Non-Interactive Time-Stamping and Proofs of Work in the Random Oracle Model

Mohammad Mahmoody* Salil Vadhan† Tal Moran‡

October 6, 2011

Abstract

We construct a non-interactive scheme for proving computational work in the Random Oracle Model. Given a uniformly random "puzzle" $\mathcal{P} \xleftarrow{\$} \{0,1\}^n$ (where n is the security parameter), a corresponding "solution" can be generated using N oracle queries (for any parameter $n \leq N \leq 2^{o(n)}$), and any adversarial strategy for generating valid solutions must make $\Omega(N)$ adaptive rounds of oracle queries 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 $\text{poly}(n)$). Applications of these "time-lock puzzles" include non-interactive time-stamping of documents and universally verifiable CPU benchmarks.

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 lower-bound of Mahmoody, Moran, and Vadhan (CRYPTO '11), 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.

1 Introduction

A *time-stamping scheme* is a mechanism for proving that a document was created before a certain time in the past. They have variety of applications, including the resolution of intellectual property disputes (e.g., an inventor may time-stamp 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 time-stamping scheme is *non-interactive* if generating a time-stamp does not require communication with a third party. This is a desirable property, both because it makes time-stamping easily scalable (multiple parties generating time-stamps do not interfere with each other) and because it allows parties to hide the fact that are generating a time-stamp (e.g., an inventor may not wish to reveal the fact that she has a new invention).

A natural cryptographic approach to time-stamping 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

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

†Harvard, salil@seas.harvard.edu.

‡Herzliya Interdisciplinary Center, talm@seas.harvard.edu.

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 $\approx T$ to evaluate, then the value $g(D)$ can be considered a "proof" that D was known T time units in the past.

Note that here we need both an upper-bound and 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 much more efficient 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 $n = \log T$. Given a document $D \in \{0,1\}^n$, the proof of work $f^{-1}(D)$ can be computed by brute force in time approximately $2^n = T$. Such a proof can be verified very quickly (e.g. in time $\text{poly}(n) = \text{poly}(\log T)$). Moreover, it is a plausible assumption that any strategy for inverting f will require time $\Omega(2^n) = \Omega(T)$, at least on a uniformly random document $D \xleftarrow{\$} \{0,1\}^n$. (If D is not uniform, then we can heuristically apply this construction to a hash of D.)

One deficiency of the aforementioned construction is that while it certifies that T 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 T. (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.)

Jerschow and Mauve [19] proposed a time-stamping scheme of this form, where $g(D) = 2^{2^D} \pmod N$, for an RSA integer N whose factors are kept secret. A verifier that 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. Jerschow and Mauve base the security of their scheme on the conjecture that modular exponentiation is an inherently serial task.

Connection to Time-Lock Puzzles. The idea of using modular exponentiation as a proof of sequential work was first proposed by Cai, Lipton, Sedgewick and Yao [12], in the context of benchmarks, and by Rivest, Shamir and Wagner [23] in the context of *time-lock puzzles*. One can think of a time-lock puzzle as an interactive proof of sequential work: solving a time-lock puzzle should take approximately T time (even for a massively parallel solver), while generating the puzzle and verifying the solution should take considerably less. Thus, we can view a solution to the puzzle as a proof that at least roughly T time units has elapsed since the prover has received the puzzle.

When used as a time-stamping scheme, however, modular exponentiation has a serious drawback that the verifier must keep secret information: the factorization of the modulus. To convert a time-lock puzzle into a non-interactive time-stamping scheme that can be publicly verified, we need the time-lock puzzle to be *public coin*. That is, the puzzle $\mathcal{P}$ should simply be a uniformly random string, say from $\{0,1\}^n$. To time-stamp a document D, we solve the puzzle $\mathcal{P} = H(D)$, where H is a cryptographic hash function. (This can be proven secure when we model H as a random oracle. As before, if D is already uniformly random, then we can just take $\mathcal{P} = D$.)

To the best of our knowledge, only two constructions of time-lock puzzles have been proposed in the literature: the one based on modular exponentiation by Rivest et al. [23], and a construction in the random oracle model by Mahmoody, Moran and Vadhan [20], that has only a linear time-

gap between puzzle generation and solution (as opposed to the exponential gap of the modular exponentiation-based construction). Neither construction has the public-coin property.

1.1 Our Results

In this paper, following Mahmoody et. al [20], we study proofs of work (in the spirit of Dwork, Naor and Wee [14, 15]) and non-interactive time-stamping in the *random oracle model (ROM)*, where 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).

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) [4,17]. A proof of security in the random oracle model does not guarantee that the instantiated real-world scheme is secure (in fact, there are examples of schemes that are provably insecure for *any* instantiation of the oracle by a concrete hash function [2, 13]), but most natural protocols analyzed using this heuristic do appear to be secure in practice. Moreover, constructions in the ROM can be viewed as first steps towards constructions in the "standard model," e.g. as occurred for identity-based encryption [7, 8].

Time-Lock Puzzles and Non-interactive Time-Stamping. Our main result is the first construction of a time-lock puzzle (secure against parallel attack) with a public-coin puzzle generator. As described above, this immediately implies a non-interactive time-stamping scheme. (The security proof for both is in the Random Oracle Model.) The verification time for the time-lock puzzle (and the corresponding time-stamping scheme) can be poly-logarithmic in the time it takes to solve the puzzle, and the time-gap ratio between the honest and adversarial solvers is bounded by a constant. That is, our honest solver makes T oracle queries (with computation time $\widetilde{O}(T)$), and any adversary who solves the puzzle must have made $\Omega(T)$ *adaptive rounds* of oracle queries (even if making many more than T queries in each round).

Universally Verifiable Benchmarks. Cai, Lipton, Sedgewick and Yao suggested using proofs of work for running "uncheatable" benchmarks [12]. 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 factorization was kept secret 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 (the assumption about the beacon is that its 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. $O(\log N)$) 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 $u_v = \mathcal{O}(\mathcal{P}, u_{v_1}, \ldots, u_{v_d})$, where $v_1, \ldots, v_d$ are all vertices that have edges pointing to v, $\mathcal{P}$ is the puzzle, and $\mathcal{O}$ is the random oracle. (This is well-defined and can be computed with $O(N)$ oracle queries due to the acyclicity of G.) It then sends the verifier a short commitment to this labeling (using a Merkle tree). 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 $u_v = \mathcal{O}(\mathcal{P}, u_{v_1}, \ldots, u_{v_d})$ for each such vertex v. Intuitively, if the prover can pass this check for a large fraction of vertices v, then by depth-robustness of G, the labeling constructed by the prover must have a hash chain of length $\Omega(N)$, and the prover must have made $\Omega(N)$ rounds of adaptive oracle queries.

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 [16, 22, 24–26]. We use a construction of Erdős, Graham and Szemerédi [16], 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 [26]. As far as we know, our work is the first "positive" use of depth-robust graphs.

Bypassing a Lower-Bound. Mahmoody, Moran and Vadhan considered time-lock puzzles in the random oracle model [20] and proved that for a large class of time-lock puzzles (all puzzles in which the verifier does not query the oracle—which includes all puzzles in which the generator produces a solution together with the puzzle), time-lock puzzles with a large gap between generation/verification and solution time are not possible. They do this by showing that any puzzle that requires t queries to generate 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 this lower-bound by constructing a verifier that does query the oracle, and in fact our construction gives a time-lock puzzle with a super-polynomial gap between solution and verification times!

1.2 Related Work

Time Stamping. While physical time-stamps 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 [18] was more recent. Haber and Stornetta's main idea relies on a *Time-Stamping Service* (TSS): a trusted third party that is responsible for generating and managing the time-stamps. Further work in this direction has improved communication and computational complexity, and allowed the use of an *untrusted* TSS (for examples, see [1, 3, 5, 6, 9–11]). The state-of-the-art schemes using third-party Time-Stamping Services are efficient, can give very precise time-stamps and even hide the contents of document that is stamped, but necessarily reveal the fact that a party is stamping *some* document.

The first construction of a *non-interactive* time-stamping scheme was given by Moran, Shaltiel and Ta-Shma [21], 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). To generate a time-stamp on a document at time t, the stamper uses the document to select a subset of a string at time t and stores that subset. At every time period, verifiers store a small random subset of the string. To prove that a time-stamp is valid, the stamper proves that her stored subset

is consistent with the values stored by the verifier.

Proofs of Work. Dwork and Naor [14] originally suggested using proofs-of-work as a "pricing mechanism" to fight SPAM and other denial-of-service 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 in terms of resources expended). For this purpose, 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 [15] 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.

2 Time-Lock Puzzles and Applications

Definition 2.1 (Time-Lock Puzzles—Informal)**.** A *time-lock puzzle* is a game between three parties $(\mathsf{Gen}, \mathsf{Sol}, \mathsf{Ver})$ 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" $\mathcal{S}$.
3. The verifier Ver receives the puzzle $\mathcal{P}$ and a solution $\mathcal{S}$ and either accepts or rejects.

We require the following properties. (The exact definition of the running time function $\mathbf{Time}(\cdot)$ below for each party depends on the computational model in which the game is performed.)

- **Completeness.** When parties execute the game honestly Ver accepts with probability ≈ 1.
- **Time-Gap.** We would like to have $\mathbf{Time}(\mathsf{Gen}) + \mathbf{Time}(\mathsf{Ver}) \ll \mathbf{Time}(\mathsf{Sol})$.
- **Soundness.** Let $\widehat{\mathsf{Sol}}$ be a solver who acts in two steps $\widehat{\mathsf{Sol}} = (\widehat{\mathsf{Sol}}_1, \widehat{\mathsf{Sol}}_2)$ as follows.
  1. Before receiving the puzzle $\mathcal{P}$, $\widehat{\mathsf{Sol}}_1$ runs in a possibly long time $\mathbf{Time}(\widehat{\mathsf{Sol}}_1)$ (which could be much larger than $\mathbf{Time}(\mathsf{Sol})$ but still should be at most exponential in the security parameter), and outputs some internal state information St. Namely, $\widehat{\mathsf{Sol}}$ is allowed to do some expensive preprocessing.
  2. After getting the puzzle $\mathcal{P}$, the solver $\widehat{\mathsf{Sol}}_2(\mathcal{P}, \mathsf{St})$ runs in some time which is slightly smaller than the time of the honest solver: $\mathbf{Time}(\widehat{\mathsf{Sol}}_2) < \mathbf{Time}(\mathsf{Sol})$ and outputs some candidate solution $\widehat{\mathcal{S}}$.

  The soundness property asserts that any such $\widehat{\mathsf{Sol}} = (\widehat{\mathsf{Sol}}_1, \widehat{\mathsf{Sol}}_2)$ should be able to generate an accepting solution $\widehat{\mathcal{S}}$ only with negligible probability. Intuitively, this guarantees that a successful $\widehat{\mathsf{Sol}}$ after receiving the puzzle must have invested almost as much computational effort as the honest solver Sol does.

- **Parallel Soundness.** The stronger notion of *parallel* soundness is defined similar to the definition of soundless above with the difference that we only restrict the adversary to $\mathbf{ParTime}(\widehat{\mathsf{Sol}}_2) < \mathbf{Time}(\mathsf{Sol})$ where $\mathbf{ParTime}(\widehat{\mathsf{Sol}}_2)$ is the parallel running time of the adversary (after receiving the puzzle). We allow the total work $\mathbf{Time}(\widehat{\mathsf{Sol}}_2)$ to be much larger than $\mathbf{Time}(\mathsf{Sol})$ (but similar to $\mathbf{Time}(\widehat{\mathsf{Sol}}_1)$ it is at most exponential in the security parameter).

Variants of Definition 2.1. One can think of a more general definition in which the puzzle generation and verification are both interactive, however since we present a construction of the form above with a minimal interaction, we will work with the simpler definition above. One can also think of a verification process in which the secret randomness of the puzzle generator is also given to the verifier, but since our construction has the feature that its puzzles are "publicly verifiable" (i.e., the puzzle description is simply a random string and the verification does not require any secret information), again Definition 2.1 suffices for our purposes.

Time Complexity in the Random Oracle Model. 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. Our goal is to achieve time-lock puzzles in this model with the following features.

- We use two input parameters n and N where n is the security parameter and $N \gg n$ is a parameter specifying the Time-Gap.
- Gen and Ver run in some small $\mathrm{poly}(n)$ time (e.g., $O(n)$ or $O(n^2)$).
- Sol runs in time $\approx N$ where we think of N as some large polynomial n^{10} or even sub-exponential $2^{n^{1/10}}$.
- The malicious solver $\widehat{\mathsf{Sol}}$ is bound to parallel time $\mathbf{ParTime}(\widehat{\mathsf{Sol}}_2) = o(N)$. This is because we want to have a true *time-gap* even for adversaries that employ a large amount of parallel resources. Since we are in the random oracle model, we do not restrict the computational power of the adversary $\widehat{\mathsf{Sol}}$, but we do restrict $\mathbf{ParTime}(\widehat{\mathsf{Sol}}_2) = o(N)$ and $\mathbf{Time}(\widehat{\mathsf{Sol}}) = 2^{o(n)}$. The restriction $\mathbf{Time}(\widehat{\mathsf{Sol}}) = 2^{o(n)}$ is because otherwise the adversary can query the entire (relevant part of the) oracle in one round.

A Barrier. Mahmoody, Moran, and Vadhan [20] showed that in the random oracle model if $\mathbf{Time}(\mathsf{Gen}) = n$ and $\mathbf{Time}(\mathsf{Sol}) = N$, and that Ver does not ask any oracle queries during the final verification, then there always exists a malicious solver $\widehat{\mathsf{Sol}}$ who asks a total of $\mathrm{poly}(n, N)$ oracle queries in only n rounds (i.e., $\mathbf{ParTime}(\widehat{\mathsf{Sol}}_2) = n \ll N$ and $\mathbf{Time}(\widehat{\mathsf{Sol}}) = \mathrm{poly}(n, N) = 2^{o(n)}$) and solves the puzzle almost as well as the honest solver. This impossibility result rules out any time-lock puzzles in the random oracle model in which the puzzle generator knows the solution "ahead of time".[1] Thus, for such natural settings it is not possible to achieve both the Time-Gap and Parallel Soundness properties for the time-lock puzzle at the same time. In this work we show that by relaxing the definition of a time-lock puzzle and allowing the verifier to access the oracle, we can achieve all the properties of Definition 2.1 simultaneously.

[1] If the verifier receives the "solution" $\mathcal{S}$ directly from the puzzle generator, it only needs to compare the solution coming from Sol to $\mathcal{S}$. We note that the impossibility of [20] extends even to the setting that the verifier receives a secret message from Gen.

Theorem 2.2 (Main Result)**.** *There exists a time-lock puzzle in the random oracle model with two input parameters n, N such that $n \leq N \leq 2^{o(n)}$ and the following holds.*

- *The puzzle generator* Gen *asks no oracle queries and simply outputs $\mathcal{P} \xleftarrow{\$} \{0,1\}^n$.*
- *The honest solver* Sol *runs in time* $\mathrm{poly}(n) \cdot N$ *and asks $O(N)$ oracle queries.*
- *The puzzle verifier* Ver *runs in time* $\mathrm{poly}(n)$ *(and so asks $\leq \mathrm{poly}(n)$ oracle queries).*
- **Completeness.** *The solution $\mathcal{S}$ generated by* Sol *is accepted with probability one.*
- **Parallel Soundness.** *Any malicious solver* $\widehat{\mathsf{Sol}}$ *who asks $2^{o(n)}$ many oracle queries totally and asks $o(N)$ rounds of oracle queries after receiving the puzzle $\mathcal{P}$ is able to make* Ver *accept only with probability at most $2^{-\Omega(n)}$.*

Time-Stamping Documents Using Time-Lock Puzzles. Let $\mathbf{D}$ be a distribution over some "documents" and for simplicity suppose that $\mathbf{D}$ is a uniform distribution over $\{0,1\}^n$. A party Bob who has access to some $D \xleftarrow{\$} \mathbf{D}$ can generate a "time-stamp" $\mathcal{S}$ proving that he has had access to this document at least $\Omega(N)$ time units before the current moment. All Bob has to do is to use D as the description of a puzzle in the time-lock puzzle of Theorem 2.2 and generate a solution $\mathcal{S}$ based on that. The soundness of this time-lock puzzle guarantees that if Bob is malicious and (as opposed to his claim) has received D only in $o(N)$ time units before the current moment, he is able to provide an accepting solution $\widehat{\mathcal{S}}$ only with probability $2^{-\Omega(n)}$. To time-stamp larger documents which come from a distribution $\mathbf{D}$ of min-entropy $\mathrm{H}_\infty(\mathbf{D}) \geq \Omega(n)$, one can simply apply the random oracle $\mathcal{O}(\cdot)$ to the received document $D \xleftarrow{\$} \mathbf{D}$ and take the output bits of the answer as the description of the puzzle. As we will see in the next section, the soundness of the puzzle of Theorem 2.2 only relies on the fact that the adversary is not able to predict the puzzle with probability more than $2^{-\Omega(n)}$, and it easy to see that by hashing a random variable $\mathbf{D}$ of min-entropy $\Omega(n)$ into n bits using the random oracle this requirement is satisfied.

3 Constructing Time-Lock Puzzles

In this section we prove Theorem 2.2.

Intuition. The high-level idea is to force the solver to commit to a labeling of a DAG G as a "hash-graph" in which each node is labeled with the hash of its in-neighbors' labels. We call these labels also "hash labels". The verifier will then challenge the solver by choosing a small subset of nodes at random. For each of the selected nodes, the solver will open the commitment to the hash labels of the selected node itself and also those of its in-neighbors. The verifier will accept if all the commitments are opened correctly and the label of each selected node is equal to the hash of the labels of its in-neighbors. Intuitively, due to this randomized verification, the adversary is forced to compute the hash labels honestly for "many" of the nodes of the hash-graph he is committing to. So what we need to guarantee the soundness is to ensure that any large subset of the nodes of the hash-graph (which correspond to the nodes whose labels are indeed the hash of their in-neighbors' labels) there is a "large" path. That path would correspond to a large sequence of adaptive queries asked by the solver. A combinatorial property to guarantee the existence of such long paths in

any large subgraph of the underlying hash-graph is formalized as the notion of *depth-robustness* of DAGs (see Definition 3.5). We use depth-robust DAGs constructed by Erdős, Graham, and Szemerédi [16].

Formally, we first describe the construction and then prove its properties. We describe our construction using an *interactive* verifier in a hybrid model in which an ideal commitment functionality with *selective opening* exists. As a final step we will use the random oracle and a Merkle tree to implement the commitment functionality (see Section A) and apply the Fiat-Shamir transformation to make the verification non-interactive (see Section B). Algorithm 1 describes the honest (interactive) solver and Algorithm 2 describes the corresponding (interactive) verifier.

Definition 3.1 (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 $V_G = \{1, \ldots, N\}$ such that for every edge $(i, j) \in E_G$ connecting the vertex i to the vertex j it holds that $i < j$. The ordering $\{1, \ldots, N\}$ is called the *topological order* of the vertices and we always assume that our DAGs are given in topological order. For any vertex $j \in [N]$ we call $\mathrm{IN}(j) = \{i \mid (i, j) \in E_G\}$ the *in-neighbors* of the node j and call $\mathrm{din}(j) = |\mathrm{IN}(j)|$ the *in-degree* of the vertex j. We say G is of *in-degree* d if $\mathrm{din}(j) \leq d$ for all $j \in V_G$. By $\mathrm{depth}(G)$ we denote the length of the longest path in 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 $\mathrm{polylog}(N)$ the index of the j-th in-neighbor of the node i.

Remark 3.2. 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 $d - \mathrm{din}(j)$ multiple edges from the node 0 to the node j to make the in-degrees of every $j \in [N]$ exactly equal to d.

Notation. For a random variable $\mathbf{x}$, by $x \xleftarrow{\$} \mathbf{x}$ we mean that x is sampled according to the distribution of $\mathbf{x}$. For a set S, $x \xleftarrow{\$} S$ we mean that x is sampled uniformly at random from S. By $[n]$ we denote the set $\{1, 2, \ldots, n\}$. All the logarithms in this paper are in base 2 unless specified otherwise (e.g., $\log_\lambda N = \log N / \log \lambda$).

Getting a "Hash Oracle" $H(\cdot)$**.** For puzzles of size $|\mathcal{P}| = n$ we use a random oracle of the same security parameter $\mathcal{O}\colon \{0,1\}^* \mapsto \{0,1\}^n$. Given the puzzle $\mathcal{P}$ we define the *hash oracle* $H(\cdot)$ as follows: $H(x) = \mathcal{O}(\mathcal{P}, x)$. One can think of $H(\cdot)$ as *family* of hash functions and the puzzle $\mathcal{P}$ as an index determining the hash function.

Construction 3.3 (Time-Lock Puzzle)**.** Given parameters n and $N \leq 2^{o(n)}$ and a DAG G (implicitly known the the parties) of N vertices and in-degree d, the components of the time-lock puzzle construction $\Pi_{n,N}$ work as follows.

- The puzzle generator Gen outputs $\mathcal{P} \xleftarrow{\$} \{0,1\}^n$.
- The puzzle solver Sol and the puzzle-verifier Ver follow, in order, Algorithm 1 and Algorithm 2 with the following modifications:
  - By using Algorithms 4, 5, and 6 (and using the hash oracle $H(\cdot)$) a Merkle commitment scheme is employed instead of an ideal commitment with selective opening.
  - The Fiat-Shamir transformation of Lemma B.1 is applied to remove the challenge message of the verifier and make the verification non-interactive.

Algorithm 1 Honest solver Sol on input puzzle $\mathcal{P}$ and hash oracle H with output length $n = |\mathcal{P}|$, using a DAG G of in-degree d and N vertices given in the topological order.

1: Initially assign the hash label $u_0 = 0^n$ to the extra redundant node that is used to make the in-degrees equal to d (see Remark 3.2).
2: **for** $v \in \{1, \ldots, N\}$ **do** {Compute the hash-labels corresponding to the nodes of G}
3: Suppose $u_{v_1}, u_{v_2}, \ldots, u_{v_d}$ are the hash labels of the d in-neighbors of v.
Set $u_v = H(\mathcal{P}, u_{v_1}, \ldots, u_{v_d})$.
4: Send a commitment c to $(u_1, u_2, \ldots, u_N)$ to the verifier with the selective opening feature so that the verifier can ask to open the block u_j for every $j \in [N]$.
5: Receive a set of challenge nodes $\{v_1, \ldots, v_k\}$ from the verifier.
6: **for** $i \in \{1, \ldots, k\}$ **do**
7: Open the commitments to u_{v_i} and u_v for all $v \in \mathrm{IN}(v_i)$.

Algorithm 2 Verifier of a solution for the puzzle $\mathcal{P}$ using oracle H and the DAG G of N vertices in topological order and in-degree d.

1: Receive the commitment c (supposedly to 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 that $u_{v_i} = H(\mathcal{P}, u_{v_{(1,i)}}, \ldots, u_{v_{(d,i)}})$ where $v_{(1,i)} \leq \cdots \leq v_{(d,i)}$ are the in-neighbors of v_i.

3.1 Properties of Construction 3.3

Now we analyze the properties of Construction 3.3.

Completeness. Clearly if the puzzle solver and the verifier follow (the interactive or non-interactive versions of) Algorithms 1 and 2 then the verifier accepts with probability one.

Running Times and the Time-Gap. Now we analyze the running time of the honest parties.

Lemma 3.4. *The running time of the parties in Construction 3.3 is as follows.*

- *The puzzle generator* Gen *asks no oracle queries and simply outputs a string.*
- *The solver* Sol *asks* $O(N)$ *oracle queries and runs in time* $\mathrm{poly}(n) \cdot (kdN)$ *(where* k *is the number of challenge nodes asked to the solver and* d *is the in-degree of the graph* G*). Note that* $\mathrm{poly}(n) \cdot (kdN) \leq \mathrm{poly}(n) \cdot N$ *assuming that* $k, d \leq \mathrm{poly}(n)$*.*
- *The verifier runs in time* $\mathrm{poly}(n) \cdot kd$ *which is at most* $\mathrm{poly}(n)$ *assuming that* $k, d \leq \mathrm{poly}(n)$*.*

Proof. The solver Sol asks N oracle queries to computes the hash labels. Using a Merkle tree to commit to the N hash labels requires $O(N) + O(N/2) + O(N/4) \cdots \leq O(N)$ more queries to the oracle $H(\cdot)$. Thus the total number of oracle queries are $O(N)$. In order to bound the actual running time (not just its number of oracle queries) of the honest solver, we note that the length of the oracle queries asked to $H(\cdot)$ are at most $(d+1) \cdot |\mathcal{P}| = O(dn)$. We shall also include the time it takes to reveal the commitments values. To open the commitment to every hash label u_v

the prover needs to reveal $O(\log N) = o(n)$ blocks of length n. The number of such reveals is at most $(d+1) \cdot k$. Therefore the running time of Sol is at most $\mathrm{poly}(n) \cdot (kdN)$.

The verifier Ver needs to verify the commitment to the hash labels of at most $k \cdot (d+1)$ many vertices. For each of these verifications she needs to ask $O(\log N) = o(n)$ queries to $H(\cdot)$ (according to Algorithm 6) to ensure the correctness of the computation of the corresponding path of the Merkle tree. Therefore the total number of oracle queries of Ver and its running time are both at most $k \cdot (1+d) \cdot O(\log N)) \cdot \mathrm{poly}(n) \leq \mathrm{poly}(n) \cdot kd$. □

3.1.1 Parallel Soundness of Construction 3.3

Proving the soundness of the Construction 3.3 requires the underlying DAG G to have some combinatorial properties. For this purpose we first define the notion of depth-robustness of DAGs which was previously studied for the purpose of complexity lower-bounds (see Section 1).

Definition 3.5 (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.

Lemma 3.6 (Parallel Soundness of Construction 3.3)**.** *Suppose the DAG G used in Construction 3.3 is (α,β)-depth-robust and $\alpha^k = 2^{-\Omega(n)}$. Then any malicious solver $\widehat{\mathsf{Sol}}$ who asks $2^{o(n)}$ oracle queries to the random oracle $\mathcal{O}$ in at most $\beta \cdot N - 3$ rounds of adaptive queries after getting the puzzle $\mathcal{P}$, is able to make the verifier accept only with probability at most $2^{-\Omega(n)}$.*

We use a construction of Erdős, Graham and Szemerédi [16] which is based on a recursive use of constant-degree expanders and can be made explicit using any explicit family of such expanders.

Theorem 3.7 ([16])**.** *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 $0 < \beta < \alpha < 1$.

Concluding Theorem 2.2. By using $k = n$ in Construction 3.3, Theorem 2.2 follows as a corollary from Lemmas 3.4 and 3.6 and Theorem 3.7, because **(1)** it holds that $d = O(\log(N)) = o(n) \leq \mathrm{poly}(n)$, **(2)** for $k = n$ and $\alpha = 1 - \Omega(1)$ it holds that $\alpha^k = 2^{-\Omega(n)}$, and **(3)** for $\beta = \Omega(1)$ it holds that $(\beta \cdot N - 3) = \Omega(N)$. In the rest of this section we prove Lemma 3.6.

Intuition. First recall that due to the properties of the Fiat-Shamir transformation (Lemma B.1) we only need to prove Lemma 3.6 for the case of the *interactive* solver and verifier as specified in Algorithms 1 and 2. Since $H(\cdot)$ is a random oracle with n output bits, any $2^{o(n)}$-query adversary is able to find collisions for $H(\cdot)$ only with $2^{-\Omega(n)}$ probability. Therefore by Lemma A.1 we can conclude that the Merkle-tree based commitment used instead of the ideal commitment has binding error at most $2^{-\Omega(n)}$. In other words, informally speaking, after the solver sends some commitment c as the root of the Merkle tree, we can assume (up to some error $2^{-\Omega(n)}$) that there is a unique sequence of hash labels $u_1, \ldots, u_N$ assigned to the nodes of the graph G. For the committed labeling $u_1, \ldots, u_N$ 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 $\alpha^k \leq 2^{-\Omega(n)}$. 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 sequence of adaptive queries by the adversary!

For a formal proof, we first note that since $\widehat{\mathsf{Sol}}$ asks at most $2^{o(n)}$ number of oracle queries and that $\mathcal{P}$ is chosen at random from $\{0,1\}^n$, when the solver receives $\mathcal{P}$, with probability at least $1-2^{-\Omega(n)}$ over the choice of $\mathcal{P}$ none of the queries asked by $\widehat{\mathsf{Sol}}$ so far has the prefix $\mathcal{P}$. Therefore, the oracle $H(\cdot)$ defined above can be thought as a fresh random oracle with output length n whose randomness is completely unknown to $\widehat{\mathsf{Sol}}$ when receiving $\mathcal{P}$. First we prove that any such adversary is not able to find a "chain" of length more than $\beta \cdot N$ as defined below.

Definition 3.8. A *chain* of length r relative to the oracle $H(\cdot)$ is a sequence of strings $w_0, w_1 \ldots, w_r$ such that $H(w_{i+1})$ is a (contiguous) substring of w_i for all $i \in [r]$.

Lemma 3.9. *Suppose $H(\cdot)$ is a random oracle from $\{0,1\}^*$ to $\{0,1\}^n$ and suppose A is an oracle algorithm who asks $2^{o(n)}$ queries of length at most $2^{o(n)}$ to $H(\cdot)$ in $r-1$ adaptive rounds. The probability that A queries a chain of length r is at most $2^{-\Omega(n)}$.*

Proof. W.l.o.g. 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 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 $H(y_i)$ 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 (up to the error $2^{-\Omega(n)}$). Thus (by induction) the probability that A can output a chain of length r in $r-1$ rounds of queries is at most $2^{o(n)} \cdot 2^{-\Omega(n)} = 2^{-\Omega(n)}$. □

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 3.3 with probability at least $\varepsilon_1 \geq 2^{-o(n)}$ (when fed with the depth-robust graph of Theorem 3.7) by asking at most $q_1 \leq 2^{o(n)}$ oracle queries and $r_1 \leq \beta \cdot N - 3$ rounds of adaptive queries after receiving the puzzle $\mathcal{P}$. Then we will show how to turn this adversary into another adversary that violates Lemma 3.9, deriving a contradiction.

Removing the Fiat-Shamir Transformation. By Lemma B.1 we can conclude that there is another adversary $\mathbf{Adv}_2$ who breaks the parallel soundness of the *interactive* version of Construction 3.3 (using Algorithms 1 and 2) with probability at least $\varepsilon_2 > \varepsilon_1/q_2 \geq 2^{-o(n)}$ by asking at most $q_2 \leq \mathrm{poly}(n)q_1 \leq 2^{o(n)}$ oracle queries and $r_2 = r_1 + 1 \leq \beta \cdot N - 2$ rounds of adaptive queries after receiving the puzzle $\mathcal{P}$. To refute the possibility of such adversary $\mathbf{Adv}_2$, it suffices to prove the following lemma (which is phrased in a slightly more general form than Lemma 3.6 because it applies even when $\alpha^k = 2^{-o(n)}$).

Lemma 3.10 (Soundness of the *interactive* version of Construction 3.3)**.** *Suppose the DAG G used in Algorithms 1 and 2 is (α, β)-depth-robust. Then any malicious solver $\widehat{\mathsf{Sol}}$ who asks $2^{o(n)}$ oracle queries to the random oracle $\mathcal{O}$ in at most $\beta \cdot N - 2$ rounds of adaptive queries after getting the puzzle $\mathcal{P}$ is able to make the verifier of Algorithm 2 accept only with probability at most $\alpha^k + 2^{-\Omega(n)}$.*

Suppose there is some $\mathbf{Adv}_2$ who violates the claim of Lemma 3.10. Namely $\mathbf{Adv}_2$ asks $q_2 \leq 2^{o(n)}$ oracle queries, has $r_2 \leq \beta \cdot N - 2$ rounds of adaptivity after getting the puzzle, and convinces the verifier of Algorithm 2 with probability at least $\alpha^k + \varepsilon_2$ for $\varepsilon_2 \geq 2^{-o(n)}$. Algorithm 3 below shows how to use $\mathbf{Adv}_2$ and turn it into another adversary $\mathbf{Adv}_3$ that violates the claim of Lemma 3.9.

Algorithm 3 For $\mathbf{Adv}_2$, $q_2 = 2^{o(n)}$, $\varepsilon_2 \geq 2^{-o(n)}$, and r_2 as described above, the adversary $\mathbf{Adv}_3$ (who "extracts" a chain from $\mathbf{Adv}_2$) works as follows.

1: Run $\mathbf{Adv}_2$ over a random puzzle $\mathcal{P} \xleftarrow{\$} \{0,1\}^n$ and a random seed rand_2 and receive some commitment $c \in \{0,1\}^n$. At this moment save the state of the adversary $\mathbf{Adv}_2$ since we are going to execute $\mathbf{Adv}_2$ in many "different branches" *in parallel* by feeding many different challenge messages to $\mathbf{Adv}_2$ and asking it to open those commitments. We cannot afford to use standard "rewinding" since we want to keep the adaptivity of $\mathbf{Adv}_3$ close to that of $\mathbf{Adv}_2$.
2: Let $\delta = \varepsilon_2/(2N)$ and $\ell = n/\delta$.
3: For all $j \in [\ell]$ choose a random subsets $S_j \subset [N]$ of size $|S_j| = k$.
4: **for** all $j \in [\ell]$ *in parallel* **do**
5: Ask $\mathbf{Adv}_2$ to open the nodes in the challenge set S_j with respect to the commitment c.
6: After receiving all the decommitments for $(d+1) \cdot k \cdot \ell$ many (perhaps multiple choices of) nodes in $[N]$, ask *all* oracle queries required to verify them *in one round* (see Remark A.2).

Claim 3.11 below shows that assuming $\mathbf{Adv}_2$ with the properties mentioned above exists, the adversary $\mathbf{Adv}_3$ of Algorithm 3 would contradict the statement of Lemma 3.9 which (is a contradiction and) finishes the proof of Lemmas 3.10 and 3.6.

Claim 3.11. *The adversary $\mathbf{Adv}_3$ of Algorithm 3 asks at most $2^{o(n)}$ oracle queries in $r_3 = r_2 + 1 \leq \beta \cdot N - 1$ rounds and finds a chain of length at least $\beta \cdot N$ with probability at least $\varepsilon_3 \geq \varepsilon_2/3$.*

Proof. Even though $\mathbf{Adv}_3$ runs $\mathbf{Adv}_2$ over many different challenges, since all of these executions are done in parallel, the adaptivity of $\mathbf{Adv}_3$ is only one round more than $\mathbf{Adv}_2$ due to the final round of verifications. Also, the number of queries of $\mathbf{Adv}_3$ is at most $q_3 \leq O(Nnq_2/\varepsilon_2) = 2^{o(n)}$. So we only need to prove the existence of the long chain in the view of $\mathbf{Adv}_3$.

Since $\mathbf{Adv}_2$ succeeds in convincing the verifier with probability at least ε_2, by an average argument, with probability at least $\varepsilon_2/2$ over the choices of the puzzle $\mathcal{P}$ and the randomness of $\mathbf{Adv}_2$ (i.e., rand_2), $\mathbf{Adv}_2$ will have at least a chance of $\varepsilon_2/2$ (over the randomness of the oracle $H(\cdot)$ and 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 3 have this property. We will show that in this case, $\mathbf{Adv}_3$ succeeds in finding a (long enough) chain with probability at least $9/10$, leading to a total probability of success at least $(\varepsilon_2/2) \cdot (9/10) > \varepsilon_2/3$.

Suppose W is the event that $\mathbf{Adv}_2$ succeeds answering a random challenge set S of k nodes. Call a node $i \in [N]$ a *heavy* node if $\Pr[i \in S \text{ and } W] \geq \delta = \varepsilon_2/(2N)$ 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 $\mathbf{Adv}_2$ is able answer a random challenge $S \subset [N]$ of k nodes correctly only with probability:

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

which is at most $\alpha^k + \varepsilon_2/2$ as opposed to our assumption. On the other hand, since $\mathbf{Adv}_3$ chooses ℓ random challenge sets S_j of size k, for every heavy node $i \in [N]$, the probability that for some $j \in [\ell]$ $\mathbf{Adv}_2$ 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 $\mathbf{Adv}_2$ over some challenge set S).

We claim that for all $i \in [N]$ with probability $1 - 2^{-\Omega(n)}$ all the extracted hash labels for the same 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 get a $2^{o(n)}$-query adversary who is able to violate the binding of the Merkle commitment of Section A with probability at least $2^{-o(n)}$. By Lemma A.1 the latter, in turn, would imply an $2^{o(n)}$-query adversary who is able to find a collision in $H(\cdot)$ with probability $2^{-o(n)}$ which is not possible (due to the long enough n-bit output length of $H(\cdot)$.

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 $(\mathcal{P}, u_{v_1}, \ldots, u_{v_d})$ 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 $H(w_v) = u_v$ where u_v is the extracted hash label of v. Since the query $H(w_v)$ is already asked by $\mathbf{Adv}_3$ during the verifications, 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 $\mathbf{Adv}_3$ gets a chain of size at least $\beta \cdot N$. □

Acknowledgement. We thank the anonymous CRYPTO 2011 reviewers of our previous paper [20], whose comments led us to investigate the topics in this paper. We also thank Moni Naor and Avi Wigderson for pointers to relevant work.

The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the US government.

References

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

[2] B. Barak. How to go beyond the black-box simulation barrier. In *FOCS*, pages 106–115, 2001. 3

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

[4] M. Bellare and P. Rogaway. Random oracles are practical: a paradigm for designing efficient protocols. In *CCS*, CCS '93, pages 62–73, New York, NY, USA, 1993. ACM. 3

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

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

[7] D. Boneh and X. Boyen. Secure identity based encryption without random oracles. In M. K. Franklin, editor, *CRYPTO*, volume 3152 of *Lecture Notes in Computer Science*, pages 443–459. Springer, 2004. 3

[8] D. Boneh and M. K. Franklin. Identity-based encryption from the weil pairing. *SIAM J. Comput.*, 32(3):586–615, 2003. 3

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

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

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

[12] J. Cai, R. J. Lipton, R. Sedgewick, and A. C.-C. Yao. Towards uncheatable benchmarks. In *Structure in Complexity Theory Conference*, pages 2–11, 1993. 2, 3

[13] R. Canetti, O. Goldreich, and S. Halevi. The random oracle methodology, revisited. *J. ACM*, 51(4):557–594, 2004. 3

[14] C. Dwork and M. Naor. Pricing via processing or combatting junk mail. In E. F. Brickell, editor, *CRYPTO*, volume 740 of *Lecture Notes in Computer Science*, pages 139–147. Springer, 1992. 3, 5

[15] C. Dwork, M. Naor, and H. Wee. Pebbling and proofs of work. In V. Shoup, editor, *CRYPTO*, volume 3621 of *Lecture Notes in Computer Science*, pages 37–54. Springer, 2005. 3, 5

[16] P. Erdős, R. L. Graham, and E. Szemerédi. On sparse graphs with dense long paths. *Computers & Mathematics with Applications*, 1:365–369, 1975. 4, 8, 10

[17] Fiat and Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO: Proceedings of Crypto*, 1986. 3, 16

[18] S. Haber and W. S. Stornetta. How to time-stamp a digital document. *J. Cryptology*, 3(2):99–111, 1991. 4

[19] Y. Jerschow and M. Mauve. Offline submission with rsa time-lock puzzles. In *Computer and Information Technology (CIT), 2010 IEEE 10th International Conference on*, pages 1058–1064, 29 2010-july 1 2010. 2

[20] M. Mahmoody, T. Moran, and S. P. Vadhan. Time-lock puzzles in the random oracle model. In P. Rogaway, editor, *CRYPTO*, volume 6841 of *Lecture Notes in Computer Science*, pages 39–50. Springer, 2011. 2, 3, 4, 6, 13

[21] T. Moran, R. Shaltiel, and A. Ta-Shma. Non-interactive timestamping in the bounded storage model. *J. Cryptology*, 22(2):189–226, 2009. 4

[22] W. J. Paul and R. Reischuk. On alternation ii. a graph theoretic approach to determinism versus nondeterminism. *Acta Inf.*, 14:391–403, 1980. 4

[23] R. L. Rivest, A. Shamir, and D. A. Wagner. Time-lock puzzles and timed-release crypto. Technical Report MIT/LCS/TR-684, MIT, February 1996. 2

[24] G. Schnitger. A family of graphs with expensive depth reduction. *Theor. Comput. Sci.*, 18:89–93, 1982. 4

[25] G. Schnitger. On depth-reduction and grates. In *FOCS*, pages 323–328. IEEE, 1983. 4

[26] L. G. Valiant. Graph-theoretic arguments in low-level complexity. In J. Gruska, editor, *MFCS*, volume 53 of *Lecture Notes in Computer Science*, pages 162–176. Springer, 1977. 4

A Commitment Using Merkle Trees

Algorithm 4 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 5 describes how the opening is performed. To verify the decommitment of Algorithm 5 the receiver simply verifies the corresponding hash evaluations according to Algorithm 6. 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 $[N]$.

Algorithm 4 For a hash function $H\colon \{0,1\}^{2n} \mapsto \{0,1\}^n$ and N strings $u_0,\ldots,u_{N-1}$ from $u_i \in \{0,1\}^n$ the *Merkle tree* of $u_1,\ldots,u_N$ is computed as follows.

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

The following lemma asserts that if $H(\cdot)$ is collision resistant ($H(\cdot)$ might be "sampled" from a family of functions for that purpose), 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 For a hash function $H\colon \{0,1\}^{2n} \mapsto \{0,1\}^n$ and an output $c \in \{0,1\}^n$ as the Merkle commitment to $N = 2^t$ strings of length n, the opening algorithm is as follows.

1: Receive some index $i \in \{0, \ldots, 2^t - 1\}$ as the index of the string to be opened.
2: Output u_i as the decommitment value.
3: 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.
4: **for** $j \in \{t, \ldots, 1\}$ **do**
5: 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 6 For a hash function $H\colon \{0,1\}^{2n} \mapsto \{0,1\}^n$ 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.

1: 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 5 and let $i = (b_t, \ldots, b_1)$ be the binary representation of i.
2: Receive u_i as the decommitment value, and also receive the strings $\overline{c}_0^j$ and $\overline{c}_1^j$ for all $j \in [t]$.
3: Verify that $u_i = \overline{c}^t_{b_1}$.
4: Define $b_{t+1} = 0$ and $\overline{c}_0^0 = c$ (where c is the commitment string received before).
5: **for** $j \in \{t, \ldots, 1\}$ **do**
6: Verify that $H(\overline{c}_0^j, \overline{c}_1^j) = \overline{c}^{j-1}_{b_{t-j+2}}$

Lemma A.1. *For a Merkle-commitment string $c \in \{0,1\}^n$ sent to the receiver, let* **Adv** *be an adversary who is able to output some $i \in [N]$ and Merkle-decommit successfully into* two different *values $u_i \neq u_i'$ (as the i-th string). Then there is an adversary* **Adv′** *who executes* **Adv** *as a black-box, asks $O(\log N)$ more queries to $H(\cdot)$, and finds a pair of colliding inputs: $x \neq x', H(x) = H(x')$.*

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 $\overline{c}^t_{b_1} = u_i \neq u_i' = \overline{c'}^t_{b_1}$, if we take j to be the smallest element in $[t]$ that $\overline{c}^j_{b_{t-j+1}} \neq \overline{c'}^j_{b_{t-j+1}}$, it holds that $H(x) = \overline{c}^{j-1}_{b_{t-j+2}} = H(x')$ for $x = (\overline{c}_0^j, \overline{c}_1^j) \neq (\overline{c'}_0^j, \overline{c'}_1^j) = x'$. **Adv′** is able to find such colliding pair by computing the relevant $2t$ hash labels $H(\cdot)$. (These queries can be asked even in one round; see Remark A.2 below). □

Remark A.2 (Nonadaptive Verification)**.** We note that even though we stated the verification of Algorithm 6 in an adaptive way, the verification can be performed in only one round of adaptivity since all the queries to be asked are known from the beginning.

B Fiat-Shamir Transformation

The following lemma is due to Fiat and Shamir [17] and shows how to remove interaction from public-coin protocols in the random oracle model. Here we prove a special case in which there is

only four messages exchanged, we deal with exponential security, and we are interested in (almost) preserving the adaptivity of the adversary.

Lemma B.1 (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 coins and the verifier does not use any private randomness to make her final decision.*
- *We have $|v_1| = n$ and $|v_2| = \ell \cdot n$ for some $\ell = \text{poly}(n)$. (If only the inequality holds $|v_2| \leq \ell \cdot n$ we can always pad v_2 to make it equal.)*
- *The verifier rejects its interaction with probability $1 - 2^{-\Omega(n)}$ against* any *prover $\widehat{P}$ who:*
  1. *$\widehat{P}$ asks a total of $2^{o(n)}$ queries to the random oracle $\mathcal{O}$.*
  2. *$\widehat{P}$ has at most r rounds of adaptivity in its queries after receiving v_1.*

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 $2^{o(n)}$ number of queries to $\mathcal{O}$ and has at most $r - 1$ rounds of adaptivity after receiving v_1 is able to convince V' with probability at most $2^{-\Omega(n)}$.*

Proof. For sake of contradiction suppose $\widehat{P'}$ is an adversary who interacts with V', asks at most $q = 2^{o(n)}$ queries to $\mathcal{O}$, has at most $r - 1$ rounds of adaptivity after receiving v_1, and is able to convince V' with probability $\varepsilon > 2^{-o(n)}$. We show how to get an adversary $\widehat{P}$ who interacts with V, asks at most $q \cdot \ell = 2^{o(n)}$ oracle queries, has at most r rounds of adaptivity after receiving v_1 and is able to make V accept with probability at least $\varepsilon/(q\ell)$ (which is also $2^{-o(n)}$).

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 if not asked already. (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 $\ell = \text{poly}(n)$ and might add one round of adaptive queries to $\widehat{P'}$. Since the first message v_1 of the verifier is of length n and since $\widehat{P'}$ asks only $2^{o(n)}$ number of queries, the probability that $\widehat{P'}$ can ask any query with the prefix v_1 *before* receiving v_1 is only $2^{-\Omega(n)}$, and therefore in the following we safely assume that $\widehat{P'}$ makes the special queries $\mathcal{O}(v_1, p_1, j)$ *after* receiving v_1. The adversary $\widehat{P}$ works as follows:

1. Emulate the execution of $\widehat{P'}$ before receiving v_1.
2. When $\widehat{P'}$ asks v_1, receive v_1 and forward it to $\widehat{P'}$.

3. Choose $i \xleftarrow{\$} [q']$ at random where $q' \leq \ell \cdot q$ is the number of the remaining queries of $\widehat{P'}$.

4. Continue emulating 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 $1/q' \geq 1/(q\ell)$ 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 a 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 $\varepsilon \cdot 1/(q\ell)$, while the number of rounds of oracle queries asked by $\widehat{P}$ is at most r. □