Time-Lock Puzzles in the Random Oracle Model

Mohammad Mahmoody[1], Tal Moran[2], and Salil Vadhan*[2]

[1] Department of Computer Science, Cornell University
mohammad@cs.cornell.edu; http://www.cs.cornell.edu/~mohammad/
[2] School of Engineering and Applied Sciences and
Center for Research on Computation and Society, Harvard University
{talm,salil}@seas.harvard.edu; http://seas.harvard.edu/~{talm,salil}/

Abstract. A time-lock puzzle is a mechanism for sending messages "to the future". The sender publishes a puzzle whose solution is the message to be sent, thus hiding it until enough time has elapsed for the puzzle to be solved. For time-lock puzzles to be useful, generating a puzzle should take less time than solving it. Since adversaries may have access to many more computers than honest solvers, massively parallel solvers should not be able to produce a solution much faster than serial ones.

To date, we know of only one mechanism that is believed to satisfy these properties: the one proposed by Rivest, Shamir and Wagner (1996), who originally introduced the notion of time-lock puzzles. Their puzzle is based on the serial nature of exponentiation and the hardness of factoring, and is therefore vulnerable to advances in factoring techniques (as well as to quantum attacks).

In this work, we study the possibility of constructing time-lock puzzles in the random-oracle model. Our main result is negative, ruling out time-lock puzzles that require more parallel time to solve than the total work required to generate a puzzle. In particular, this should rule out black-box constructions of such time-lock puzzles from one-way permutations and collision-resistant hash-functions.

On the positive side, we construct a time-lock puzzle with a linear gap in parallel time: a new puzzle can be generated with one round of n parallel queries to the random oracle, but n rounds of *serial* queries are required to solve it (even for massively parallel adversaries).

1 Introduction

In this paper we revisit the subject of "timed-release crypto" based on "time-lock puzzles". The goal of timed-release crypto, introduced by May [22], is to encrypt a message in such a way that it will be readable at some specified time in the future (even without additional help from the sender), but not before then.

In addition to the basic use of "sending messages to the future", there are many other potential uses of timed-release crypto. Rivest, Shamir and Wagner [25] suggest, among other uses, delayed digital cash payments, sealed-bid auctions and key escrow. Boneh and Naor [9] define timed commitments and timed signatures and show that they can be used for fair contract signing, honesty-preserving auctions and more.

* Supported by NSF grant CNS-0831289.

A natural approach to building a timed-release crypto scheme is the use of time-lock puzzles: puzzles that take a prespecified amount of time to solve (which should be significantly longer than the time to generate the puzzle). Intuitively, using the solution of a time-lock puzzle as the key to an encryption scheme would force anyone wanting to decrypt the message to perform the computation for the time required to solve the puzzle. By tuning the difficulty of the solution according to the time we would like the message to remain secure, we can ensure that decryption will take at least that amount of time.

Inverting a (suitably weak) one-way function seems like an obvious candidate for a time-lock puzzle. However, as Rivest et al. observed, for many uses a generic one-way function would not suffice. This is because a potential adversary may have access to much larger computational resources than an honest party. Even if the processors available to the adversary are not be significantly faster than those available to the honest parties, it is reasonable to assume that a well-funded adversary could have access to many more processors (that could be used in parallel). Thus, we require that time-lock puzzles be "essentially sequential" in nature: having many parallel processors should not give a large advantage over a single processor in solving the puzzle.

The puzzles proposed in [25] are based on the conjecture that exponentiation (modulo an RSA integer) is such a task. In particular, if the factorization of the modulus is not known, the best known method for exponentiation is repeated squaring (which is conjectured to be essentially sequential). Given the factors of the modulus, however, there is a shortcut that allows the exponentiation to be performed much faster (so that the puzzles can be generated efficiently). Thus, there seems to be a super-polynomial gap between the work required to generate the puzzle and the parallel time required to solve it (for a polynomial number of parallel processors).

To the best of our knowledge, this construction of time-lock puzzles is the only one currently known that is resistant to parallel attack. The construction of Boneh and Naor [9] uses essentially the same idea. This leads to the natural question of whether we can construct time-lock puzzles based on other assumptions, preferably weaker and more general ones.

Biham, Goren and Ishai [5] suggest an additional motivation as well: obtaining (weak) key-agreement protocols based on one-way functions that resist quantum attack. They show that in the classical world there do exist weak key-agreement protocols based on one-way functions (of exponential strength) that force an adversary to work in time quadratic in the time of the honest parties, based on a variant of Merkle puzzles [23]. However, both their construction and the original Merkle puzzles are vulnerable to quantum attack via Grover's search algorithm [20]. Biham et al. note that Grover's speedup only applies to *parallel* search, and leave as an open problem whether such puzzles exist that are resistant to parallel attack (and thus, potentially, to quantum attack as well).

In this paper, we study time-lock puzzles in the *random oracle model*. In the random oracle model, we assume all parties have access to an oracle, H, modeled as a random function. In a real implementation, the random oracle is usually "instantiated" with a cryptographic hash function. We assume the adversary in this model is computationally

unbounded, and measure the difficulty of the time-lock puzzle by the number of queries the adversary is required to make to the random oracle in order to solve it.

The random oracle model is interesting for several reasons. First, negative results in this model rule out "black-box" constructions from one-way permutations and collision-resistant hash functions (since a random function is collision-resistant and indistinguishable from a random permutation using only a small number of queries; see e.g., [21,19,3] for details). Second, most "natural" protocols that have been proven secure in the random oracle model appear to be secure in practice as well (even though some "artificial" protocols are secure in this model but insecure for *any* explicit instantiation of the random oracle [11]), and constructing a protocol in this model is sometimes a first step towards constructing a provably-secure protocol in the plain model (e.g., the first efficient IBE scheme was proven secure in the random oracle model [8], after which constructions were found in the standard model as well [12,6,7]). Finally, the random oracle model is much simpler to analyze than models that incorporate computational complexity, and better understanding the problem in this setting may give insight into the complexity-theoretic case.

We can think of a time-lock puzzle generator as a randomized oracle algorithm f. The output of $f^H(r_A)$ (where r_A is the random input and H the random oracle) is a pair $(M, \mathcal{V})$: the puzzle M and a solution validator $\mathcal{V}$. The solver, given M, must output a solution x such that $\mathcal{V}(x) = 1$. When a time-lock puzzle has a single solution, such as when it is used to hide an encrypted message, $\mathcal{V}$ just compares its input to that constant value. In general, however, $\mathcal{V}$ may perform more complex verification and our negative results hold even when $\mathcal{V}$ is not efficient (note that $\mathcal{V}$ does not have access to the random oracle). For this to be a good time-lock puzzle, we would like f to be easy to compute but moderately hard to solve, even for a parallel adversary. More precisely, if we can compute $(M, \mathcal{V}) \leftarrow f^H(r_A)$ using n queries to H, we would like f to satisfy:

Completeness.

- There exists a (randomized) polynomial-time algorithm g (the honest solver) that solves puzzles generated by f: with high probability (over the random coins of f and g and the random oracle H), if we generate $(M, \mathcal{V}) \leftarrow f^H(r_A)$ and $x \leftarrow g^H(r_B, M)$ then $\mathcal{V}(x) = 1$. We use the shorthand notation $[(M, \mathcal{V}) \leftarrow f^H(r_A); x \leftarrow g^H(r_B, M); \mathcal{V}(x) = 1]$ to denote the event that the puzzle was generated as described above, the solver was run and its solution was valid. We denote by $m > n$ the number of queries g makes to H. m measures the difficulty for the honest solver and should be moderately larger than n, e.g., a large polynomial in n.

Soundness.

- Any algorithm that solves f and makes up to $q \gg m$ queries to H must use at least $m' \approx m$ levels of adaptivity. For example, we might take $q = n^{\omega(1)}$ and $m' = m/2$. The number of levels of adaptivity measures the complexity for a parallelized adversary; this requirement means that unless the adversary makes a very large number of queries, using parallelism won't give it an advantage over the honest solver.

1.1 Our Results

Time-lock puzzles with large difficulty gap are impossible. Our main result is a negative one. We show that for every time-lock puzzle there exists a parallel adversary that can solve the puzzle in no more time than it takes to generate and makes only polynomially more queries to the random oracle than the best honest (serial) solver. Thus, constructing time-lock puzzles with a "gap" between the work of the puzzle generator and the parallel time of the solver cannot be done in the random-oracle model.

Concretely, we prove two similar theorems but with incomparable parameters. The first provides an adversary that makes an optimal number of parallel query rounds, but may require super-polynomial time to run, even if the honest solver is efficient. Our second theorem gives a much simpler adversary construction that runs in polynomial time if the honest solver does, but has slightly worse adaptivity.

Formally, we prove the following two theorems:

Theorem 1 (Optimally Adaptive but Inefficient Adversary). *Let f be an oracle algorithm that makes at most n queries to a random oracle H and g an oracle algorithm that makes at most $m > n$ queries to H. If*

$$\Pr\left[(M,\mathcal{V}) \leftarrow f^H(r_A); x \leftarrow g^H(r_B, M); \mathcal{V}(x) = 1\right] \geq 1 - \nu$$

(i.e., when a puzzle is randomly generated after which the solver g is executed, its output is a valid solution with probability at least $1-\nu$ over the random coins r_A, r_B and H) then for all $\varepsilon \in (0,1)$ there exists an adversary, Ivy, that makes $\tilde{O}(nm/\varepsilon)$ queries to H, uses only n levels of adaptivity and satisfies $\Pr\left[(M,\mathcal{V}) \leftarrow f^H(r_A); x \leftarrow h^H(r_I, M); \mathcal{V}(x) = 1\right] \geq 1 - \nu - \varepsilon$ (where r_I is the variable denoting the random coins used by Ivy).

Theorem 2 (Efficient but Non-Optimal Adversary). *Let f be an oracle algorithm that makes at most n queries to a random oracle H and g an oracle algorithm that makes at most $m > n$ queries to H. If $\Pr\left[(M,\mathcal{V}) \leftarrow f^H(r_A); x \leftarrow g^H(r_B, M); \mathcal{V}(x) = 1\right] \geq 1 - \nu$ then for all $\varepsilon \in (0,1)$ there exists a deterministic adversary Javier (denoted by J) who makes at most nm/ε queries to H, uses only n/ε levels of adaptivity and satisfies $\Pr\left[(M,\mathcal{V}) \leftarrow f^H(r_A); x \leftarrow J^H(M); \mathcal{V}(x) = 1\right] \geq 1 - \nu - \varepsilon$. Moreover, the running time of J is $O(n/\varepsilon)$ times the running time of g.*

Some intuition for the proofs of both theorems, as well as the full proof of Theorem 2, can be found in Section 2. Due to space considerations, the proof of Theorem 1 is deferred to the full version.

By combining Theorem 1 with the result of [4], we partially resolve the open question of Biham et al. [5] by showing that every key-agreement protocol in the random-oracle model can be broken by a parallel attack that makes polynomially many queries to the random-oracle: Biham et al. were interested in whether there exist key-agreement protocols that resist quantum attack, but as a step towards this goal asked the question of whether there exist such protocols secure against a parallel *classical* attacker, and specifically whether such protocols could be based on random functions.

Corollary 1. *Let Π be a two-party protocol in the random oracle model such that when executing Π the two parties Alice and Bob make at most n queries each and their outputs*

are identical with probability at least $1 - \nu$. *Then for every* $0 < \varepsilon < 1$ *there exists an adversary that, given the public transcript of the protocol, outputs a value that agrees with Bob's output with probability* $1 - \nu - \varepsilon$ *using* $2n$ *levels of adaptivity and making* $\tilde{O}(n^3/\varepsilon^3)$ *total queries to* H.

Proof. Let $f^H(r_A, r_B) = (M, \mathcal{V})$, where M is the complete public transcript of Π when Alice uses the random coins r_A and Bob the random coins r_B. Define the corresponding solution validator to be

$$\mathcal{V}(x) = \begin{cases} 1 & \text{if } x \text{ is Bob's output in the execution of } \Pi \text{ in } f^H(r_A, r_B) \\ 0 & \text{otherwise} \end{cases}.$$

By the result of [4], there exists an adversary that makes at most $O(n^2/\varepsilon^2)$ queries to H and outputs a "correct" solution to this puzzle (i.e., an output that agrees with Bob) with probability $1 - \nu - \varepsilon/2$. Think of this adversary as the solver, g. By Theorem 1, this implies that there exists a solver that succeeds with probability $1 - \nu - \varepsilon$, makes $\tilde{O}((2n/\varepsilon) \cdot n^2/\varepsilon^2) = \tilde{O}(n^3/\varepsilon^3)$ total queries to H and uses only $2n$ levels of adaptivity (the $2n$ is because the total number of queries made by f is bounded by the total number of queries made by both Alice and Bob).

A time-lock puzzle with a linear gap in parallel time. Although our negative results rule out "strong" time-lock puzzles, they still leave open the possibility for a weaker version: one that can be generated with n parallel queries to the oracle but requires n rounds of adaptive queries to solve.

In a positive result, we show that such a puzzle can indeed be constructed. More formally, we prove:

Theorem 3. *Let* k *be a security parameter. There exist oracle functions* f *and* g *that satisfy:*

1. (Efficiency) $(M, \mathcal{V}_M) \leftarrow f^H(k, r)$ *can be computed using* n *parallel (non-adaptive) queries to* H.
2. (Completeness) $x \leftarrow g^H(k, M)$ *can be computed using* n *serial (adaptive) queries to* H *and the output of* g *always satisfies* $\mathcal{V}_M(x) = 1$ *(*g *is deterministic).*
3. (Soundness) *For every oracle function* h *that makes less than* n *serial rounds of queries to* H *and poly*(k) *queries overall to* H *in total,* $\Pr\left[(M, \mathcal{V}_M) \leftarrow f^H(k, r); x \leftarrow h^H(k, r_J, M); \mathcal{V}_M(x) = 1\right] = neg(k)$ *(where neg is some negligible function in* k*).*

The idea behind the construction is to force the solver to make sequential queries by "encrypting" each successive query with the result of an oracle call on its predecessor. The full construction and a sketch of its security proof appear in Section 3.

1.2 Related Work

Timed-Release Crypto Constructions. The notion of timed-release crypto was introduced by May [22]. May's proposal was to publish an encrypted message and distribute

the decryption key between several trusted agents. The agents would be instructed to publish their shares of the key at a specified future date. Rivest, Shamir and Wagner [25] introduced the idea of using time-lock puzzles instead of requiring a sender to trust an external entity and also developed May's "trusted-agent" approach, suggesting a scheme where the trusted agents' storage does not grow with the number of timed-release messages (as it does in May's scheme).

These two approaches, one based on puzzles and the other on trusted agents, have remained the basis of all new timed-release crypto schemes that we know of. There have been many improvements in the agent-based approach, focusing on reducing interaction between the agents and the users, achieving various verifiability and privacy properties ([8,13,14], among others). On the other hand, to the best of our knowledge, all existing time-lock puzzle constructions (that are resistant to parallel attack) are based on the problem originated by Rivest et al., namely that of exponentiation modulo an RSA integer.

Puzzles. The term "puzzle" to describe a cryptographic construction that is "meant to be broken" was first used by Merkle in the context of key agreement protocols [23]. Merkle's key-exchange protocol allows two users to exchange a key by solving a single puzzle, while forcing an adversary to solve multiple puzzles in order to discover it. The protocol does not require much structure from the puzzles, and can be instantiated with black-box use of one-way functions. The computation gap between the honest users and the adversary is quadratic in Merkle's scheme: if an honest user requires $O(N)$ time to recover the key, an adversary can recover it in $O(N^2)$ time.

Barak and Mahmoody showed that this is essentially optimal [4], improving a previous result by Impagliazzo and Rudich [21]. Both of these works give an upper bound for the computation gap of arbitrary key-exchange protocols in the random oracle model (including protocols that require multiple rounds of interaction between the two honest parties). Our work considers only one-message protocols, but bounds the *parallel* complexity of the adversary (in contrast to [21,4], who analyze the complexity of serial adversaries).

Puzzles have also been proposed as proof-of-work mechanisms for controlling spam and preventing denial-of-service-attacks. The idea was first introduced by Dwork and Naor [17], and was developed in multiple subsequent works [1,2,16,18]. Rivest and Shamir even suggest one variant for use as a micropayment system [24].

One major difference between these types of puzzles and those we consider in this work is that resistance to parallel attack is not as critical: for example, an adversary generating spam messages can always parallelize at the message level rather than by attacking a specific puzzle. Proofs-of-work, on the other hand, must be resistant to amortization (solving one puzzle should not help in solving others), whereas this is usually not a concern for time-lock puzzles.

For both types of puzzles, it is still important to take into account the gap between the computational capability of an honest user and that of the adversary. Abadi, Burrows, Manasse and Wobbler suggest basing the difficulty of a puzzle on memory access time [1], under the assumption that this has less variance among users than CPU speed. In a subsequent work, Dwork, Goldberg and Naor [16] construct such a function in the random oracle model that uses "pointer-chasing" in a large random table. This has a

very similar flavor to our time-lock puzzle construction in Section 3, although the goal is somewhat different and the analysis focuses on bounding memory accesses (to the table) rather than layers of adaptivity or queries to the random oracle.

2 Negative Results for Time-Lock Puzzles

In this section we give the intuition behind the proofs of our main results (Theorems 1 and 2) as well as the full proof of Theorem 2.

Following the seminal work of Impagliazzo and Rudich [21] on key agreement protocols in the random oracle model, our adversaries (for both theorems) attempt to find all *intersection queries* between the puzzle-generator f (Alice) and the solver g (Bob) — all queries made by both Alice and Bob. If successful, the adversary can then simulate an honest solver without asking additional queries. The novelty of our work is that we care not just about the total number of queries made by the adversary, but about the number of levels of adaptivity.

Both adversary constructions work in rounds, and query the random oracle only at the end of a round. Our aim is to reduce the total number of rounds (this is the "adaptivity level" of the adversary). The constructions differ in how they choose which queries to ask in each round, and in the corresponding proofs that the adversary succeeds in learning all of the intersection queries with high enough probability.

2.1 Intuition for Theorem 1

For the proof of Theorem 1, we use ideas (and a construction) of Impagliazzo and Rudich [21] (and later improvements by Barak and Mahmoody [4]), but modified to minimize the number of query rounds used by the adversary. Our attacker, Ivy, selects her queries to the random oracle in n rounds (n is the number of queries made by Alice). In round j, Ivy computes a set of *heavy* queries on which she will query the oracle at the end of the round. Heavy queries are those that have a high probability (given Ivy's view) of having been made by Alice, where "high" is a parameter that depends on n, m (the number of queries made by the honest solver) and ε (the probability with which Ivy is allowed to fail).

The intuition for why Ivy's attack works is that, as long as Bob has not hit any of Alice's "private" queries (those not made by Ivy), Bob doesn't know any more than Ivy about Alice's view. Thus, any private query must be "light" conditioned on his view. By definition, the probability that Bob hits a light query is small. We can then take a union bound over all of Bob's queries, and conclude that the total probability that Bob hits a private query is small.

Unfortunately, the intuition above isn't entirely correct: even querying an index that was not queried by Alice may give Bob information about Alice's view: the fact that Alice *didn't* query a particular index. We observe, however, that this is the *only* information about Alice's view that Bob can gain from making a non-intersection query. Thus, if we condition on Bob not having made any private intersection queries so far, our intuition still holds.

The main technical difficulty in the proof is making sure that Ivy can ask many queries in parallel, in order to bound the number of rounds of adaptivity. Our solution to this is to have Ivy condition her probability space after each query on the event that this query was *not an intersection query* (rather than on the response to the query). Since this event does not require Ivy to query the oracle in order to compute the new query probabilities, she can ask multiple parallel queries in each round. Loosely speaking, Ivy's query strategy ensures that if there are any remaining heavy queries, then one of her queries will be an intersection query with high probability. Since the number of intersection queries can be at most n, within n rounds Ivy can ensure that there are no remaining heavy queries.

Note that in order to find heavy queries, Ivy uses her unbounded computational power (e.g., if Alice queries the oracle on an index i and sends Bob an encryption of i, the index i is heavy conditioned on Ivy's view, but Ivy may have to break the encryption to find it).

2.2 Proof of Theorem 2

Javier, the adversary constructed in the proof of Theorem 2, works by running the honest solver in each round, but replacing its queries to the random oracle with a simulated oracle (so no queries to the real oracle are made during an execution). After the simulation, Javier updates the simulated oracle by querying the real oracle (in parallel) on every index that was queried during the simulated execution. The main idea in the proof is that, since the puzzle generator asks only n queries, there can be at most n rounds in which the simulated execution "hits" an intersection query that was not already known to Javier. In the remaining rounds, Javier does know all the intersection queries, and hence the simulated solver will behave just like the real honest solver (and output a correct solution to the puzzle with the same probability). Formally:

Proof (of Theorem 2). The adversary Javier follows Alg. 1. In the algorithm description, $Q_i(J)$ is the set of queries Javier made to H up to (but not including) round i, while $Q(B_i)$ is the set of queries the simulated Bob made to H_i in Javier's i^{th} round.

Algorithm 1 Javier's query algorithm on message M and oracle H with parameter ε

1: Randomly choose $i^* \in [n/\varepsilon]$.
2: **for** $i \in \{1, \ldots, i^*\}$ **do**
3: Run Bob to get: $x_i \leftarrow g^{H_i}(r_B, M)$ where H_i is an oracle that answers any query $x \in Q(J)$ the same as H does, and H_i answers any new query $q \notin Q(J)$ uniformly at random. Note that to run $g^{H_i}(r_B, M)$ we do not need to ask any new query to H because all the answers to queries in $Q(J)$ are already known and the rest are answered at random.
4: Query H on all indices in $Q(B_i) \setminus Q_i(J)$ where $Q(B_i)$ is the queries Bob made to H_i.
5: Output x_{i^*}.

The total number of queries made by Javier is at most nm/ε and Javier's running time is $O(n/\varepsilon)$ times the running time of Bob. It remains to show that the probability that Javier's output is accepted by the solution validator is at least $1 - \nu - \varepsilon$.

Denote the event that Javier's output is accepted by the solution validator:

$$\text{Success} \stackrel{def}{=} (M, \mathcal{V}) \leftarrow f^H(r_A) \wedge x_{i^*} \leftarrow g^{H_{i^*}}(r_B, M) \wedge \mathcal{V}(x_{i^*}) = 1 \ .$$

Call a round i *good* if Javier did not ask any new intersection queries in round i (i.e., $Q(B_i) \cap Q(A) \subseteq Q_i(J)$). Denote Good_i the event that round i was good. Since Alice asks at most n queries, there can be at most n rounds that are not good. Thus, $\Pr[\text{Good}_{i^*}] \geq 1 - \varepsilon$. Note that if Good_{i^*} holds, the tuple $(f^H(r_A), g^{H_{i^*}}(r_B, M))$ is distributed identically to $(f^H(r_A), g^H(r_B, M))$, because as long as Bob's queries in round i^* were not queried by Alice, H and H_{i^*} both choose their answers at random and independently of all previous queries and answers. Therefore, for an arbitrary event E defined over the joint view of $f^H(r_A)$ and that of Javier till the end of round i^*, to know the quantity $\Pr[E \wedge \text{Good}_{i^*}]$ it does not matter weather we use the oracle H or H_{i^*} in round i^*, and the probabilities will remain the same. By misusing the notation, we also use Good_{i^*} to refer to the similar event when Javier uses the oracle H in his simulation of Bob in round i^*. Thus, we finally conclude:

$$\begin{aligned}
\Pr[\text{Success}] &\geq \Pr[\text{Success} \wedge \text{Good}_{i^*}] \\
&= \Pr\left[(M, \mathcal{V}) \leftarrow f^H(r_A); x_{i^*} \leftarrow g^{H_{i^*}}(r_B, M); \mathcal{V}(x_{i^*}) = 1 \wedge \text{Good}_{i^*}\right] \\
&= \Pr\left[(M, \mathcal{V}) \leftarrow f^H(r_A); x \leftarrow g^H(r_B, M); \mathcal{V}(x) = 1 \wedge \text{Good}_{i^*}\right] \\
&\geq \Pr\left[(M, \mathcal{V}) \leftarrow f^H(r_A); x \leftarrow g^H(r_B, M); \mathcal{V}(x) = 1\right] - (1 - \Pr[\text{Good}_{i^*}]) \\
&\geq 1 - \nu - \varepsilon.
\end{aligned}$$

3 A Time-Lock Puzzle with a Linear Difficulty Gap

In this section we give the construction and proof for Theorem 3.

In the description below, we omit the security parameter k: the security parameter is only used to determine the range of the random oracle — we assume w.l.o.g. that $H(q)$ returns k bits (if the random oracle returns fewer bits, we can interpret a query $H(q)$ as concatenation of multiple queries (e.g., $H(kq) \odot H(kq+1) \odot \cdots \odot H(kq+k-1)$). To further simplify notation, our definition of f only generates the message M. The (implicit) solution validator $\mathcal{V}_M$ checks whether its input is equal to f's input (our soundness proof is slightly stronger — we show that no adversary making less than n serial rounds of queries to H can find *any* valid preimage of M under f).

We define the puzzle-generating function f to be:

$$f^H(x_0, \ldots, x_n) \stackrel{def}{=} (x_0, H(x_0) \oplus x_1, \ldots, H(x_{n-1}) \oplus x_n)$$

(where the input is interpreted as $n+1$ k-bit query indices).

The honest solver g inverts f by running Algorithm 2:

Proof (Sketch for Theorem 3). By inspection, f can be computed with n non-adaptive queries: the values $H(x_1), \ldots, H(x_n)$ can be obtained in parallel. The correctness of the honest inverter (Alg. 2) and the fact that it uses n serial queries is also easy to see.

Algorithm 2 Honest solver g on input $M = (M_0, \ldots, M_n)$ and oracle H

1: $x_0 \leftarrow M_0$ // x_0 is not "encrypted".
2: **for** $i \in \{1, \ldots, n\}$ **do**
3: $\quad x_i \leftarrow H(x_{i-1}) \oplus M_i$ // "decrypting" x_i requires an oracle query on index x_{i-1}.
4: Output $(x_0, \ldots, x_n)$

The main part of the proof is to show that every inverter making poly(k) queries to H needs to use at least n rounds of adaptive queries. To prove this, we first claim that any algorithm that outputs x_{i+1} with non-negligible probability must query H on x_i. This is because, even taken together, the value of $f^H(x_0, \ldots, x_n)$, the values of $\{x_0, \ldots, x_n\} \setminus \{x_{i+1}\}$ and the responses of the random oracle on all queries except x_i give no information (in the information-theoretic sense) about x_{i+1}. Thus, the probability that an algorithm outputs x_{i+1} without querying H on x_i is negligible in k (the output size of H). Note that this remains true if we allow the algorithm to output a polynomial number of guesses for x_{i+1}.

Now, consider an algorithm h making multiple rounds of queries to the oracle H, such that in each round the indices queried depend only on the responses from previous rounds. We can think of h as also outputting the indices it queries in each round (and the total number of indices output by h is polynomial in k). If h correctly inverts f on input $M = f^H(x_0, \ldots, x_n)$, it must output x_n at some round (since f is injective). By induction (and using the reasoning above), the probability that h first outputs (queries) x_i and x_{i+1} in the same round is negligible (since we showed it must query x_i before x_{i+1}). Therefore, the algorithm must use at least n rounds of adaptivity.

3.1 Increasing the Computation/Communication Ratio

Note that while our positive construction of a time-lock puzzle in the random-oracle model is optimal with respect to query complexity, the description of a puzzle that requires n adaptive queries to solve is also linear in n. When the cost of communication is comparable to an oracle query, simply communicating the puzzle takes $O(n)$ time, negating the benefit of parallel queries. We improve this ratio arbitrarily by replacing each oracle call with d composed calls (i.e., each call querying the oracle on the response to the previous call). This will increase both the (parallel) generation and solution time by a factor of d without changing the size of the puzzle description. Formally, let $H^{(1)} \stackrel{def}{=} H$ and for $d > 1$ let $H^{(d)}(q) \stackrel{def}{=} H(H^{(d-1)}(q))$. Then the function $f^{H^{(d)}}(x_0, \ldots, x_n)$ can be computed with d rounds of n non-adaptive queries, and the soundness condition from Theorem 3 holds with the increased parameters:

Claim. For every oracle function h that makes less than dn serial rounds of queries to H and poly(k) queries overall to H in total,

$$\Pr\left[(M, \mathcal{V}_M) \leftarrow f^{H^{(d)}}(k, r); x \leftarrow h^H(k, r_J, M); \mathcal{V}_M(x) = 1\right] = \mathrm{neg}(k)$$

(where neg is some negligible function in k).

Proof (Sketch). The main idea is that any algorithm that outputs $H^{(i)}(x)$ with non-negligible probability must query H on $H^{(i-1)}(x)$ (otherwise the algorithm has no information about $H^{(i)}(x)$). By induction, it follows that an algorithm that makes only a polynomial number (in k) of queries to H needs d adaptive rounds to compute $H^{(d)}(x)$. Composing this idea with the induction in the proof of Theorem 3, we get the required parameters.

4 Discussion and Open Questions

The most obvious open question relating to time-lock puzzles is finding constructions based on assumptions other than the difficulty of factoring. Although this work rules out black-box constructions (with a super-constant gap) from one-way permutations and collision-resistant hash functions, we have no reason to believe that time-lock puzzles based on other concrete problems (e.g., lattice-based problems) do not exist. Extending our approach to other general assumptions (e.g., trapdoor permutations) is also an interesting open problem.

One of the motivations for looking at time-lock puzzles in the random-oracle model is the search for puzzles that are resistant to quantum attack. In this direction there still remains work to be done: on the positive side, our construction may not be secure against adversaries with quantum access to the random oracle (e.g., Dagdelen et al. show protocols that are secure in the random oracle model but can be broken by attackers with quantum access to the random oracle [15]). On the other hand, when the honest parties are quantum, the lower bound question is still open as well (Brassard and Salvail [10] and, independently, Biham et al [5], give a quantum version of Merkle puzzles that require the adversary to make $n^{3/2}$ queries in order to recover the shared key, but do not prove optimality).

Acknowledgements

We thank the anonymous reviewers for their helpful comments and suggestions.

References

1. M. Abadi, M. Burrows, M. S. Manasse, and T. Wobber. Moderately hard, memory-bound functions. *ACM Trans. Internet Techn.*, 5(2):299–327, 2005.
2. A. Back. Hashcash — a denial of service counter-measure, 2002. `http://www.hashcash.org/papers/hashcash.pdf`.
3. B. Barak and M. Mahmoody. Lower bounds on signatures from symmetric primitives. In *FOCS '07*, pages 680–688. IEEE Computer Society.
4. B. Barak and M. Mahmoody. Merkle puzzles are optimal - an $O(n^2)$-query attack on any key exchange from a random oracle. In *CRYPTO '09*, volume 5677 of *LNCS*, pages 374–390.
5. E. Biham, Y. J. Goren, and Y. Ishai. Basing weak public-key cryptography on strong one-way functions. In *TCC '08*, volume 4948 of *LNCS*, pages 55–72.
6. D. Boneh and X. Boyen. Efficient selective-id secure identity-based encryption without random oracles. In *EUROCRYPT '04*, volume 3027 of *LNCS*, pages 223–238.

7. D. Boneh and X. Boyen. Secure identity based encryption without random oracles. In *CRYPTO '04*, volume 3152 of *LNCS*, pages 443–459.
8. D. Boneh and M. K. Franklin. Identity-based encryption from the weil pairing. *SIAM J. Comput.*, 32(3):586–615, 2003.
9. D. Boneh and M. Naor. Timed commitments. In *CRYPTO '00*, volume 1880 of *Lecture Notes in Computer Science*, pages 236–254.
10. G. Brassard and L. Salvail. Quantum merkle puzzles. In *ICQNM*, pages 76–79. IEEE Computer Society, 2008.
11. R. Canetti, O. Goldreich, and S. Halevi. The random oracle methodology, revisited. *J. ACM*, 51(4):557–594, 2004.
12. R. Canetti, S. Halevi, and J. Katz. A forward-secure public-key encryption scheme. In *EUROCRYPT '03*, volume 2656 of *LNCS*, pages 255–271.
13. J. Cathalo, B. Libert, and J.-J. Quisquater. Efficient and non-interactive timed-release encryption. In *ICICS*, volume 3783 of *LNCS*, pages 291–303, 2005.
14. G. D. Crescenzo, R. Ostrovsky, and S. Rajagopalan. Conditional oblivious transfer and timed-release encryption. In *EUROCRYPT '99*, volume 1592 of *LNCS*, pages 74–89.
15. O. Dagdelen, M. Fischlin, A. Lehmann, and C. Schaffner. Random oracles in a quantum world. Cryptology ePrint Archive, Report 2010/428, 2010. `http://eprint.iacr.org/2010/428.pdf`.
16. C. Dwork, A. Goldberg, and M. Naor. On memory-bound functions for fighting spam. In *CRYPTO '03*, volume 2729 of *LNCS*, pages 426–444.
17. C. Dwork and M. Naor. Pricing via processing or combatting junk mail. In *CRYPTO '92*, volume 740 of *LNCS*, pages 139–147.
18. C. Dwork, M. Naor, and H. Wee. Pebbling and proofs of work. In *CRYPTO '05*, volume 3621 of *LNCS*, pages 37–54.
19. R. Gennaro, Y. Gertner, J. Katz, and L. Trevisan. Bounds on the efficiency of generic cryptographic constructions. *SIAM J. Comput.*, 35(1):217–246, 2005.
20. L. K. Grover. A fast quantum mechanical algorithm for database search. In *STOC '96*, pages 212–219. ACM.
21. R. Impagliazzo and S. Rudich. Limits on the provable consequences of one-way permutations. In *STOC '89*, pages 44–61. ACM.
22. T. C. May. Timed-release crypto. `http://www.hks.net/cpunks/cpunks-0/1460.html`, February 1993.
23. R. C. Merkle. Secure communications over insecure channels. *Commun. ACM*, 21(4):294–299, 1978.
24. R. L. Rivest and A. Shamir. Payword and micromint: Two simple micropayment schemes. In *Security Protocols Workshop*, volume 1189 of *LNCS*, pages 69–87, 1996.
25. 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.