs we used in Construction A.3 to connect the two copies of G_i are all of in-degree $\widetilde{O}(\log N) < \log^2 N$ for large enough N. Since the depth of the recursion is $\lceil \log N \rceil$, the total in-degree of the final graph is at most $\widetilde{O}(\log^2 N) < \log^3 N$ for large enough N. The explicitness of our constructed graph follows from the explicitness of the expanding graphs used. In particular, given any node $v \in [N] = [2^t]$ let $v - 1 = (b_t, \ldots, b_1)$ where b_i is the i-th bit of the binary representation of $v - 1$. To know the list of in-neighbors of v in $G = G_t$ we will consider every $i \in [t]$ such that $b_i = 1$. The fact that $b_i = 1$ means that in the construction of G_i, the node v has been in the set R_i and we should find the list of the vertices L_i that are connected to it. The list of in-neighbors of v in G_i (which are due to the edges of the expanding graphs planted in G_i) can be computed in time polylog N (due to explicitness of G_i). Suppose the latter list is $\{v_1, \ldots, v_\ell\}$ where the binary representation of the numbers $v_j - 1$ (for all $j \in [\ell]$) have $i - 1$ bits. To get the index of v_j as a node in G we can take $u_j = 1 + (b_t, \ldots, b_{i+1}, 0, (v_j - 1))$. To find out all the in-neighbors of v in G we just have to go over all $i \in \lceil \log N \rceil$ and extract the in-neighbors as above.

Depth-Robustness. We now prove the depth-robustness of the DAG of Construction A.3.

Lemma A.4. *For every i and $\alpha \in (0,1)$, the graph G_i is $(\alpha, \alpha - i\gamma)$-depth-robust, and since $\gamma = \varepsilon/\log N$, the final graph $G = G_{\log N}$ is $(\alpha, \alpha-\varepsilon)$-depth-robust for every $\alpha \in (0,1)$.*

Proof. The proof is by induction over i. For $i = 1$, G_i consists of two vertices, and is trivially (α, α) depth-robust for all $\alpha \in (0,1)$. Assuming the hypothesis holds for i, consider the graph $G_{i+1} = ((L_{i+1}, R_{i+1}), E_{i+1})$.

Fix an arbitrary $\alpha \in (0,1)$. Suppose we select a subset S of the nodes of G_{i+1} of size at least $\alpha \cdot 2^{i+1}$ and call them "good" nodes. Let $\delta 2^i$ be the number of good nodes $S_L \subseteq S$ in L_{i+1}. Since the total number of good nodes is $\alpha 2^{i+1}$, there must be at least $(2\alpha - \delta)2^i$ good nodes $S_R \subseteq S$ in R_{i+1}. Below, by the length $|P|$ of a path P we denote the number of vertices in it.

By the induction hypothesis, there exists a path $P_L \subseteq S_L$ of good nodes such that $|P_L| \geq (\delta - i\gamma)2^i$. In the same way, there exists a path $P_R \subseteq S_R$ such that $|P_R| \geq (2\alpha - \delta - i\gamma)2^i$.

We must show that there exists a path $P \subseteq S_L \cup S_R = S$ such that $|P| \geq (\alpha - (i+1)\gamma)2^{i+1}$. If $\delta \leq 2\gamma$, then we can simply set $P = P_R$, since in this case $|P_R| \geq (2\alpha - (i+2)\gamma)2^i \geq (\alpha - (i+1)\gamma)2^{i+1}$. On the other hand if $\delta \geq 2\alpha - 2\gamma$, then in the same way we can set $P = P_L$. Otherwise, consider the set $\widehat{P}_R \subseteq P_R$ consisting of the first $\lceil \gamma 2^i \rceil$ nodes on P_R and the set $\widehat{P}_L \subseteq P_L$ consisting of the last $\lceil \gamma 2^i \rceil$ nodes on P_L. Since G_{i+1} contains a $(1/\gamma)$-expanding graph between the nodes of L_{i+1} and R_{i+1}, and given that $|\widehat{P}_L| = |\widehat{P}_R| = \lceil \gamma 2^i \rceil$, there exists an edge $(v_L, v_R) \in E_{i+1}$ going from $\widehat{P}_L$ to $\widehat{P}_R$. We define our new path P by connecting (most of) the paths P_L and P_R together with the edge (v_L, v_R) as follows: P is defined to be the nodes in P_L up to the node v_L, concatenated with the nodes of P_R starting from the node v_R. This way the number of vertices in P is at least

$$\begin{aligned} |P| &\geq |P_R| - (\lceil \gamma 2^i \rceil - 1) + |P_L| - (\lceil \gamma 2^i \rceil - 1) \\ &> (\delta - i\gamma + 2\alpha - \delta - i\gamma)2^i - 2\gamma 2^i = (\alpha - (i+1)\gamma)2^{i+1} \ . \end{aligned}$$

□

B Commitments by Merkle Trees

Algorithm 5 shows how a Merkle tree is computed as a commitment to a set of strings with the possibility of opening the commitment to each string separately. Algorithm 6 describes how the opening is performed. To verify the decommitment of Algorithm 6 the receiver simply verifies the corresponding hash evaluations according to Algorithm 7. For simplicity, in this section we assume that the N strings being committed to are indexed by $\{0, 1, \ldots, N-1\}$ (rather than $[N]$), but when we use the Merkle commitment we might choose to index the strings with $[N]$.

The following lemma asserts that if the family of hash functions $\mathcal{CH}$ from which $ch(\cdot)$ is sampled is collision resistant, then the commitment scheme based on the Merkle tree is binding. It can be shown that the commitment using Merkle trees has some strong hiding properties as well, but here we are only concerned with the binding property of such efficient commitments.

Algorithm 5 (Merkle Commitment) For a hash function $ch\colon \{0,1\}^{2n} \mapsto \{0,1\}^n$ and N strings $u_0, \ldots, u_{N-1}$ from $u_i \in \{0,1\}^n$ the Merkle tree is computed as follows.

Let $t = \lceil \log N \rceil$, and define $u_i = 0^n$ for $N \leq i < 2^t$.
for $i \in \{0, \ldots, 2^t - 1\}$ **do**
    set $c_i^t = u_i$.
for $j \in \{t, t-1, \ldots 1\}$ **do** {compute $(j-1)$-th "layer" of the Merkle tree}
    for $i \in \{0, 1, \ldots, 2^{j-1} - 1\}$ **do**
        Let $c_i^{j-1} = ch(c_{2i}^j, c_{2i+1}^j)$
Output $c = c_0^0$ as the commitment string.

Algorithm 6 (Merkle Opening) For a hash function $ch\colon \{0,1\}^{2n} \mapsto \{0,1\}^n$ and as the Merkle commitment $c \in \{0,1\}^n$ for $N = 2^t$ strings of length n, the opening algorithm is as follows.

Receive some index $i \in \{0, \ldots, 2^t - 1\}$ as the index of the string to be opened.
Output u_i as the decommitment value.
To help the verifier verify the decommitment u_i, let $i = (b_t, \ldots, b_1)$ be the binary representation of i (i.e., $i = \sum_{j \in [t]} b_j 2^{j-1}$) and do the following.
for $j \in \{t, \ldots, 1\}$ **do**
    Output the two strings $\overline{c}_0^j = c^j_{(b_t, \ldots b_{t-j+2}, 0)}$ and $\overline{c}_1^j = c^j_{(b_t, \ldots b_{t-j+2}, 1)}$ (from the j-th layer).
    (Note that one of $c^t_{(b_t, \ldots, b_2, 0)}$ and $c^t_{(b_t, \ldots, b_2, 1)}$ is simply equal the decommitted value u_i.)

Algorithm 7 (Merkle Verification) For a hash function $ch\colon \{0,1\}^{2n} \mapsto \{0,1\}^n$ and a received $c \in \{0,1\}^n$ as the Merkle commitment of $N = 2^t$ strings of length n, the verifying algorithm is as follows.

Send the index $i \in \{0, \ldots, 2^t - 1\}$ (i.e., the index of the string desired to be decommitted) to the opener of Algorithm 6 and let $i = (b_t, \ldots, b_1)$ be the binary representation of i.
Receive u_i as the decommitment value, and also receive the strings $\overline{c}_0^j$ and $\overline{c}_1^j$ for all $j \in [t]$.
Verify that $u_i = \overline{c}^t_{b_1}$.
Define $b_{t+1} = 0$ and $\overline{c}_0^0 = c$ (where c is the commitment string received before).
For all $j \in \{t, \ldots, 1\}$ (can be done in parallel) verify that $ch(\overline{c}_0^j, \overline{c}_1^j) = \overline{c}^{j-1}_{b_{t-j+2}}$.

Lemma B.1. *Suppose* **Adv** *is an adversary who sends a a Merkle-commitment string* $c \in \{0,1\}^n$*, then send some* $i \in [N]$*, and finally Merkle-decommit successfully into* two different *values* $u_i \neq u'_i$ *(as the* i*-th string). Then there is an adversary* $\mathbf{Adv}'$ *who executes* **Adv** *as a black-box, asks* $O(\log N)$ *more queries to* $ch(\cdot)$*, and finds a collision:* $x \neq x', ch(x) = ch(x')$*.*

Proof. Let $t = \lceil \log N \rceil$. Suppose **Adv** is able to decommitment successfully into two different strings $u_i \neq u'_i$ as the i-th string with respect to the *same* commitment string c. For $j \in [t]$ let $\overline{c}_0^j$ and $\overline{c}_1^j$ be the pair of strings provided by **Adv** as the two needed strings from the j-th layer of the Merkle tree when decommitting to u_i, and similarly let $\overline{c'}_0^j$ and $\overline{c'}_1^j$ be the corresponding strings for u'_i. Define $\overline{c}_0^0 := c$ and $\overline{c'}_0^0 := c$. Since $\overline{c}_{b_1}^t = u_i \neq u'_i = \overline{c'}_{b_1}^t$, if we take j to be the smallest element in $[t]$ that $\overline{c}_{b_{t-j+1}}^j \neq \overline{c'}_{b_{t-j+1}}^j$, it holds that $ch(x) = \overline{c}_{b_{t-j+2}}^{j-1} = ch(x')$ for $x = (\overline{c}_0^j, \overline{c}_1^j) \neq (\overline{c'}_0^j, \overline{c'}_1^j) = x'$. $\mathbf{Adv}'$ is able to find such colliding pair by computing the relevant $2t$ hash labels $ch(\cdot)$. □