Time-Lock Puzzles in the Random Oracle Model

Mohammad Mahmoody*
mohammad@cs.cornell.edu

Tal Moran†
talm@seas.harvard.edu

Salil Vadhan‡
salil@seas.harvard.edu

Department of Computer Science
Cornell University

School of Engineering and Applied Sciences
and Center for Research on Computation and Society,
Harvard University

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 rules 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.

A natural approach to building a timed-release crypto scheme is the use of time-lock puzzles: puzzles that take a pre-specified 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

*http://www.cs.cornell.edu/~mohammad/

†http://seas.harvard.edu/~talm

‡http://seas.harvard.edu/~salil. Supported by NSF grant CNS-0831289.

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 the problem 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. 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. On one hand, we show how to construct 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. On the other hand we give a much simpler, efficient adversary construction (which runs in polynomial time if the honest solver does), but has slightly worse adaptivity (an additional advantage of the second construction is that its proof is much simpler).

Formally, we prove the following two theorems:

Theorem 1.1 (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 h that makes $\tilde{O}(nm/\varepsilon)$ queries to H, uses only n levels of adaptivity and satisfies $\Pr\left[(M, \mathcal{V}) \leftarrow f^H(r_A); x \leftarrow h^H(r_J, M); \mathcal{V}(x) = 1\right] \geq 1 - \nu - \varepsilon$ (where r_J is the variable denoting the random coins used by h).*

Theorem 1.2 (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 $\Pr\left[(M, \mathcal{V}) \leftarrow f^H(r_A); x \leftarrow g^H(r_B, M); \mathcal{V}(x) = 1\right] \geq 1 - \nu$ (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, there exists a deterministic adversary Javier (denoted by J) who asks n/ε rounds of adaptive queries with a total of at most $n \cdot m/\varepsilon$ queries in time $O(N \cdot T)$ where T is the running time of g, and it achieves completeness $\Pr\left[(M,\mathcal{V}) \leftarrow f^H(r_A); x \leftarrow J^H(M); \mathcal{V}(x) = 1\right] \geq 1-(\nu+\varepsilon)$.

(See Section 2 for a more detailed version of Theorem 1.2.)

The adversaries constructed in the proof of both the theorem attempt to find all *intersection queries* between the puzzle generator and an honest puzzle solver — all queries to the oracle that were made by both. If successful, the adversary can then simulate an honest solver without asking additional queries.

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.

Loosely speaking, Javier, the adversary constructed in the proof of Theorem 1.2, 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). The full proof of Theorem 1.2 appears in Section 2.

For the adversary of Theorem 1.1, we use ideas (and a construction) similar to those of Impagliazzo and Rudich[21] (and later developments by Barak and Mahmoody [4]). Our analysis is novel in that we attempt to minimize the number of query rounds used by the adversary and not just the total number of queries. (The analysis of [4] strongly uses the fact that the adversary can make its queries fully adaptively.) The full proof of Theorem 1.1 appears in Section 3.

By combining Theorem 1.1 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[1]:

Corollary 1.3. *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 $f^H(r_A, r_B)$ be the complete public transcript of Π when Alice uses the random coins r_A and Bob the random coins r_B. Think of f as the puzzle generator, and 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 1.1, this implies that there exists a solver that succeeds

[1] 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.

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 1.4. *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 4.

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 4, 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.

Organization Sections 2 and 3 contain our main (negative) results for general time-lock puzzles (Section 3 contains the proof of Theorem 1.1, and Section 2 the considerably simpler construction and proof of Theorem 1.2). In Section 4 we give a construction of a time-lock puzzle that satisfies the conditions of Theorem 1.4 and give a sketch of the proof. Finally, in Section 6 we describe some open problems. Note: to aid in following the proofs, a glossary of symbols is provided at the end of the paper (pg. 26).

2 Efficient but Non-Optimal Adversary

In this section we give the full proof for Theorem 1.2:

Theorem 1.2 (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 $\Pr\left[(M, \mathcal{V}) \leftarrow f^H(r_A); x \leftarrow g^H(r_B, M); \mathcal{V}(x) = 1\right] \geq 1 - \nu$ (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 each of the settings below, there exists a deterministic adversary Javier (denoted by J) who asks $N > n$ rounds of adaptive queries, asks at most $N \cdot m$ queries in time $O(N \cdot T)$ where T is the running time of g, and achieves the following bounds for the completeness error probability $\delta = 1 - \Pr\left[(M, \mathcal{V}) \leftarrow f^H(r_A); x \leftarrow J^H(M); \mathcal{V}(x) = 1\right]$.*

1. **Error close to ν:** *Javier asks $N = n/\varepsilon$ levels of adaptivity with completeness error $\delta \leq \nu + \varepsilon$.*
2. $2n$ **adaptivity:** *Javier asks $N = 2n$ levels of adaptivity with completeness error $\delta \leq (2n + 1) \cdot \nu$.*
3. $O(n)$ **adaptivity and** $O(\nu)$ **error:** *For arbitrary $\rho > 0$, Javier asks $N = 2(1 + \rho) \cdot n$ levels of adaptivity with completeness error $\delta \leq 2\nu \cdot (\rho + 1)/\rho$. For example, by letting $\rho = 1$, we get $N = 4n$, and $\delta \leq 4\nu$.*

Proof. We will first prove Part 1 of Theorem 1.2. We will then describe how to modify this attack to obtain Parts 1 and 2 throgh similar attacks with different analysis.

The adversary Javier (for Part 1) 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 as long as 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 whether 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:

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

□

Parts 2 and 3. The idea here is to use *majority* of the solutions obtained in *all* the emulations of the solver g instead of using a random one. In particular, we can use a similar algorithm to Algorithm 2 for both Parts 2 and 3 with N given as the adaptivity parameter, and use majority at the end.

To analyze the needed variants of Algorithm 2 (for Parts 2 and 3), first recall that because we have n queries during the puzzle generation, it holds that for $S = \{i \mid i \in [N+1], \neg\text{Good}_i\}$, it holds that $|S| \leq n$. We also need the following claim whose proof follows from the same argument used in the proof of Part 1.

Algorithm 2 Javier's query algorithm on message M and oracle H with adaptivity parameter $N \geq 2n$

1: **for** $i \in \{1, \ldots, N\}$ **do**
2: 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.
3: 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.
4: Make one more execution to get: $x_{N+1} \leftarrow g^{H_{N+1}}(r_B, M)$ where H_{N+1} is as defined above.
5: If there is x such that $|\{i \mid x_i = x\}| \geq (N+1)/2$ (i.e., there is a majority), output x, otherwise output $\perp$.

Claim 2.2. *For all $i \in [N+1]$, it holds that* $\Pr[\neg Good_i \vee Success] \geq 1 - \nu$. *Equivalently, it holds that* $\Pr[i \in T] \leq \nu$, *where* $T = \{i \mid i \in [N+1], Good_i \wedge Fail\}$.

The proof of the above claim again is based on the fact that during the i'th execution, so long as we do not hit any new private queries, we are indeed executing the solver with the right distribution of the oracle. Now, note that if $i \in [N+1] \setminus (S \cup T)$ then $x = x_i$.

For Part 2, by a union bound we have $\Pr[T \neq \emptyset] \leq (N+1)\nu$. So if we choose $N = 2n$, then we have at least $n+1$ index i in $[N+1] \setminus (S \cup T)$, and so the majority of x_i's will be correct.

For Part 3, we use a slight modification of Algorithm 2 where we skip Step 4 and simply use the majority of the N answers $x_1, \ldots, x_N$, and we adapt the definition of S, T to subsets of $[N]$. By the linearity of expectation, we have that $\mathbf{E}[|T|] \leq N \cdot \nu$. By Markov inequality, for any $\alpha > 0$, we have $\Pr[|T| \geq \alpha \cdot N] \leq \nu/\alpha$. For $\alpha = \rho/(2 \cdot (1+\rho))$, we get that

$$\Pr[|T| \geq \alpha \cdot N] \leq \frac{2\nu \cdot (1+\rho)}{\rho}.$$

Thus, all we have to prove is that whenever $|T| < \alpha \cdot N$, then we find the right answer. But, in that case:

$$|S \cup T| \leq |S| + |T| < n + \alpha \cdot N = n + \frac{N \cdot \rho}{2(1+\rho)} = n + \rho \cdot n = N/2.$$

In this case, since $x = x_i$ for all $i \in [N] \setminus (S \cup T)$ is correct, the majority would be the correct answer.

3 Tight Bound for Time-Lock Puzzle Generation/Solution Gap

In this section, we present the proof of Theorem 1.1, showing that if a time-lock puzzle can be generated with n queries to the random oracle and solved with m queries, then there exists a parallel solver that can find a solution in time $O(n)$ and makes only $\text{poly}(n, m)$ queries to the oracle.

Overview of Proof. Our proof is heavily based on ideas from Impagliazzo and Rudich [21] and Barak and Mahmoody [4]. Given a puzzle-generator f (Alice) and a solver g (Bob), we construct an adversary, Ivy, that finds all *intersection queries* to the random oracle (the queries made by both Alice and Bob). If Ivy is successful, we can run a simulation of Bob using the answers from the real oracle on the intersection queries and randomly chosen answers for all other queries. The simulation of Bob is distributed identically to an honest Bob (since the answers given by the "real" oracle to the other queries are independent of those seen by Alice). Thus, the simulation will output a valid solution to the puzzle with the same probability as an honest solver.

The main technical difficulty is bounding the number of queries (and the levels of adaptivity) used by Ivy. Impagliazzo and Rudich, and later Barak and Mahmoody, considered interactive protocols (where Alice and

Bob may exchange messages in multiple rounds). They bound the total number of queries an intersection-query-finding adversary must make to the random oracle as a function of n, the number of queries Alice makes to the random oracle, m, the number of queries made by Bob and ε (the probability that the adversary will miss an intersection query), but do not attempt to analyze the levels of adaptivity required.

We use similar techniques to show that, in the case of one-message protocols (as obtained from f and g), the total number of adaptive *rounds* of queries the adversary must use is at most n. Translating back to the setting of time-lock puzzles, this means a puzzle that requires n queries to generate can always be solved by making n rounds of queries (where the queries in each round depend only on the results of queries in previous rounds).

3.1 Oracle Query Processes

Consider a process that iteratively queries the random oracle $H\colon D \mapsto R$ and determines its next query by some arbitrary function of the responses up to that point. We call such a process an Oracle-Query (OQ) process. An OQ process F is defined by an initial state $V_0 = V_0(F) \in \mathcal{I}$ (which is a random variable that may, in general, be correlated with H) and its sequence of "next query" functions $F_i\colon \mathcal{I} \times R^i \mapsto D$. The *view* of the process after the i^{th} iteration, denoted $V_i(F)$ is a random variable consisting of the vector of query answers seen up to that point (we will omit F if the process is clear from the context):

$$V_i = V_{i-1} \odot H(F_i(V_{i-1}))\ ,$$

where $\odot$ denotes vector concatenation (i.e., $(x_1, \ldots, x_n) \odot y \stackrel{def}{=} (x_1, \ldots, x_n, y)$) and $V_0(F)$ represents the input and random coins of the process F. We denote $Q_i(F) \stackrel{def}{=} (F_1(V_0), \ldots, F_i(V_{i-1}))$ the vector of queries asked by F up to iteration i.

For a vector of queries $S = (s_1, \ldots, s_j)$, we denote $H(S) \stackrel{def}{=} (H(s_1), \ldots, H(s_j))$ the corresponding vector of oracle responses to the queries in S. To simplify notation, when the order of entries in a vector doesn't matter, we will sometimes use set notation to refer to it.

In the context of our puzzle protocol, Alice, Bob and the adversary can be thought of as OQ processes. In the analysis of the adversary, we will consider them together as a single OQ process (whose input is the combined random coins of Alice, Bob and the adversary). We rely in the analysis on the fact that the answers of the random oracle for indices that have not been queried yet are independent of any previous view. Formally, we will use the following claim:

Claim 3.1. *Let F be an OQ process such that $V_0(F)$ is independent of the random oracle H. For all query sets $S \subseteq T$, every view v, answer vector s and $i \geq 0$ such that* $\Pr[V_i = v \wedge T \cap Q_i = \emptyset] > 0$,

$$\Pr[H(S) = s \mid V_i = v \wedge T \cap Q_i = \emptyset] = \Pr[H(S) = s]\ .$$

Proof. We can think of the oracle H as performing "lazy-evaluation": the answer to each query is determined by flipping new, independent, coins when the query is asked for the first time. Thus, if an OQ process did not query the oracle on a set of indices, the oracle responses on those indices are independent of the process view. Since the set of queries made by the process can be computed from its view, and $\Pr[V_i = v \wedge T \cap Q_i = \emptyset] > 0$, it must be the case that the view v does not contain queries to any of the indices in $T \supseteq S$. Hence, $H(S)$ is independent of the event $V_i = v$ (and the event it implies, $T \cap Q_i = \emptyset$). □

Corollary 3.2. *Let F be an OQ process such that $V_0(F)$ is independent of the random oracle and let G be an arbitrary function of its view. For all query sets $S \subseteq T$, every g in the range of G, answer vector s and $i \geq 0$ such that* $\Pr[G(V_i) = g \wedge T \cap Q_i = \emptyset] > 0$,

$$\Pr[H(S) = s \mid G(V_i) = g \wedge T \cap Q_i = \emptyset] = \Pr[H(S) = s]\ .$$

Proof. By the law of total probability:

$$\begin{aligned}
&\Pr\left[H(S) = s \mid G(V_i) = g \wedge T \cap Q_i = \emptyset\right] \\
&= \sum_{v|G(v)=g} \Pr\left[H(S) = s \mid G(V_i) = g \wedge T \cap Q_i = \emptyset \wedge V_i = v\right] \\
&\qquad \cdot \Pr\left[V_i = v \mid G(V_i) = g \wedge T \cap Q_i = \emptyset\right] \\
&= \sum_{v|G(v)=g} \Pr\left[H(S) = s \mid T \cap Q_i = \emptyset \wedge V_i = v\right] \qquad (3.1) \\
&\qquad \cdot \Pr\left[V_i = v \mid G(V_i) = g \wedge T \cap Q_i = \emptyset\right] \\
&= \Pr\left[H(S) = s\right] \cdot \sum_{v|G(v)=g} \Pr\left[V_i = v \mid G(V_i) = g \wedge T \cap Q_i = \emptyset\right] \qquad (3.2) \\
&= \Pr\left[H(S) = s\right] \ ,
\end{aligned}$$

where (3.1) is because $V_i = v$ implies $G(V_i) = g$ and (3.2) follows from Claim 3.1. □

3.2 Attack and Analysis

3.2.1 Overview

Our attacker, Ivy, selects her queries to the random oracle in n rounds. 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 of having been made by Alice (where "high" is a parameter that depends on n, m and ε).

The intuition behind Ivy's attack, at a high level, 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.

Note that 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 — the response of the oracle to the query does not give any additional information (this is a general property of OQ processes that is captured in Claim 3.1). Thus, if we condition on Bob not having made any private intersection queries so far, our intuition still holds. By lowering Ivy's "heaviness threshold" we can ensure that for each query Bob makes, if Ivy hasn't also made the query then it is a non-intersection query with "high probability" (conditioned on Bob's view); The probability is high enough that this additional information will not change the distribution of Alice's queries according to Bob too much (compared to Ivy's view), so the same argument will continue to work for Bob's successive queries.

Ivy's algorithm is very similar to that of the adversary of Barak and Mahmoody (who we will call Eve). Eve iteratively determines the heavy queries by conditioning her probability space after each query on her entire view so far. Thus, the levels of adaptivity Eve uses is bounded by her *total number* of queries to the oracle. In our work, the point is to bound the number of rounds of adaptivity. To do this, we must modify the attack to let Ivy ask many queries in parallel.

Our solution to this is to have Ivy condition her probability space after each query only on the event that this query was *not an intersection query*, rather than on the response of the oracle to that 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.

3.2.2 Notation

Below we formally describe the game played by Alice, Bob and Ivy (the adversary). All three can be thought of as OQ processes, with next-query functions $\{A_i\}$, $\{B_i\}$ and $\{I_i\}$ respectively. Since Ivy makes queries non-adaptively, we will abuse the notation somewhat: $V_j(I)$ (resp. $Q_j(I)$) will refer to Ivy's view (resp. queries made) after the j^{th} *round* of queries (rather than the j^{th} individual query). When we refer to the views or query vectors without a subscript, we mean the entire view (resp. query vector).

Probabilities are always taken to be over the space of full executions of the protocol; the randomness consists of the parties' random coins and the coins of the random oracle, H.

M will be the random variable whose value is the message sent by Alice ($(M, \mathcal{V}) \leftarrow f^H(r_A)$). This message is given to Bob and Ivy as input: $V_0(B) = (r_B, M)$ and $V_0(I) = (M)$. (Ivy is deterministic and has no random coins.) Note that the inputs of Bob and Ivy are not independent of H (since M may depend on H). We denote by $q_i(B)$ the i^{th} query asked by Bob (i.e., $Q(B) = (q_1(B), \ldots, q_m(B))$).

We say q is an *intersection query* if $q \in Q(A) \cap Q(B)$ (i.e., q is queried by both Alice and Bob). We say q is a *private query* if q is an intersection query that is not made by Ivy (i.e., $q \in Q(A) \cap Q(B) \setminus Q(I)$). We define $\text{Fail}^{(j)}$ to be the event that after Ivy's j^{th} round of queries to the random oracle, she has not managed to "hit" all the intersection queries:

$$\text{Fail}^{(j)} \stackrel{def}{=} \left[Q(B) \cap Q(A) \not\subseteq Q_j(I) \right] .$$

Denote

$$\text{Fail} \stackrel{def}{=} \text{Fail}^n .$$

Let ε be the error parameter (we will allow Ivy to fail with probability ε) and

$$\gamma \stackrel{def}{=} \varepsilon/(2em)$$

a "heaviness threshold" used by Ivy in her attack algorithm (e is the Euler constant).

We denote $\mathcal{Q}_j$ the set of queries that Ivy makes to the random oracle in the j^{th} round, and let

$$\text{Miss}_j \stackrel{def}{=} \left[Q_j(I) \cap Q(A) = Q_{j-1}(I) \cap Q(A) \right]$$

be the event that $Q_j(I) \setminus Q_{j-1}(I)$ does not contain any queries of Alice. Let

$$\text{NoMiss}_j \stackrel{def}{=} \left[\bigwedge_{k=1}^{j} \neg \text{Miss}_k \right]$$

denote the event that Ivy hits at least one more of Alice's queries in each of the first j rounds and

$$\text{FirstMiss}_j \stackrel{def}{=} \left[\text{Miss}_j \wedge \text{NoMiss}_{j-1} \right]$$

denote the event that round j is the first at which Ivy misses all of Alice's queries.

3.2.3 The Attack

Ivy attempts to "hit" queries that were made by Alice by including in Q_j all the queries that Alice is likely to have made, conditioned on Ivy's view to that point and on having missed all the previous queries in Q_j. Ivy continues adding heavy queries to Q_j until none are left, or until the probability that all the queries miss (i.e., that the event Miss_j occurs) is small enough. Since Alice makes at most n queries and Ivy's algorithm has n rounds, either there exists at least one round for which Miss_j occurs or over all rounds Ivy makes all of Alice's queries (in which case she must also have made all intersection queries).

The formal description of Ivy's algorithm is as follows:

Algorithm 3 Ivy's query algorithm on message M and oracle H

$v_I^{(0)} \leftarrow (M)$ // Ivy's initial state consists of the message M sent by Alice
for $j \in \{1, \ldots, n\}$ **do** // j is the adaptivity level
  Let $Q_j \leftarrow \emptyset$. // Q_j is the set of planned queries for adaptivity level j
  while $\exists q$ such that
    $\Pr\left[q \in Q(A) \setminus (Q_j \cup Q_{j-1}(I)) \mid Q_j \cap Q(A) = \emptyset \wedge \text{NoMiss}_{j-1} \wedge V_{j-1}(I) = v_I^{(j-1)}\right] \geq \gamma$ **do**
    Let q be the lowest-indexed query that satisfies the condition.
    $Q_j \leftarrow Q_j \odot q$.
    if $\Pr\left[Q_j \cap Q(A) = \emptyset \wedge \text{NoMiss}_{j-1} \mid V_{j-1}(I) = v_I^{(j-1)}\right] < \varepsilon/2n$ **then**
      break // Exit while loop
  Query the random oracle on Q_j.
  $v_I^{(j)} \leftarrow v_I^{(j-1)} \odot H(Q_j)$.

We claim that Ivy is successful in asking all the intersection queries (except with small probability) and that Ivy does not ask too many queries or use too many levels of adaptivity. The former is captured by Lemma 3.3 and the latter by Lemma 3.7.

3.2.4 Success of the attack

Lemma 3.3. *The probability that there is an intersection query that is not queried by Ivy is at most ε.*

We break up the proof of the lemma into several claims, all of which use the following notation:

Let $\text{Good}_i^{(j)}$ be the event that the first i queries made by Bob are either not intersection queries or were queried by Ivy in her first j rounds (i.e., $\{q_1(B), \ldots, q_i(B)\} \cap Q(A) \subseteq Q_j(I)$). Thus, $\text{Good}_0^{(j)}$ is always true. Let $\text{Fail}_i^{(j)}$ be the event that Bob's i^{th} query is the first intersection query that was missed by Ivy in her first j rounds (i.e., $\text{Good}_{i-1}^{(j)}$ holds and $q_i(B) \in Q(A) \setminus Q_j(I)$).

Intuitively, the following claim says that conditioned on FirstMiss_j and $\text{Good}_i^{(j)}$, knowing the answer to Bob's i^{th} query doesn't affect the heaviness of any query.

Claim 3.4. *For every query q, all $j \in \{1, \ldots, n\}$, $i \in \{1, \ldots, m-1\}$ and all views $v_I^{(j-1)}, v_B^{(i)}$ such that $\Pr\left[\text{FirstMiss}_j \wedge V_{j-1}(I) = v_I^{(j-1)} \wedge \text{Good}_i^{(j)} \wedge V_i(B) = v_B^{(i)}\right] > 0$ it holds that*

$$\begin{aligned}&\Pr\left[q \in Q(A) \setminus Q_j(I) \mid \text{FirstMiss}_j \wedge V_{j-1}(I) = v_I^{(j-1)} \wedge \text{Good}_i^{(j)} \wedge V_i(B) = v_B^{(i)}\right] \\ &= \Pr\left[q \in Q(A) \setminus Q_j(I) \mid \text{FirstMiss}_j \wedge V_{j-1}(I) = v_I^{(j-1)} \wedge \text{Good}_i^{(j)} \wedge V_{i-1}(B) = v_B^{(i-1)}\right]\end{aligned}$$

Proof. Denote $v_B^{(i)} = v_B^{(i-1)} \odot z$ (i.e., let z be the answer to the i^{th} oracle query in the view described by $v_B^{(i)}$). Note that $q_i(B)$ is a deterministic function of $V_{i-1}(B)$, so after conditioning on $V_{i-1}(B) = v_B^{(i-1)}$ it is a

constant. Denote this constant q' and Let $W \stackrel{def}{=} \left[\text{FirstMiss}_j \wedge V_{j-1}(I) = v_I^{(j-1)} \wedge \text{Good}_i^{(j)} \wedge V_{i-1}(B) = v_B^{(i-1)}\right]$. Then,

$$\begin{aligned}&\Pr\left[q \in Q(A) \setminus Q_j(I) \mid \text{FirstMiss}_j \wedge V_{j-1}(I) = v_I^{(j-1)} \wedge \text{Good}_i^{(j)} \wedge V_i(B) = v_B^{(i)}\right]\\ &= \Pr\left[q \in Q(A) \setminus Q_j(I) \mid W \wedge H(q') = z\right]\end{aligned}$$

Since $Q_j(I)$ is a deterministic function of $V_{j-1}(I)$, it is also a constant when conditioning on $V_{j-1}(I) = v_I^{(j-1)}$. We proceed by considering the following cases:

Case 1: $q' \in Q_{j-1}(I)$. In this case, the event $H(q') = z$ is implied by the event W. Indeed, if

$$\Pr\left[\text{FirstMiss}_j \wedge V_{j-1}(I) = v_I^{(j-1)} \wedge \text{Good}_i^{(j)} \wedge V_i(B) = v_B^{(i)}\right] > 0\,,$$

then the oracle's response to query q' given by $v_B^{(i)}$ must be consistent with its response in $v_I^{(j-1)}$). Thus,

$$\begin{aligned}&\Pr\left[q \in Q(A) \setminus Q_j(I) \mid W \wedge H(q') = z\right]\\ &= \Pr\left[q \in Q(A) \setminus Q_j(I) \mid W\right]\\ &= \Pr\left[q \in Q(A) \setminus Q_j(I) \mid \text{FirstMiss}_j \wedge V_{j-1}(I) = v_I^{(j-1)} \wedge \text{Good}_i^{(j)} \wedge V_{i-1}(B) = v_B^{(i-1)}\right]\end{aligned}$$

Case 2: $q' \in Q_j(I) \setminus Q_{j-1}(I)$ or $q' \notin Q_j(I)$. In the first case, the event FirstMiss_j implies that $q' \notin Q(A)$. In the second case, the event $\text{Good}_i^{(j)}$ implies that $q' \notin Q(A)$. Since all of Bob's queries are distinct, $V_{i-1}(B) = v_B^{(i-1)}$ implies $q' \notin Q_{i-1}(B)$. Thus, W implies $q' \notin Q(A) \cup Q_{i-1}(B) \cup Q_{j-1}(I)$. Applying Bayes rule, we can write:

$$\begin{aligned}&\Pr\left[q \in Q(A) \setminus Q_j(I) \mid W \wedge H(q') = z\right]\\ &= \Pr\left[q \in Q(A) \setminus Q_j(I) \mid W\right] \cdot \frac{\Pr\left[H(q') = z \mid W \wedge q \in Q(A) \setminus Q_j(I)\right]}{\Pr\left[H(q') = z \mid W\right]}\\ &= \Pr\left[q \in Q(A) \setminus Q_j(I) \mid \text{FirstMiss}_j \wedge V_{j-1}(I) = v_I^{(j-1)} \wedge \text{Good}_i^{(j)} \wedge V_{i-1}(B) = v_B^{(i-1)}\right]\\ &\quad \cdot \frac{\Pr\left[H(q') = z \mid W \wedge q \in Q(A) \setminus Q_j(I)\right]}{\Pr\left[H(q') = z \mid W\right]}\,.\end{aligned}$$

Thus, to prove the claim it is enough to show that

$$\Pr\left[H(q') = z \mid W \wedge q \in Q(A) \setminus Q_j(I)\right] = \Pr\left[H(q') = z \mid W\right]\,.$$

To see this, note that FirstMiss_j is a deterministic function of $V(A)$ and $V_{j-1}(I)$ and $\text{Good}_i^{(j)}$ is a deterministic function of $V_{i-1}(B)$, $V_{j-1}(I)$ and $V(A)$, hence W is a deterministic function of $V_{i-1}(B)$, $V_{j-1}(I)$ and $V(A)$, as is the event $W \wedge q \in Q(A) \setminus Q_j(I)$. Consider a hypothetical OQ process Y that first simulates Alice with random coins r_A to compute her output $(M, \mathcal{V})$, then simulates Bob for $i-1$ queries using Alice's output and random coins r_B and finally simulates Ivy for $j-1$ rounds. $V(Y) = V(A) \odot V_{i-1}(B) \odot V_{j-1}(I)$ and $Q(Y) = Q(A) \cup Q_{i-1}(B) \cup Q_{j-1}(I)$. We can think of both W and

$W \wedge q \in Q(A) \setminus Q_j(I)$ as deterministic functions of $V(Y)$. Moreover, $V_0(Y) = (r_A, r_B)$ is independent of the random oracle, thus by Corollary 3.2 (taking $S = T = \{q'\}$) we have

$$\begin{aligned}
\Pr\left[H(q') = z \mid W\right] &= \Pr\left[H(q') = z \mid W \wedge q' \notin Q(Y)\right] \\
&= \Pr\left[H(q') = z\right] \\
&= \Pr\left[H(q') = z \mid W \wedge q \in Q(A) \setminus Q_j(I) \wedge q' \notin Q(Y)\right] \\
&= \Pr\left[H(q') = z \mid W \wedge q \in Q(A) \setminus Q_j(I)\right]
\end{aligned}$$

□

Using the previous claim, we can show that if, conditioned on Ivy's view and FirstMiss_j, all queries that have not already been made by Ivy are "light", then the probability that Ivy has missed an intersection query at round j is small:

Claim 3.5. *Let* $0 < \varepsilon < 1$*,* $j \in \{1, \ldots, n\}$ *and* $v_I^{(j-1)}$ *be a view such that* $\Pr\left[\mathit{FirstMiss}_j \wedge V_{j-1}(I) = v_I^{(j-1)}\right] > 0$*. If for every query* q *it holds that*

$$\Pr\left[q \in Q(A) \setminus Q_j(I) \mid \mathit{FirstMiss}_j \wedge V_{j-1}(I) = v_I^{(j-1)}\right] < \gamma \leq \frac{\varepsilon}{2em}$$

then $\Pr\left[\mathit{Fail}^{(j)} \mid \mathit{FirstMiss}_j \wedge V_{j-1}(I) = v_I^{(j-1)}\right] < \varepsilon/2$*.*

Proof. Since $\text{Fail}^{(j)} = \biguplus_{i=1}^{m} \text{Fail}_i^{(j)}$, we can write

$$\Pr\left[\text{Fail}^{(j)} \mid \text{FirstMiss}_j \wedge V_{j-1}(I) = v_I^{(j-1)}\right] = \sum_{i=1}^{m} \Pr\left[\text{Fail}_i^{(j)} \mid \text{FirstMiss}_j \wedge V_{j-1}(I) = v_I^{(j-1)}\right] .$$

Let $\delta = 1/(m+1)$, $i > k$ and $v_B^{(k)} = ((r_B, M), t_1, \ldots, t_k)$ be a potential view of Bob. We will prove by induction on i that, for every $i \in \{1, \ldots, m\}$ and all views $v_I^{(j-1)}, v_B^{(i-1)}$ such that

$$\Pr\left[\text{FirstMiss}_j \wedge V_{j-1}(I) = v_I^{(j-1)} \wedge \text{Good}_{i-1}^{(j)} \wedge V_{i-1}(B) = v_B^{(i-1)}\right] > 0 ,$$

the following conditions hold:

1. For all q, q is light:

$$\Pr\left[q \in Q(A) \setminus Q_j(I) \mid \text{FirstMiss}_j \wedge V_{j-1}(I) = v_I^{(j-1)} \wedge \text{Good}_{i-1}^{(j)} \wedge V_{i-1}(B) = v_B^{(i-1)}\right] < \frac{\gamma}{(1-\delta)^{i-1}} .$$

2. The probability that Bob's i^{th} query is the first missed intersection query is small:

$$\Pr\left[\text{Fail}_i^{(j)} \mid \text{FirstMiss}_j \wedge V_{j-1}(I) = v_I^{(j-1)} \wedge \text{Good}_{i-1}^{(j)} \wedge V_{i-1}(B) = v_B^{(i-1)}\right] < \frac{\gamma}{(1-\delta)^{i-1}}$$

For $i = 1$, condition 1 is the same as the precondition of the claim except that we also conditioning on $V_0(B) = v_B^{(0)}$. Since we are already conditioning on $v_I^{(0)}$ which is equal to Alice's output M, $V_0(B) = (r_B, M)$ is independent of both Alice and Ivy's views and conditioning on it does not affect the probability that $q \in Q(A) \setminus Q_j(I)$. Condition 2 follows from condition 1, since $q_1(B)$ is a deterministic function of $V_0(B)$ and $\text{Fail}_1^{(j)}$ is the event $q_1(B) \in Q(A) \setminus Q_j(I)$.

Assume all the conditions hold up to some $i \in \{1, \ldots, m-1\}$. By Claim 3.4, to show that condition 1 is satisfied for $i+1$ it is sufficient to prove that

$$\Pr\left[q \in Q(A) \setminus Q_j(I) \mid \text{FirstMiss}_j \wedge V_{j-1}(I) = v_I^{(j-1)} \wedge \text{Good}_i^{(j)} \wedge V_{i-1}(B) = v_B^{(i-1)}\right] < \frac{\gamma}{(1-\delta)^i} .$$

Note that this doesn't follow immediately from the induction hypothesis, since we are conditioning on $\text{Good}_i^{(j)}$ rather than $\text{Good}_{i-1}^{(j)}$.

To simplify notation, let $Z \stackrel{def}{=} \left[\text{FirstMiss}_j \wedge V_{j-1}(I) = v_I^{(j-1)} \wedge V_{i-1}(B) = v_B^{(i-1)}\right]$. Since $\text{Good}_i^{(j)}$ implies $\text{Good}_{i-1}^{(j)}$, for all q it holds that:

$$\begin{aligned}
&\Pr\left[q \in Q(A) \setminus Q_j(I) \mid \text{FirstMiss}_j \wedge V_{j-1}(I) = v_I^{(j-1)} \wedge \text{Good}_i^{(j)} \wedge V_{i-1}(B) = v_B^{(i-1)}\right] \\
&= \Pr\left[q \in Q(A) \setminus Q_j(I) \mid Z \wedge \text{Good}_{i-1}^{(j)} \wedge \text{Good}_i^{(j)}\right] \\
&\leq \frac{\Pr\left[q \in Q(A) \setminus Q_j(I) \mid Z \wedge \text{Good}_{i-1}^{(j)}\right]}{\Pr\left[\text{Good}_i^{(j)} \mid Z \wedge \text{Good}_{i-1}^{(j)}\right]} \\
&< \frac{\gamma}{(1-\delta)^{i-1}} \cdot \frac{1}{\Pr\left[\text{Good}_i^{(j)} \mid Z \wedge \text{Good}_{i-1}^{(j)}\right]}
\end{aligned}$$

where the first inequality is because for all events A and B, $\Pr[A|B] \leq \Pr[A] / \Pr[B]$, and the second is due to condition 1 of the induction hypothesis. Since $\text{Good}_i^{(j)}$ is equivalent to $\neg\text{Fail}_i^{(j)} \wedge \text{Good}_{i-1}^{(j)}$,

$$\begin{aligned}
\Pr\left[\text{Good}_i^{(j)} \mid Z \wedge \text{Good}_{i-1}^{(j)}\right] &= 1 - \Pr\left[\text{Fail}_i^{(j)} \mid Z \wedge \text{Good}_{i-1}^{(j)}\right] \\
&> 1 - \frac{\gamma}{(1-\delta)^{i-1}} \\
&\geq 1 - \delta
\end{aligned}$$

where the first inequality is due to condition 2 of the induction hypothesis and the second follows from the fact that $i \leq m$, the inequality $(1 - 1/(m+1))^m \geq e^{-1}$ and our choices of $\delta = 1/(m+1)$ and $\gamma \leq \delta/e$. Therefore,

$$\begin{aligned}
&\Pr\left[q \in Q(A) \setminus Q_j(I) \mid \text{FirstMiss}_j \wedge V_{j-1}(I) = v_I^{(j-1)} \wedge \text{Good}_i^{(j)} \wedge V_i(B) = v_B^{(i)}\right] \\
&= \Pr\left[q \in Q(A) \setminus Q_j(I) \mid Z \wedge \text{Good}_i^{(j)}\right] \\
&< \frac{\gamma}{(1-\delta)^{i-1}} \cdot \frac{1}{1-\delta} = \frac{\gamma}{(1-\delta)^i} ,
\end{aligned} \tag{3.3}$$

hence condition 1 holds for $i+1$. We now show that condition 2 for $i+1$ follows from condition 1 for $i+1$. Note that $q_{i+1}(B)$ is a deterministic function of $V_i(B)$. Hence, conditioned on $V_i(B) = v_B^{(i)}$, $q_{i+1}(B)$ is a constant q. Thus,

$$\begin{aligned}
&\Pr\left[\text{Fail}_{i+1}^{(j)} \mid \text{FirstMiss}_j \wedge V_{j-1}(I) = v_I^{(j-1)} \wedge V_i(B) = v_B^{(i)} \wedge \text{Good}_i^{(j)}\right] \\
&= \Pr\left[q_{i+1}(B) \in Q(A) \setminus Q_j(I) \mid \text{FirstMiss}_j \wedge V_{j-1}(I) = v_I^{(j-1)} \wedge V_i(B) = v_B^{(i)} \wedge \text{Good}_i^{(j)}\right] \\
&< \frac{\gamma}{(1-\delta)^i}
\end{aligned}$$

where the inequality is due to condition 1.

Finally, to bound the probability of $\text{Fail}^{(j)}$ conditioned on $\text{FirstMiss}_j \wedge V_{j-1}(I) = v_I^{(j-1)}$:

$$\begin{aligned}
&\Pr\left[\text{Fail}^{(j)} \mid \text{FirstMiss}_j \wedge V_{j-1}(I) = v_I^{(j-1)}\right] \\
&= \sum_{i=1}^{m} \Pr\left[\text{Fail}_i^{(j)} \mid \text{FirstMiss}_j \wedge V_{j-1}(I) = v_I^{(j-1)}\right] \\
&\leq \sum_{i=1}^{m} \Pr\left[\text{Fail}_i^{(j)} \mid \text{FirstMiss}_j \wedge V_{j-1}(I) = v_I^{(j-1)} \wedge \text{Good}_{i-1}^{(j)}\right] \\
&= \sum_{i=1}^{m} \mathop{\mathbf{E}}_{v_B^{(i-1)}}\left[\Pr\left[\text{Fail}_i^{(j)} \mid \text{FirstMiss}_j \wedge V_{j-1}(I) = v_I^{(j-1)} \wedge \text{Good}_{i-1}^{(j)} \wedge V_{i-1}(B) = v_B^{(i-1)}\right]\right] \\
&< \sum_{i=1}^{m} \frac{\gamma}{(1-\delta)^{i-1}} \leq m\gamma e \leq \frac{\varepsilon}{2}\,,
\end{aligned}$$

where the first inequality is because $\text{Fail}_i^{(j)}$ implies $\text{Good}_{i-1}^{(j)}$ the expectation is over all of Bob's possible views ($V_{i-1}(B) = v_B^{(i-1)}$) and the second inequality is due to condition 2 and of the induction claim. □

We use Claim 3.5 to show that for every round j, the probability that Ivy fails for the first time at round j conditioned on missing all of Alice's queries at that round is low:

Claim 3.6. *For every $j \in \{1, \ldots, n\}$ and every view $v_I^{(j-1)}$ such that* $\Pr[\mathit{FirstMiss}_j \mid V_{j-1}(I) = v_I^{(j-1)}] \geq \varepsilon/2n$, $\Pr[\mathit{Fail}^{(j)} \mid \mathit{FirstMiss}_j \wedge V_{j-1}(I) = v_I^{(j-1)}] < \varepsilon/2$

Proof. Fix j and $v_I^{(j-1)}$. Since $\Pr\left[\text{FirstMiss}_j \mid V_{j-1}(I) = v_I^{(j-1)}\right] \geq \varepsilon/2n$, the condition in step 6 did not hold at the end of Ivy's j^{th} round so the while condition (in step 4) must not hold. Thus,

$$\forall q : \Pr\left[q \in Q(A) \setminus (Q_j \cup Q_{j-1}(I)) \mid \text{FirstMiss}_j \wedge V_{j-1}(I) = v_I^{(j-1)}\right] < \gamma\,.$$

Since $Q_j(I) = Q_j \cup Q_{j-1}(I)$, we can rewrite this as:

$$\forall q : \Pr\left[q \in Q(A) \setminus Q_j(I) \mid \text{FirstMiss}_j \wedge V_{j-1}(I) = v_I^{(j-1)}\right] < \gamma\,.$$

Noting that we set $\gamma = \varepsilon/(2em)$, the claim's statement follows from Claim 3.5. □

Finally, we can use Claim 3.6 to complete the proof of the lemma (that is, to show that $\Pr[\text{Fail}] \leq \varepsilon$):

Proof of Lemma 3.3. Note that Fail implies that FirstMiss_j occurred for some $j \in \{1, \ldots, n\}$ (otherwise Ivy

hit all n of Alice's queries, hence all intersection queries). Thus,

$$\begin{aligned}
\Pr[\text{Fail}] &= \sum_{j=1}^{n} \Pr\left[\text{Fail} \wedge \text{FirstMiss}_j\right] \\
&\leq \sum_{j=1}^{n} \Pr\left[\text{Fail}^{(j)} \wedge \text{FirstMiss}_j\right] \\
&= \sum_{j=1}^{n} \sum_{v_I^{(j-1)}} \Pr\left[V_{j-1}(I) = v_I^{(j-1)}\right] \Pr\left[\text{Fail}^{(j)} \wedge \text{FirstMiss}_j \mid V_{j-1}(I) = v_I^{(j-1)}\right] \\
&= \sum_{j=1}^{n} \sum_{v_I^{(j-1)}} \Pr\left[V_{j-1}(I) = v_I^{(j-1)}\right] \Pr\left[\text{FirstMiss}_j \mid V_{j-1}(I) = v_I^{(j-1)}\right] \Pr\left[\text{Fail}^{(j)} \mid \text{FirstMiss}_j \wedge V_{j-1}(I) = v_I^{(j-1)}\right] \\
&= \sum_{(j, v_I^{(j-1)})} \Pr\left[V_{j-1}(I) = v_I^{(j-1)}\right] \Pr\left[\text{FirstMiss}_j \mid V_{j-1}(I) = v_I^{(j-1)}\right] \Pr\left[\text{Fail}^{(j)} \mid \text{FirstMiss}_j \wedge V_{j-1}(I) = v_I^{(j-1)}\right],
\end{aligned}$$

where the inequality is because Fail implies $\text{Fail}^{(j)}$ for all j and the final sum is over all pairs $(j, v_I^{(j-1)})$ such that $j \in \{1, \ldots, n\}$ and $\Pr\left[\text{FirstMiss}_j \wedge V_{j-1}(I) = v_I^{(j-1)}\right] > 0$.

We can partition the set of pairs $(j, v_I^{(j-1)})$ into pairs for which it holds that $\Pr\left[\text{FirstMiss}_j \mid V_{j-1}(I) = v_I^{(j-1)}\right] < \varepsilon/2n$ and those for which it does not hold. Denote this first set

$$S \stackrel{def}{=} \left\{(j, v_I^{(j-1)}) \mid \Pr\left[\text{FirstMiss}_j \mid V_{j-1}(I) = v_I^{(j-1)}\right] < \frac{\varepsilon}{2n}\right\}.$$

Then,

$$\begin{aligned}
&\sum_{(j, v_I^{(j-1)}) \in S} \Pr\left[V_{j-1}(I) = v_I^{(j-1)}\right] \Pr\left[\text{FirstMiss}_j \mid V_{j-1}(I) = v_I^{(j-1)}\right] \Pr\left[\text{Fail}^{(j)} \mid \text{FirstMiss}_j \wedge V_{j-1}(I) = v_I^{(j-1)}\right] \\
\leq &\sum_{(j, v_I^{(j-1)}) \in S} \Pr\left[V_{j-1}(I) = v_I^{(j-1)}\right] \cdot \frac{\varepsilon}{2n} \\
\leq &\sum_{(j, v_I^{(j-1)})} \Pr\left[V_{j-1}(I) = v_I^{(j-1)}\right] \cdot \frac{\varepsilon}{2n} \\
= &\frac{\varepsilon}{2}
\end{aligned}$$

and, by Claim 3.6,

$$\begin{aligned}
&\sum_{(j, v_I^{(j-1)}) \notin S} \Pr\left[V_{j-1}(I) = v_I^{(j-1)}\right] \Pr\left[\text{FirstMiss}_j \mid V_{j-1}(I) = v_I^{(j-1)}\right] \Pr\left[\text{Fail}^{(j)} \mid \text{FirstMiss}_j \wedge V_{j-1}(I) = v_I^{(j-1)}\right] \\
\leq &\sum_{(j, v_I^{(j-1)}) \notin S} \Pr\left[V_{j-1}(I) = v_I^{(j-1)}\right] \Pr\left[\text{FirstMiss}_j \mid V_{j-1}(I) = v_I^{(j-1)}\right] \cdot \frac{\varepsilon}{2} \\
\leq &\sum_{(j, v_I^{(j-1)})} \Pr\left[V_{j-1}(I) = v_I^{(j-1)}\right] \Pr\left[\text{FirstMiss}_j \mid V_{j-1}(I) = v_I^{(j-1)}\right] \cdot \frac{\varepsilon}{2} \\
= &\frac{\varepsilon}{2}.
\end{aligned}$$

Thus, $\Pr[\text{Fail}] \leq \varepsilon/2 + \varepsilon/2 = \varepsilon$. □

3.2.5 Bounding the number of queries

Lemma 3.7. *Ivy makes $\tilde{O}(nm/\varepsilon)$ queries using n rounds of adaptivity.*

Proof of Lemma 3.7. Since the queries in each round of the for-loop in Algorithm 3 depend only on the results of the previous round, it is clear that Ivy uses at most n rounds of adaptivity.

Fix a view $v_I^{(j-1)}$ of Ivy at the start of round j. Let $Q_j^{(k)}$ be the value of Q_j at the end of the k^{th} iteration of the while-loop in round j. We bound the total number of queries made in each round by showing that

$$\Pr\left[Q_j^{(k)} \cap Q(A) = \emptyset \wedge \text{NoMiss}_{j-1} \mid V_{j-1}(I) = v_I^{(j-1)}\right] < (1-\gamma)^{k-1}$$

(Claim 3.8, below). This implies that for $k > (1/\gamma) \cdot \ln(n/\varepsilon) = O((m/\varepsilon) \cdot \log(n/\varepsilon))$, we have

$$\Pr\left[Q_j^{(k)} \cap Q(A) = \emptyset \wedge \text{NoMiss}_{j-1} \mid V_{j-1}(I) = v_I^{(j-1)}\right] < (1-\gamma)^k < e^{-\gamma k} < \frac{\varepsilon}{n},$$

hence the condition of step 6 will be met and Ivy will exit the while-loop. Since there are n rounds and each iteration of the while loop adds a single query, the total number of queries is bounded by $O((nm/\varepsilon) \cdot \log(n/\varepsilon)) \subseteq \tilde{O}(nm/\varepsilon)$. □

Claim 3.8. *For every round $j \in \{1, \ldots, n\}$, view $v_I^{(j-1)}$ and iteration k of the while-loop,*

$$\Pr\left[Q_j^{(k)} \cap Q(A) = \emptyset \wedge \textit{NoMiss}_{j-1} \mid V_{j-1}(I) = v_I^{(j-1)}\right] \leq (1-\gamma)^{k-1}.$$

Proof. We prove the claim by induction on k. For $k = 1$ the claim is trivial. Assume the hypothesis is true up to iteration k. If the while-loop continues beyond the k^{th} iteration, it must be that at the start of iteration $k+1$ there exists $q' \notin Q_j^{(k)} \cup Q_{j-1}(I)$ such that

$$\Pr\left[q' \in Q(A) \mid Q_j^{(k)} \cap Q(A) = \emptyset \wedge \text{NoMiss}_{j-1} \wedge V_{j-1}(I) = v_I^{(j-1)}\right] \geq \gamma.$$

Noting that $Q_j^{(k+1)} = Q_j^{(k)} \cup \{q'\}$, we can write

$$\begin{aligned}
&\Pr\left[Q_j^{(k+1)} \cap Q(A) = \emptyset \wedge \text{NoMiss}_{j-1} \mid V_{j-1}(I) = v_I^{(j-1)}\right] \\
&= \Pr\left[q' \notin Q(A) \wedge Q_j^{(k)} \cap Q(A) = \emptyset \wedge \text{NoMiss}_{j-1} \mid V_{j-1}(I) = v_I^{(j-1)}\right] \\
&= \Pr\left[q' \notin Q(A) \mid Q_j^{(k)} \cap Q(A) = \emptyset \wedge \text{NoMiss}_{j-1} \wedge V_{j-1}(I) = v_I^{(j-1)}\right] \\
&\qquad \cdot \Pr\left[Q_j^{(k)} \cap Q(A) = \emptyset \wedge \text{NoMiss}_{j-1} \mid V_{j-1}(I) = v_I^{(j-1)}\right] \\
&\leq (1-\gamma) \cdot (1-\gamma)^{k-1} = (1-\gamma)^k
\end{aligned}$$

where the final inequality is due to the induction hypothesis. □

3.3 Proof of Theorem 1.1

Theorem 1.1. *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 h 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 h).*

Proof. We now show how the theorem follows from Lemmas 3.7 and 3.3. Let f be an oracle algorithm that makes at most n queries to H and g a solver that makes at most $m > n$ queries to H, and satisfies

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

We claim the following process will successfully solve a puzzle with probability at least $1 - \nu - \varepsilon$ and make only the queries required by an execution of Alg. 3 (which by Lemma 3.7 satisfies the requirements of the theorem):

1. Run Ivy (Alg. 3) on input M with parameter ε.
2. Generate a new oracle, H^*, consistent with H on $Q(I)$ (i.e., $H^*(q) = H(q)$ for all $q \in Q(I)$ and $H^*(q)$ is chosen independently at random for all other q).
3. Run Bob with the new oracle: $x^* \leftarrow g^{H^*}(r_B, M)$ and output x^*.

It is easy to see that the only oracle queries are made in the execution of Ivy. To see why the success probability is $1 - \nu - \varepsilon$, we will show a sequence of three processes, beginning with an honest solving process (that succeeds with probability $1 - \nu$) and ending with our new solver, and prove that at most an additional ε error is incurred when moving between them.

We describe the honest solving process as an OQ process, F, consisting of the a combination of Alice, Ivy and Bob. An execution of F consists of:

1. Run Alice: $(M, \mathcal{V}) \leftarrow f^H(r_A)$
2. Run Ivy (Alg. 3) with parameter ε.
3. Run Bob: $x \leftarrow g^H(r_B, M)$

The initial state of F, $V_0(F)$, consists of the independent random coins of Alice, Ivy and Bob. Think of the random oracle as using lazy evaluation: $H(q)$ is randomly chosen the first time the oracle is queried on index q. By our assumption, $\Pr[\mathcal{V}(x) = 1] \geq 1 - \nu$ (where the probability is over the random coins of the parties and the random oracle).

Now, consider a modified OQ process F' that works as follows:

1. Run Alice: $(M, \mathcal{V}) \leftarrow f^H(r_A)$
2. Run Ivy (Alg. 3) with parameter ε.
3. Generate a new oracle, H', consistent with H on $Q(I) \cup Q(A)$ (i.e., $H'(q) = H(q)$ for all $q \in Q(I) \cup Q(A)$ and $H'(q)$ is chosen independently at random for all other q).
4. Run Bob with the new oracle: $x' \leftarrow g^{H'}(r_B, M)$.

It is easy to see that $(\mathcal{V}, x')$ are distributed identically to $(\mathcal{V}, x)$, hence $\Pr[\mathcal{V}(x') = 1] \geq 1 - \nu$. Finally, we consider the OQ process F^*, corresponding to our solver:

1. Run Alice: $(M, \mathcal{V}) \leftarrow f^H(r_A)$
2. Run Ivy (Alg. 3) with parameter ε.
3. Generate a new oracle, H^*, consistent with H on $Q(I)$.
4. Run Bob with the new oracle: $x^* \leftarrow g^{H^*}(r_B, M)$.

Note that the only difference between F^* and F' is that H^* may differ from H' on indices in $Q(A) \setminus Q(I)$. If Bob does not query indices in $Q(A) \setminus Q(I)$, the two processes behave identically. However, saying that Bob did not query indices in $Q(A) \setminus Q(I)$ is equivalent to saying that Ivy did not miss any intersection queries, which happens with probability at most ε. Thus, the pairs $(\mathcal{V}, x')$ and $(\mathcal{V}, x^*)$ have statistical distance at most ε, so

$$\Pr_{F^*}[\mathcal{V}(x^*) = 1] \geq \Pr_{F'}[\mathcal{V}(x') = 1] - \varepsilon \geq 1 - \nu - \varepsilon \,.$$

□

4 A Time-Lock Puzzle with a Linear Difficulty Gap

In this section we give the construction and proof for Theorem 1.4:

Theorem 1.4. *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,* $\Pr\left[(M, \mathcal{V}_M) \leftarrow f^H(k, r); x \leftarrow h^H(k, r_I, M); \mathcal{V}_M(x) = 1\right] = neg(k)$ *(where neg is some negligible function in k).*

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 4:

Algorithm 4 Honest solver g on input $M = (M_0, \ldots, M_n)$ and oracle H

```
x_0 ← M_0  // x_0 is not "encrypted".
for i ∈ {1, ..., n} do
    x_i ← H(x_{i-1}) ⊕ M_i  // "decrypting" x_i requires an oracle query on index x_{i-1}.
Output (x_0, ..., x_n)
```

Proof Sketch for Theorem 1.4. 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. 4) and the fact that it uses n serial queries is also easy to see.

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. □

4.1 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:

Claim 4.2. *The function $f^{H^{(d)}}(x_0, \ldots, x'_n)$ satisfies the requirements of Theorem 1.4 with $n = dn'$.*

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 1.4, we get the required parameters. □

5 Public Time-Lock Puzzle Constructions

Our negative result only applies to time-lock puzzles for which the solution verifier does not have access to the random oracle. This rules out any puzzles for which the puzzle generator knows the solution "ahead of time".

In this section we construct time-lock puzzles whose verifier makes essential use of the random oracle. Our constructions are "public puzzles", in the sense that the puzzle description is simply a random string and the verification does not require any secret information.

We describe both our constructions using an *interactive* verifier in a hybrid model in which an ideal commitment functionality exists. As a final step we will use the random oracle to implement the commitment functionality and the Fiat-Shamir heuristic to make the protocol non-interactive.

The high-level idea is to force the solver to commit to a directed "hash-graph" (in which each node is labeled with the hash of its children, and the leaves with the puzzle string). The verifier will then challenge the solver by choosing a subset of nodes at random. For each of the selected nodes, the solver will open the commitment to the labels of the node and its children. The verifier will accept if all the commitments are opened correctly and the label of each selected node is consistent with the labels of its children.

More formally, let $G = (V, E)$ be a directed, acyclic graph and denote $n \stackrel{def}{=} |V|$. Denote $\nu(v)$ the list of children of v in G. We say v is in layer ℓ of the graph if the longest path from v to a leaf is of length ℓ. We denote $\mathrm{diam}(G)$ the diameter of the graph (the length of the longest path in the graph).

Definition 5.1 (Good Node)**.** Let G' be an arbitrary labeling of graph nodes. We will say a node label v is *good* in G' if is consistent with $H(M, \nu(v)_1, \ldots, \nu(v)_{d^+(v)})$ (where $\nu(v)_1, \ldots, \nu(v)_{d^+(v)}$ denote the labels of its children in G').

Algorithm 5 describes the honest public solver and Algorithm 6 the corresponding verifier.

Algorithm 5 Honest "public" solver g on input M and oracle H

label every leaf node of G with the string 0^k.
for $\ell \in \{1, \ldots, \mathrm{diam}(G)\}$ **do** // Compute the hash-graph corresponding to G, by "layers"
    for every $v \in V$ such that v is in layer ℓ of G **do**
        label v with $H(M, \nu(v)_1, \ldots, \nu(v)_{d^+(v)})$
Send a commitment to the labeling of G to the verifier. // The commitment binds each label to a specific node.
Receive a list of challenge nodes $v_1, \ldots, v_k$ from the verifier.
for $i \in \{1, \ldots, k\}$ **do** // Repeat for each challenge node
    Open the commitments to the labels of v_i and $\nu(v_i)$.

Algorithm 6 "public" verifier on puzzle M and oracle H

Wait to receive commitment to labeling of G from solver.
Randomly choose k nodes and send $v_1, \ldots, v_k$ to the solver.
for $i \in \{1, \ldots, k\}$ **do** // Repeat for each challenge node
    Verify the commitment opening for the labels of v_i and $\nu(v_i)$
    Verify that the v is good in G (by making the query $H(M, \nu(v)_1, \ldots, \nu(v)_{d^+(v)})$ and comparing the the label of v).

Note that total query complexity of the honest solver is n and the parallel complexity is the diameter of the graph G. The query complexity of the verifier is $c \cdot k$, where c is the number of oracle queries required to verify a commitment.

To ensure that a dishonest solver cannot solve the puzzle with parallel complexity significantly less, we will rely on the following claim:

Claim 5.2. *Every adversary that outputs a labeled graph G' that contains a path of length d consisting entirely of good nodes must use at least d parallel rounds of queries.*

We will use graphs with the following property:

Definition 5.3 (diameter robustness)**.** We say G is (α, β)*-diameter robust* if, for every $S \subseteq V$ such that $|S| \geq \alpha|V|$, the diameter of the subgraph of G induced by S is least $\lfloor \beta|V| \rfloor$.

For every such graph, we can prove:

Lemma 5.4. *If G is a (α, β)-diameter robust with n vertices, any adversary that makes less than βn parallel rounds of queries to H will be caught by the verifier of Alg. 6 with probability $1 - \alpha^k$*

Proof Sketch. Let G' be the graph labeling committed to by the adversary. If the adversary is challenged on a bad node, the verifier will reject. Hence, the number of good nodes must be at least αn (otherwise the adversary will be caught with probability at least $1 - \alpha^k$). However, in this case there must be a path in G' of length at least $\lfloor \beta n \rfloor$ consisting entirely of good nodes (due to the diameter robustness property). By Claim 5.2, this completes the proof. □

5.1 Simple Construction

In our simplest construction, the graph G is the complete DAG (every node is connected to all previous nodes). We also use a trivial commitment scheme for this construction: the solver sends the entire labeling of G. The verifier needs only k queries to H (since the commitment verification doesn't require an oracle query), while the honest solver needs n serial queries.

Note that the complete DAG is (α, α)-diameter robust for all $\alpha \in (0, 1)$ (since every subgraph of m nodes contains a path of length m. Thus, by lemma 5.4 it follows that any adversary that makes less than $\frac{1}{2}n$ parallel rounds of queries will be caught with probability $1 - 2^{-k}$.

5.2 Lowering the Communication Complexity

The major drawback of our simple construction is the communication complexity, which is $\Omega(n)$ since the labeling of the entire graph is sent. The communication complexity of the protocol depends on the out-degree of the vertices in G (since for each challenge vertex v the solver sends the labels of v and all its children) and on the communication complexity of the commitments.

5.2.1 Ingredients

To improve the communication complexity, we will use a more compact commitment and also require the nodes of G to have bounded degree. Our construction will be based on expansion properties of the graph G:

Definition 5.5 (disperser graph)**.** A bipartite graph $G = (V, W, E)$, $|V| = |W| = N$ is a (K, ϵ)-disperser graph if every subset $S \subseteq W$ of cardinality K has at least $(1 - \epsilon)N$ distinct neighbors in V.

We will use $(\gamma N, \gamma)$-disperser graphs for $gamma = O(1/\log N)$. A random bipartite graph with degree $O(\log N \log\log N)$ is such a disperser with high probability.

5.2.2 Construction

We propose the following recursive construction for G. Let G_1 be the two-vertex graph. Informally, G_{i+1} consists of two identical copies of G_i connected with the edges of a bipartite disperser graph. Formally, $G_{i+1} = (V_{i+1}, W_{i+1}, E_{i+1})$, where $V_{i+1} = \left\{1, \ldots, 2^i\right\}$, $W_{i+1} = \left\{2^i + 1, \ldots, 2^{i+1}\right\}$ and

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

such that the graph $G' = (V, W, E'_{i+1})$ is a $(\gamma 2^i, \gamma)$-disperser.

Lemma 5.6. *For every i and $\alpha \in (0, 1)$, the graph G_i is $(\alpha, \alpha - i\gamma)$-diameter robust.*

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

Fix an arbitrary $\alpha \in (0, 1)$ (i.e., the number of good nodes in G_{i+1} is at least $\alpha 2^{i+1}$). Let $\delta 2^i$ be the number of good nodes in W_{i+1}. Since the total number of good nodes is $\alpha 2^{i+1}$, there must be at least $(2\alpha - \delta)2^i$ good nodes in V_{i+1}.

By the induction hypothesis, there exists a path $P_W \subseteq W_{i+1}$ such that $|P_W| \geq (\delta - i\gamma)2^i$ and all the nodes in P_W are good. In the same way, there exists a path $P_V \subseteq V_{i+1}$ such that $|P_V| \geq (2\alpha - \delta - i\gamma)2^i$ and all the nodes in P_V are good.

We must show that there exists a path $P \subset W_{i+1} \cup V_{i+1}$ such that $|P| \geq (\alpha - (i+1)\gamma)2^{i+1}$. If $\delta < 2\gamma$, then we can set $P = P_W$, since in this case $|P_W| > (2\alpha - (i+2)\gamma)2^i \geq (\alpha - (i+1)\gamma)2^{i+1}$. If $\delta > 2\alpha - 2\gamma$, then in the same way we can set $P = P_V$. Otherwise, consider the sets $S \subset P_W \subseteq W_{i+1}$ consisting of the last $\gamma 2^i$ nodes on P_W and $T \subset P_V \subseteq V_{i+1}$ consisting of the first $\gamma 2^i$ nodes on P_V. Since G_{i+1} contains a $(\gamma 2^i, \gamma)$-disperser graph between the nodes of W_{i+1} and V_{i+1}, and given that $|S| = \gamma 2^i$, the neighbor set of S is of size at least $(1-\gamma)2^i$, hence must have an intersection with T; that is there exist vertices $w \in S$ and $v \in T$ such that $(w, v) \in E_{i+1}$. We define our new path P by "stitching together" the paths P_W and P_V with the edge (w, v): i.e., P is defined to be the nodes in P_W up to node w, concatenated with the nodes of P_V from node v onwards. Since w and v are in the last (resp. first) $\gamma 2^i$ nodes of P_W (resp. P_V) the length of P is at least

$$|P| \geq |P_W| - \gamma 2^i + |P_V| - \gamma 2^i \geq (\delta - i\gamma + 2\alpha - \delta - i\gamma)2^i - 2\gamma = (\alpha - (i+1)\gamma)2^{i+1} \ .$$

□

Corollary 5.7. *Let $G_{\log n}$ be constructed with parameter $\gamma = 1/(4 \log n)$. Then by Lemma 5.6, $G_{\log n}$ is a $(1/2, 1/4)$-diameter robust graph with n vertices.*

If we plug $G_{\log n}$ into Alg. 5, Lemma 5.4 guarantees that any adversary who solves a puzzle must make at least $\frac{1}{4}n$ serial rounds of oracle queries or be caught with probability $1 - 2^{-k}$.

6 Discussion and Open Questions

Time-Lock Puzzle Variations. In our definition of a time-lock puzzle, the solution validator, $\mathcal{V}$, output by the puzzle generator does not have access to the random oracle. This does not matter for most constructive uses of time-lock puzzles in the literature: the majority require the puzzle generator to know the solution to the puzzle when it is generated (in which case the solution validator is the trivial one that compares its input to a hard-wired constant). The significant exceptions are proofs-of-work and timed signatures [9] (where the time-lock puzzle's solutions don't matter, only whether the verifier accepts them).

A natural question is whether our negative results also hold for the more general definition of time-lock puzzles, in which $\mathcal{V}$ is allowed to query the random oracle. The answer in this case is no: if $\mathcal{V}$ is allowed to query the oracle, there exists a simple counter-example for our negative result: the puzzle generator outputs a random string M (without making any oracle queries), and the corresponding $\mathcal{V}$ outputs 1 on input x iff $x = H(H(\cdots H(M)\cdots))$ (i.e., x is the result of n iterated queries to the hash function). It is easy to see that any solver must make at least one query to the oracle in order to produce a valid solution (in fact, at least n adaptive rounds are required). This contradicts the statement of Theorems 1.1 and 1.2, who guarantee a solver that does not make any queries at all (since the puzzle generator did not make any).

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 (as noted by Dagdelen et al. [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, indepedently, 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).

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*, pages 680–688. IEEE Computer Society, 2007.

[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 S. Halevi, editor, *CRYPTO*, volume 5677 of *Lecture Notes in Computer Science*, pages 374–390. Springer, 2009.

[5] E. Biham, Y. J. Goren, and Y. Ishai. Basing weak public-key cryptography on strong one-way functions. In R. Canetti, editor, *TCC*, volume 4948 of *Lecture Notes in Computer Science*, pages 55–72. Springer, 2008.

[6] D. Boneh and X. Boyen. Efficient selective-id secure identity-based encryption without random oracles. In C. Cachin and J. Camenisch, editors, *EUROCRYPT*, volume 3027 of *Lecture Notes in Computer Science*, pages 223–238. Springer, 2004.

[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.

[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 M. Bellare, editor, *CRYPTO*, volume 1880 of *Lecture Notes in Computer Science*, pages 236–254. Springer, 2000.

[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 E. Biham, editor, *EUROCRYPT*, volume 2656 of *Lecture Notes in Computer Science*, pages 255–271. Springer, 2003.

[13] J. Cathalo, B. Libert, and J.-J. Quisquater. Efficient and non-interactive timed-release encryption. In S. Qing, W. Mao, J. Lopez, and G. Wang, editors, *ICICS*, volume 3783 of *Lecture Notes in Computer Science*, pages 291–303. Springer, 2005.

[14] G. D. Crescenzo, R. Ostrovsky, and S. Rajagopalan. Conditional oblivious transfer and timed-release encryption. In J. Stern, editor, *EUROCRYPT*, volume 1592 of *Lecture Notes in Computer Science*, pages 74–89. Springer, 1999.

[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 D. Boneh, editor, *CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 426–444. Springer, 2003.

[17] 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.

[18] 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.

[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*, pages 212–219. ACM, 1996.

[21] R. Impagliazzo and S. Rudich. Limits on the provable consequences of one-way permutations. In *STOC*, pages 44–61. ACM, 1989.

[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 T. M. A. Lomas, editor, *Security Protocols Workshop*, volume 1189 of *Lecture Notes in Computer Science*, pages 69–87. Springer, 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.

Nomenclature

$\odot$ Vector concatenation operator: $(x_1, \ldots, x_n) \odot y \stackrel{def}{=} (x_1, \ldots, x_n, y)$, page 9

δ $1/(m+1)$, page 14

ε The error parameter: Ivy can fail to find all intersection queries with probability $O(\varepsilon)$, page 11

γ The "heaviness" parameter Ivy uses in Algorithm 3. We set $\gamma = \varepsilon/(2em)$, page 11

ν The failure probability of the inverter, g (a.k.a Bob): $\Pr\left[(M, \mathcal{V}) \leftarrow f^H(r); x \leftarrow g^H(M); \mathcal{V}(x) \neq 1\right] \leq \nu$, page 3

$q_i(B)$ The i^{th} query asked by Bob, page 11

$Q_F(i)$ The query list of the Oracle-Query (OQ) process F: $Q_i(F) \stackrel{def}{=} (F_1(V_0), \ldots, F_i(V_{i-1}))$, page 9

$Q_j(I)$ The vector of queries asked by Ivy in the first j rounds, page 11

$\mathcal{V}$ The puzzle solution validator; this may be computed by Alice together with the puzzle itself $((M, \mathcal{V}) = f^H(r_A))$; A solution x is considered valid if $\mathcal{V}(x) = 1$., page 3

$V_i(F)$ The view of an Oracle-Query (OQ) process F: $V_i = V_{i-1} \odot H(F_i(V_{i-1}))$ and $V_0(F)$ consists of the input and random coins for the process, page 9

$V_j(I)$ Ivy's view after the j^{th} round of queries, page 11

f The puzzle-generator function. f makes at most n queries to the random oracle, page 3

g The honest solver for the puzzle-generator function. g makes at most m queries to the random oracle, page 3

h The puzzle-solving adversary guaranteed by Theorem 1.1. h makes at most $\tilde{O}(nm/\varepsilon)$ queries to the random oracle and uses only n levels of adaptivity., page 3

J Javier, the adversary of Theorem 1.2, page 4

M The puzzle message sent by Alice $((M, \mathcal{V}) = f^H(r_A))$, page 3

m The number of queries to the random oracle made by the inverter, g, page 3

n The number of queries to the random oracle made by f, page 3

r_A Alice's random coins (the coins used in puzzle generation), page 3

r_B Bob's random coins (the coins used by the honest puzzle solver), page 3

r_J The random coins used by the puzzle-solving adversary, h, page 3

W The event $\text{FirstMiss}_j \wedge V_{j-1}(I) = v_I^{(j-1)} \wedge \text{Good}_i^{(j)} \wedge V_{i-1}(B) = v_B^{(i-1)}]$, page 13

Fail Fail^n, page 11

$\text{Fail}^{(j)}$ The event that Ivy has not hit all intersection queries in her first j rounds ($Q(B) \cap Q(A) \not\subseteq Q_j(I)$), page 11

$\text{Fail}_i^{(j)}$ The event that that Bob's i^{th} query is the first intersection query that was missed by Ivy in the first j rounds of Algorithm 3 (i.e., $\text{Good}_{i-1}^{(j)}$ holds and $q_i(B) \in Q(A) \setminus Q_j(I)$)., page 12

FirstMiss_j the event that round j is the first at which Ivy misses all of Alice's queries ($\text{FirstMiss}_j \stackrel{def}{=} \text{Miss}_j \wedge \text{NoMiss}_{j-1}$), page 11

$\text{Good}_i^{(j)}$ The event that the first i queries made by Bob are either not intersection queries or were queried by Ivy in her first j rounds (i.e., $\{q_1(B), \ldots, q_i(B)\} \cap Q(A) \subseteq Q_j(I)$)., page 12

Miss_j The event that Ivy "missed" all of Alice's queries in round j (we say Ivy missed if $Q_j(I) \cap Q(A) = Q_{j-1}(I) \cap Q(A)$; equivalently: $\mathcal{Q}_j \cap Q(A) = \emptyset$)., page 11

NoMiss_j the event that Ivy hits at least one more of Alice's queries in each of the first j rounds ($\text{NoMiss}_j \stackrel{def}{=} \bigwedge_{k=1}^{j} \neg\text{Miss}_k$), page 11

$\mathcal{Q}_j$ The set of queries that Ivy makes to the random oracle in the j^{th} round of Algorithm 3, page 11