Time-Lock Puzzles from Randomized Encodings

Time-Lock Puzzles from Randomized Encodings

Nir Bitansky* Shafi Goldwasser† Abhishek Jain‡ Omer Paneth§
Vinod Vaikuntanathan¶ Brent Waters‖

August 10, 2015

Abstract

Time-lock puzzles are a mechanism for sending messages "to the future". A sender can quickly generate a puzzle with a solution s that remains hidden until a moderately large amount of time t has elapsed. The solution s should be hidden from any adversary that runs in time significantly less than t, including resourceful parallel adversaries with polynomially many processors.

While the notion of time-lock puzzles has been around for 22 years, there has only been a *single* candidate proposed. Fifteen years ago, Rivest, Shamir and Wagner suggested a beautiful candidate time-lock puzzle based on the assumption that exponentiation modulo an RSA integer is an "inherently sequential" computation.

We show that various flavors of *randomized encodings* give rise to time-lock puzzles of varying strengths, whose security can be shown assuming *the existence* of non-parallelizing languages, which are languages that require circuits of depth at least t to decide, in the worst-case. The existence of such languages is necessary for the existence of time-lock puzzles. We instantiate the construction with different randomized encodings from the literature, where increasingly better efficiency is obtained based on increasingly stronger cryptographic assumptions, ranging from one-way functions to indistinguishability obfuscation. We also observe that time-lock puzzles imply one-way functions, and thus the reliance on some cryptographic assumption is necessary.

Finally, generalizing the above, we construct other types of puzzles such as *proofs of work* from randomized encodings and a suitable worst-case hardness assumption (that is necessary for such puzzles to exist).

*MIT. Email: nirbitan@csail.mit.edu.

†MIT and Weizmann Institute. Email: shafi@theory.csail.mit.edu.

‡Johns Hopkins University. Email: abhishek@cs.jhu.edu.

§Boston University. Email: omer@bu.edu. Supported by the Simons award for graduate students in theoretical computer science and an NSF Algorithmic foundations grant 1218461.

¶MIT. Email: vinodv@mit.edu.

‖UT Austin. Email: bwaters@cs.utexas.edu.

Contents

1 Introduction

A central theme in cryptography is the design of schemes that are secure against adversaries whose running time is bounded by *some* polynomial. Nevertheless, in some scenarios a more precise quantification of the adversary's computational resources may be called for. A useful notion in such scenarios is that of *cryptographic puzzles* that require some precise amount of time or space to solve. Such puzzles are utilized in a wide range of applications including digital-currency, combating junk mail, and timed-release encryption [DN92, RSW00, JJ99, Nak00].

As a leading example, consider the notion of *time-lock puzzles* introduced by Rivest, Shamir, and Wagner [RSW00], following May's work on timed-release cryptography [May93]. Informally, this is a mechanism for sending messages to the future. The sender generates a puzzle with a solution s that remains hidden until time t has elapsed allowing the puzzle to be solved. Concretely, s should be hidden from adversaries that run in time significantly less than t, including parallel adversaries with polynomially many processors, or more broadly, polynomial size circuits of depth much less than t.

While the notion of time-lock puzzles has been around for 22 years, there has essentially only been a *single* candidate proposed. Fifteen years ago Rivest, Shamir, and Wagner [RSW00] suggested candidate time-lock puzzles based on the assumption that exponentiation modulo an RSA integer is an "inherently sequential" computation. Since, besides variants of the same construction [BN00, GMPY11], there have been no other candidates meeting the standard notion of time-lock puzzles.[1] This is in contrast to other cryptographic primitives such as one-way functions for which several candidates have been discovered and studied over the years. Moreover, the hardness of several such candidates can be based on the worst-case hardness of lattice problems that are currently resilient against quantum algorithms (unlike the existing candidate for time-lock puzzles, which necessitates the hardness of factoring).

We identify two main challenges in designing time-lock puzzles based on natural cryptographic assumptions.

- *Complexity-theoretic bounds on parallelism.* While *typical* cryptographic assumptions address adversaries of *any* polynomial size,[2] very crudely, the time-lock puzzle notion requires differentiating between polynomial size adversaries that are of different polynomial depth (i.e., different parallel running time). For example, the existence of time-lock puzzles implies that $\mathrm{P} \not\subseteq \mathrm{NC}$, whereas cryptographic primitives such as one-way functions, fully-homomorphic encryption, or even indistinguishability obfuscation may exist even if $\mathrm{P} = \mathrm{NC}$.

- *Inherent sequentiality vs. fast generation.* While there are several candidates for problems that are "inherently sequential to solve" (even on the average-case), the notion of time-lock puzzles further requires that instances for such problems can be generated fast *together with their solution*. Finding candidate constructions satisfying both of these requirements has proved to be rather elusive.

We thus set out to study what is the *weakest complexity-theoretic condition that, combined with well studied cryptographic assumptions* suffice for the construction of time-lock puzzles.

[1] In nonstandard models, other constructions are known. For example, Mahmoody, Moran and Vadhan [MMV11] constructed *weak* time-lock puzzles in the *random oracle model* where the puzzle generator must spend roughly the same amount of computation to make the puzzle as the solver must use to solve it. The key requirement is that the generator should be able to spread its computation in parallel over several machines, yet a solver must work sequentially.

We also note that, when dropping the requirement regarding parallelism, additional solutions exist [BN00], including recent ones which harness the public bitcoin network for the solution of puzzles [LGR15, Jag15].

[2] We restrict our discussion to non-uniform adversaries.

1.1 This work

Time-Lock Puzzles From Non-Parallelizing Languages and Randomized Encodings. We put forth the notion of *t-non-parallelizing languages*: decidable in time t, but hard for circuits of depth significantly smaller than t. We focus our attention on languages that are non-parallelizing *in the worst case*, meaning that every shallow decider fails on *some* instance. The assumption that worst-case non-parallelizing languages exist can be seen as a natural generalization of the assumption that P $\not\subseteq$ NC, and has universal candidates. Indeed, any language $\mathcal{L} \in \text{Dtime}(t(\cdot))$ that is P-complete [GHR95] under linear reductions (e.g. *the bounded halting problem*) would be non-parallelizing, assuming that t-non-parallelizing languages exist.

It is easy to see that worst-case non-parallelizing languages are necessary for time-lock puzzles. Indeed, time-lock puzzles can be seen as samplable and *hard on average* non-parallelizing languages: they yield a pair of samplable distributions over yes-instances and no-instances that cannot be distinguished by circuits that are too shallow. In this work, we ask whether worst-case non-parallelizing languages are also sufficient, which is somewhat analogous to the fundamental problem of basing one-way functions on NP-hardness. While the latter question concerns the nature of general polynomial-time computation, the former concerns computations with some precise depth complexity.

Our main result shows how to construct time-lock puzzles starting from any efficient *randomized encoding* scheme, assuming the existence of a *worst-case* non-parallelizing language. Randomized encodings [IK00, AIK06] allow one to express a complex computation given by a function f and input x, by a *simpler-to-compute* representation $\widehat{f}(x)$ whose distribution encodes the output $f(x)$, but computationally hides any other information. Looking ahead, the efficiency of generating (or "locking") the new time-lock puzzles will be tightly related to the complexity parameters of the randomized encoding used.

Our construction possesses a salient universality feature: its security can be based on the existence of *any* family of non-parallelizing languages. This is unlike the candidate of Rivest, Shamir and Wagner [RSW00] who rely on the (average-case) non-parallelizability of a particular computation with respect to a particular distribution.

Instantiations of New Time Lock Puzzles. We instantiate the construction with different randomized encodings from the literature, where increasingly better efficiency is obtained based on increasingly stronger cryptographic assumptions, ranging from one-way functions to indistinguishability obfuscation (IO). We also observe that time-lock puzzles imply one-way functions, and thus the reliance on some cryptographic assumption is necessary.

At one end of the spectrum, the standard (and strongest) notion of time-lock puzzles requires that the time to generate a puzzle is essentially independent of the time t required to solve the puzzle. To obtain such puzzles, we rely on *succinct randomized encodings* where the complexity of encoding is essentially independent of the complexity of the encoded computation. Such succinct randomized encodings are known based on IO [BGL+15, CHJV15, KLW15].[3]

At the other end of the spectrum, we consider *weak time-lock puzzles* where only the parallel-time of generating a puzzle is independent of the time t required to solve the puzzle, whereas the overall (sequential) time may be proportional t. Such puzzles were constructed by Mahmoody, Moran, and Vadhan in the random oracle model [MMV11]. We construct weak time-lock puzzles based on the traditional notion of (non-succinct) randomized encodings (in another language, Yao's garbled circuits [Yao86]), which can be based on one-way functions, and where encoding is indeed highly parallelizable as required.

[3] We note that a strong form of succinct randomized encoding that are *output compressing* imply succinct functional encryption [AJ15], which in turn implies IO [AJ15, BV15, LPST15]. Such output compression is not required for our results, and as far as we know succinct randomized encodings may be weaker than IO.

We also consider an intermediate notion of *time-lock puzzles with pre-processing*. Here the puzzle generator performs a one-time expensive preprocessing phase and can subsequently generate an unbounded number of puzzles with low additional cost. Concretely, preprocessing takes total time t but can be parallelized as in weak time-lock puzzles, and then generating each puzzle is done in time independent of t as in standard time-lock puzzles. Such puzzles can be obtained from *reusable randomized encodings* (also known as *reusable garbled circuits*), where a function f is encoded in an expensive preprocessing phase, and subsequently any input x for the function can be encoded in time that is independent of the complexity of f. Reusable randomized encodings are, in turn, known based on sub-exponential LWE [GKP+13].

A caveat of the two latter constructions is that the size of the puzzle (a garbled circuit) is proportional to the time t required to solve the puzzle, thus making it meaningful only in a model where communication is cheaper than computation.[4]

Other puzzles. Finally, generalizing the above, we construct other types of puzzles such as *proofs of work* [DN92] where the measure of parallel-time is replaced by another complexity measure. Proofs of work are puzzles that require some precise computational effort to solve and are non-amortizable. Concretely, it is required that, for any t chosen by the generator, a single puzzle can be solved in time t, but any polynomial number of puzzles k cannot be solved by a circuit of size significantly smaller than $t \cdot k$.

Analogously to the time-lock puzzles construction, we prove the security of the proof-of-work thus constructed, based on randomized encodings and on a suitable worst-case hardness assumption that is necessary for such puzzles to exist. That is, the existence of *non-amortizing languages* (instead of non-parallelizing languages). A language $\mathcal{L}$ decidable in time t is worst-case non-amortizing if for every k, no circuit of size significantly smaller than $t \cdot k$ can decide the direct product language $\mathcal{L}^k$. The proof of security follows the same lines as above.

We note that even more generally, one can apply the above approach with different complexity measures, relying on an appropriate worst-case assumption.

1.2 The Main Idea: Construction and Proof

Let f be a function, x an input, and $\widehat{f}(x)$ its randomized encoding. By the definition of randomized encodings, the encodings of two computations with the same output are indistinguishable, even if the computations behave differently before producing this output. For instance, one computation might solve some hard problem, whereas the other one stalls and then outputs a hard-coded solution to a hard problem. Our construction follows this intuition: a time-lock puzzle with solution s solvable in time t, consists of a randomized encoding of a "dummy computation" that outputs s after t dummy steps.

In the security proof, we will show that any adversary $\mathcal{A}$ of depth significantly smaller than t that distinguishes puzzles with different solutions s_0 and s_1, can be turned into a circuit $\mathcal{D}$ deciding any given non-parallelizing languages $\mathcal{L}$ with roughly the same depth as $\mathcal{A}$. The decider $\mathcal{D}$, given an input x, first computes a randomized encoding of the program $M^{\mathcal{L}}_{s_0,s_1}$ that decides (in time t) whether $x \in \mathcal{L}$, and outputs s_0 or s_1 accordingly. By the guarantee of the randomized encoding, the encoding $\widehat{M}^{\mathcal{L}}_{s_0,s_1}$ is either indistinguishable from a time-lock puzzle with solution s_0 or from one with solution s_1 according to whether $x \in \mathcal{L}$. Since $\mathcal{A}$ can distinguish between the two, $\mathcal{D}$ can invoke $\mathcal{A}$ to successfully decide the language. The depth of the decider $\mathcal{D}$ is the same as the depth of $\mathcal{A}$ plus the depth required to compute the randomized encoding. To reach a contradiction we therefore rely on encodings that can be computed in depth that is significantly smaller than t.

[4] In the random oracle model, [MMV11] show how to trade between communication and the parallel puzzle generation time d, achieving a solution with communication t/d.

1.3 Alternative Approaches

A Refined Approach via Message-hiding Encodings. We observe that a different construction of time-lock puzzles can be obtained from a relaxation of randomized encodings called *message-hiding encodings* [IW14, KLW15], at the price of assuming stronger non-parallelizing languages.

In message-hiding encodings, a secret message m is encoded with respect to some public predicate P and input x. Decoding m is possible only if $P(x) = 1$, and otherwise the encoding computationally hides m. As in succinct randomized encodings, the complexity of encoding the message is essentially independent of the complexity of P.

The strengthening of non-parallelizing languages requires that, not only do shallow circuits fail to decide the language $\mathcal{L}$ in the worst-case, but they also fail on some samplable instance distribution $\mathcal{X}$ with probability approximately half. While this is already an average-case guarantee, it is still seemingly weaker than the average-case hardness that is equivalent to time-lock puzzles outlined before; there, both yes-instances and no-instances should be efficiently samplable.

In the new construction, the solution s is encoded as the message twice, once with respect to $(P_{\mathcal{L}}, x)$ and then with respect to $(1 - P_{\mathcal{L}}, x)$, where the predicate $P_{\mathcal{L}}$ tests membership in $\mathcal{L}$ and x is sampled from $\mathcal{X}$. See further details in Appendix A. We note that, unlike in our previous construction from randomized encodings, the construction from message-hiding encodings explicitly depends on the specific non-parallelizing language together with the corresponding hard distribution $\mathcal{X}$.

Currently, message-hiding encodings are only known based on the same assumption required for succinct randomized encodings, namely, indistinguishability obfuscation. However, the above construction may become appealing if message-hiding encodings can be shown from qualitatively weaker assumptions. We observe that this "gap" can always be bridged based on fully-homomorphic encryption. Concretely, fully-homomorphic encryption lets us transform any message-hiding encoding into a succinct randomized encoding, relying on similar ideas to those used by Goldwasser, Kalai, Popa, Vaikuntanathan and Zeldovich [GKP+13] to transform attribute-based encryption to functional-encryption. See further details in Appendix A.

The above suggests a refined view of our original approach separated into two steps. The first step translates worst-case hardness to average-case hardness (and also removes the need to explicitly know the language and hard distribution). The second step translates average-case hardness with oblivious instance sampling to average-case hardness where instances can be sampled with a solution. Whereas randomized encodings achieve both effects together, they can be separated: the second step can be achieved based on message-hiding encodings, and the first, by adding fully-homomorphic encryption. Indeed, fully-homomorphic encryption was previously used to yield similar worst-case/average-case connections for functions in deterministic time classes (e.g., in [CKV10]).

A Heuristic Approach Using Obfuscation. We briefly note that there exists a natural heuristic approach to designing time lock puzzles using obfuscation. The puzzle generator on input time t and solution s will first create a MAC key k and a signature σ_1 on the message $m = 1$. Next, it creates an obfuscation of the following program. The program will take as input a message, signature pair (m, σ) for $m \in [1, t]$. For $m < t$ it first verifies (using k) that σ is a signature on m, if this check passes the program outputs a signature σ' on message $m + 1$. Finally, if $m = t$ and the signature verifies, the program will output the solution s. The published puzzle is σ_1 together with the obfuscated program. To solve the puzzle, one simply starts by inputting the initial signature into the obfuscated program and receiving a new signature. This process is repeated t times until the solution s is received.

Heuristically, it might appear that this is a potential candidate for a time lock puzzle. Unfortunately, we do not currently see any way to analyze its security.

1.4 On the Necessity of One-way Functions

We observe that (standard) time-lock puzzles imply the existence of one-way functions. To convey the idea behind the implication, assume for starters that we are given time-lock puzzles where it is possible to generate, in fixed polynomial time, puzzles that are solvable only in some super-polynomial time, e.g. $t = \lambda^{\log \lambda}$. Then, we claim that the function that maps the random coins r of the generator to a puzzle with solution $s = 0$ solvable in time t is a one-way function. First, since the puzzle can be computed fast so can the function. Second, any polynomial inverter for the function can also distinguish in polynomial time $\lambda^{O(1)} \ll t = \lambda^{\log \lambda}$ a puzzle with solution $s = 0$ from one with solution $s = 1$.

In standard time-lock puzzles, however, while correctness is guaranteed for any value of t, security is only guaranteed for t that is polynomial in the security parameter. At high-level, we can bridge this gap by choosing the time t at random from an appropriate set resulting in a weak one-way function, and then use standard amplification [Gol01a]. See further details in Appendix B.

2 Preliminaries

The cryptographic definitions in the paper follow the convention of modeling security against non-uniform adversaries. An efficient adversary $\mathcal{A}$ is modeled as a sequence of circuits $\mathcal{A} = \{\mathcal{A}_\lambda\}_{\lambda \in \mathbb{N}}$, such that, for security parameter λ, the circuit $\mathcal{A}_\lambda$ is of polynomial size $\lambda^{O(1)}$ with $\lambda^{O(1)}$ input and output bits. Accordingly, the hardness assumptions made throughout the paper address non-uniform circuits. The results can be cast into the uniform setting, with some adjustments. As usual, all honest algorithms are modeled as uniform machines.

Parallelism. Throughout, we shall also be interested in the *parallel complexity* of certain tasks. To unify terminology, we shall also model parallel algorithms as (uniform or non-uniform) circuits. Here the parallel time of a given algorithm is determined by the depth $\mathsf{dep}(C)$ of the corresponding circuit C and the total running time (which in particular bounds the number of processors) is determined by the total size of the circuit $|C|$.

3 Time-Lock Puzzles

In this section, we define time-lock puzzles and construct them based on succinct randomized encodings. The security of the construction is assuming also the existence of a hard language that is non-parallelizing in the worst case.

3.1 Definitions

We start by defining the notion of puzzles. Then we define the security requirement for time-lock puzzles.

Puzzles. A puzzle is associated with a pair of parameters: a security parameter λ determining the cryptographic security of the puzzle, as well as a *difficulty parameter* t that determines how difficult it is to solve the puzzle.

Definition 3.1 (Puzzles)**.** *A puzzle is a pair of algorithms* $(\mathsf{Puzzle.Gen}, \mathsf{Puzzle.Sol})$ *satisfying the following requirements.*

- *Syntax:*
  - $Z \leftarrow \mathsf{Puzzle.Gen}(t, s)$ *is a probabilistic algorithm that takes as input a difficulty parameter* t *and a solution* $s \in \{0,1\}^\lambda$*, where* λ *is a security parameter, and outputs a puzzle* Z*.*
  - $s \leftarrow \mathsf{Puzzle.Sol}(Z)$ *is a deterministic algorithm that takes as input a puzzle* Z *and outputs a solution* s*.*
- *Completeness: For every security parameter* λ*, difficulty parameter* t*, solution* $s \in \{0,1\}^\lambda$ *and puzzle* Z *in the support of* $\mathsf{Puzzle.Gen}(t, s)$*,* $\mathsf{Puzzle.Sol}(Z)$ *outputs* s*.*
- *Efficiency:*
  - $Z \leftarrow \mathsf{Puzzle.Gen}(t, s)$ *can be computed in time* $\mathrm{poly}(\log t, \lambda)$*.*
  - $\mathsf{Puzzle.Sol}(Z)$ *can be computed in time* $t \cdot \mathrm{poly}(\lambda)$*.*

Time-Lock Puzzles In a time-lock puzzle, we require that the parallel time required to solve a puzzle is proportional to the time it takes to solve the puzzle honestly, up to some fixed polynomial loss.

Definition 3.2 (Time-Lock Puzzles)**.** *A puzzle* $(\mathsf{Puzzle.Gen}, \mathsf{Puzzle.Sol})$ *is a time-lock puzzle with gap* $\varepsilon < 1$ *if there exists a polynomial* $\underline{t}(\cdot)$*, such that for every polynomial* $t(\cdot) \geq \underline{t}(\cdot)$ *and every polysize adversary* $\mathcal{A} = \{\mathcal{A}_\lambda\}_{\lambda \in \mathbb{N}}$ *of depth* $\mathsf{dep}(\mathcal{A}_\lambda) \leq t^\varepsilon(\lambda)$*, there exists a negligible function* μ*, such that for every* $\lambda \in \mathbb{N}$*, and every pair of solutions* $s_0, s_1 \in \{0,1\}^\lambda$*:*

$$\Pr\left[b \leftarrow \mathcal{A}_\lambda(Z) : \begin{array}{l} b \leftarrow \{0,1\} \ , \\ Z \leftarrow \mathsf{Puzzle.Gen}(t(\lambda), s_b) \end{array}\right] \leq \frac{1}{2} + \mu(\lambda) \ .$$

3.2 Succinct Randomized Encodings

The main tool used in the construction is a *succinct randomized encoding* scheme. A randomized encoding [IK00] allows to express a complex computation given by a function f and input x, by a *simpler-to-compute* representation $\widehat{f}(x)$ that encodes the output $f(x)$, but computationally hides any other information. In succinct randomized encoding the function f is given by a Turing machine and M and the (sequential) time required to compute $\widehat{M}(x)$ is independent of the complexity of f.

Definition 3.3 (Succinct Randomized Encoding)**.** *A succinct randomized encoding scheme* RE *consists of two algorithms* $(\mathsf{RE.Encode}, \mathsf{RE.Decode})$ *satisfying the following requirements.*

- *Syntax:*
  - $\widehat{M}(x) \leftarrow \mathsf{RE.Encode}(M, x, t, 1^\lambda)$ *is a probabilistic algorithm that takes as input a machine* M*, input* x*, time bound* t*, and a security parameter* 1^λ*. The algorithm outputs a randomized encoding* $\widehat{M}(x)$*.*
  - $y \leftarrow \mathsf{RE.Decode}(\widehat{M}(x))$ *is a deterministic algorithm that takes as input a randomized encoding* $\widehat{M}(x)$ *and computes an output* $y \in \{0,1\}^\lambda$*.*
- *Functionality: for every input* x *and machine* M *such that, on input* x*,* M *halts in* t *steps and produces a* λ*-bit output, it holds that* $y = M(x)$ *with overwhelming probability over the coins of* RE.Encode*.*

- *Security: there exists a PPT simulator* Sim *satisfying: for any poly-size distinguisher* $\mathcal{D} = \{\mathcal{D}_\lambda\}_{\lambda\in\mathbb{N}}$ *and polynomials* $m(\cdot), n(\cdot), t(\cdot)$*, there exists a negligible* $\mu(\cdot)$*, such that for any* $\lambda \in \mathbb{N}$*, machine* $M \in \{0,1\}^{m(\lambda)}$*, input* $x \in \{0,1\}^{n(\lambda)}$*:*

$$\Big|\Pr[\mathcal{D}_\lambda(\widehat{M}(x)) = 1 : \widehat{M}(x) \leftarrow \mathsf{RE.Encode}(M, x, t(\lambda), 1^\lambda)] -$$
$$\Pr[\mathcal{D}_\lambda(\widehat{S}_y) = 1 : \widehat{S}_y \leftarrow \mathsf{Sim}(y, 1^{m(\lambda)}, 1^{n(\lambda)}, t(\lambda), 1^\lambda)]\Big| \leq \mu(\lambda) \ ,$$

  where y *is the output of* $M(x)$ *after* $t(\lambda)$ *steps.*

- *Efficiency: For any machine* M *that on input* x *produces a* λ*-bit output in* t *steps:*
  - $\mathsf{RE.Encode}(M, x, t, 1^\lambda)$ *can be computed in (sequential) time* $\mathrm{polylog}(t)\cdot\mathrm{poly}(|M|, |x|, \lambda)$.
  - $\mathsf{RE.Decode}(\widehat{M}(x))$ *can be computed in (sequential) time* $t \cdot \mathrm{poly}(|M|, |x|, \lambda)$.

Succinct randomized encoding were constructed in [BGL+15, CHJV15, KLW15] based on *indistinguishability obfuscation*. The works of [BGL+15, CHJV15] gave constructions met the above efficiency property with the exception that the encoding took time (and size) proportional to the maximum memory used by the computation. This restriction is not present in [KLW15].

Theorem 3.4 ([KLW15])**.** *Assuming one-way functions and indistinguishability obfuscation for all circuits there exist a succinct randomized encoding scheme.*

3.3 Construction

We describe the construction of time-lock puzzles from succinct randomized encodings.

Construction 3.5 (Time-Lock Puzzles). Let RE be a succinct randomized encoding scheme. For $s \in \{0,1\}^\lambda$ and $t \leq 2^\lambda$, let M_s^t be a machine that, on any input $x \in \{0,1\}^\lambda$, outputs the string s after t steps (here we assume that $t \geq \lambda + \omega(1)$). Further assume that M_s^t is described by 3λ bits (which is possible for large enough λ).

The time-lock puzzle is constructed as follows:

- $\mathsf{Puzzle.Gen}(t, s)$ samples $\widehat{M}_s^t(0^\lambda) \leftarrow \mathsf{RE.Encode}(M_s^t, 0^\lambda, t, 1^\lambda)$ and outputs $Z = \widehat{M}_s^t(0^\lambda)$.
- $\mathsf{Puzzle.Sol}(Z)$ outputs $\mathsf{RE.Decode}(Z)$.

3.4 Proof of Security

The security of Construction 3.5 relies on the existence of a language with a specific kind of *worst-case hardness* that we refer to as "non-parallelizing" language. A poly-time decidable language $\mathcal{L}$ is *non-parallelizing* if parallel algorithms cannot do significantly better (than sequential algorithms) in deciding it. That is, the circuit depth of any family of circuits deciding the language $\mathcal{L}$ is as large as the (sequential) time required to decide $\mathcal{L}$ up to some fixed polynomial loss.

Definition 3.6 (Non-Parallelizing Language)**.** *A language* $\mathcal{L} \in \mathit{Dtime}(t(\cdot))$ *is non-parallelizing with gap* $\varepsilon < 1$ *if for every family of non-uniform polysize circuits* $\mathcal{B} = \{\mathcal{B}_\lambda\}_{\lambda\in\mathbb{N}}$ *where* $\mathsf{dep}(\mathcal{B}_\lambda) \leq t^\varepsilon(\lambda)$ *and every large enough* λ*,* $\mathcal{B}_\lambda$ *fails to decide* $\mathcal{L}_\lambda = \mathcal{L} \cap \{0,1\}^\lambda$.

The security of Construction 3.5 is stated in the following theorem.

Theorem 3.7. *Let* $\varepsilon < 1$*. Assume that, for every polynomial bounded function* $t(\cdot)$*, there exists a non-parallelizing language* $\mathcal{L} \in \mathit{Dtime}(t(\cdot))$ *with gap* ε*. Then, for any* $\underline{\varepsilon} < \varepsilon$*, Construction 3.5 is a time-lock puzzle with gap* $\underline{\varepsilon}$*.*

3.4.1 Proof of Theorem 3.7

The completeness and efficiency properties of the puzzle follow directly from the completeness and efficiency properties of the succinct randomized encoding scheme. We proceed to argue that the puzzle is a secure time-lock puzzle. Let $q_{\mathsf{RE}}(\lambda)$ be the fixed polynomial given by the efficiency property of the succinct randomized encoding scheme, bounding the time required to compute a randomized encoding with machine size 3λ, input size λ, and any time bound $t \leq 2^{\lambda}$. Let $\underline{t}(\lambda) := (q_{\mathsf{RE}}(\lambda))^{1/\underline{\varepsilon}}$.

Assume towards contradiction that there exists a polysize adversary $\mathcal{A} = \{\mathcal{A}_\lambda\}_{\lambda\in\mathbb{N}}$, and a polynomially bounded function $t(\cdot) \geq \underline{t}(\cdot)$ such that $\mathsf{dep}(\mathcal{A}_\lambda) < t^{\underline{\varepsilon}}(\lambda)$ and for some polynomial $p(\cdot)$ and infinitely many $\lambda \in \mathbb{N}$ there exists a pair of solutions $s_0, s_1 \in \{0,1\}^\lambda$ such that:

$$\Pr\left[b \leftarrow \mathcal{A}_\lambda(Z) : \begin{array}{l} b \leftarrow \{0,1\} \ , \\ Z \leftarrow \mathsf{Puzzle.Gen}(t(\lambda), s_b) \end{array}\right] \geq \frac{1}{2} + \frac{1}{p(\lambda)} \ . \tag{1}$$

Let $\mathcal{L} \in \mathrm{Dtime}(t(\cdot))$ be a non-parallelizing language with gap ε, which exists by assumption. We construct a polysize circuit family $\mathcal{B} = \{\mathcal{B}_\lambda\}_{\lambda\in\mathbb{N}}$ of depth $\mathsf{dep}(\mathcal{B}_\lambda) \leq t^{\varepsilon}(\lambda)$ that decides $\mathcal{L}_\lambda = \mathcal{L} \cap \{0,1\}^\lambda$ for any λ as above, contradicting the fact that $\mathcal{L}$ is non-parallelizing.

We start by constructing a *probabilistic* polysize adversary $\mathcal{B}'$ such that that $\mathsf{dep}(\mathcal{B}'_\lambda) = o(t^{\varepsilon}(\lambda))$ and $\mathcal{B}'_\lambda$ decides $\mathcal{L}_\lambda$ *with some noticeable advantage*. Then, we conclude the proof using a standard parallel repetition argument.

Fix any λ as above with corresponding $s_0, s_1 \in \{0,1\}^\lambda$. Let $M^{\mathcal{L},t}_{s_0,s_1}$ be a machine that, on input $x \in \{0,1\}^\lambda$, outputs s_1 if $x \in \mathcal{L}$ and and s_0 if $x \notin \mathcal{L}$, after exactly $t(\lambda)$ steps. Such a machine indeed exists since $\mathcal{L} \in \mathrm{Dtime}(t(\cdot))$. Further assume that $M^{\mathcal{L},t}_{s_0,s_1}$ is described by 3λ bits (which is possible for large enough λ), and thus has the same description length as $M^t_{s_b}$.

Given input $x \in \{0,1\}^\lambda$, to decide if $x \in \mathcal{L}$, the randomized $\mathcal{B}'_\lambda$ acts as follows:

- Sample $Z := \widehat{M}^{\mathcal{L},t}_{s_0,s_1}(x) \leftarrow \mathsf{RE.Encode}(M^{\mathcal{L},t}_{s_0,s_1}, x, t(\lambda), 1^\lambda)$.
- Obtain $b \leftarrow \mathcal{A}_\lambda(Z)$ and output b.

First, note that $\mathcal{B}'$ is of polynomial size and its depth is given by:

$$\mathsf{dep}(\mathcal{B}'_\lambda) = q_{\mathsf{RE}}(\lambda) + \mathsf{dep}(\mathcal{A}_\lambda) = \underline{t}^{\underline{\varepsilon}}(\lambda) + t^{\underline{\varepsilon}}(\lambda) \leq 2t^{\underline{\varepsilon}}(\lambda) = o(t^{\varepsilon}(\lambda)) \ .$$

We next show that $\mathcal{B}'$ distinguishes instances $x \in \mathcal{L}$ from instances $x \notin \mathcal{L}$ with noticeable advantage. For any $x \in \{0,1\}^\lambda$, let $b \in \{0,1\}$ indicate whether $x \in \mathcal{L}_\lambda$ we have that $s_b = M^{\mathcal{L},t}_{s_0,s_1}(x) = M^t_{s_b}(0^\lambda)$. Therefore, by the security of the randomized encoding scheme there exists a PPT simulator Sim and a negligible function $\mu(\cdot)$ such that for any $x \in \{0,1\}^\lambda$:

$$\begin{aligned}
&\Pr[\mathcal{B}'_\lambda(x) = 1] = \\
&\Pr\left[\mathcal{A}'_\lambda(\widehat{M}^{\mathcal{L},t}_{s_0,s_1}(x)) = 1 : \widehat{M}^{\mathcal{L},t}_{s_0,s_1}(x) \leftarrow \mathsf{RE.Encode}(M^{\mathcal{L},t}_{s_0,s_1}, x, t(\lambda), 1^\lambda)\right] = \\
&\Pr\left[\mathcal{A}'_\lambda(\widehat{S}_{s_b}) = 1 : \widehat{S}_{s_b} \leftarrow \mathsf{Sim}\left(s_b, 1^{3\lambda}, 1^\lambda, t(\lambda), 1^\lambda\right)\right] \pm \mu(\lambda) \ ,
\end{aligned}$$

and:

$$\begin{aligned}
&\Pr\left[\mathcal{A}_\lambda(Z) = 1 : Z \leftarrow \mathsf{Puzzle.Gen}(t(\lambda), s_b)\right] = \\
&\Pr\left[\mathcal{A}'_\lambda(\widehat{M}^t_{s_b}(0^\lambda)) = 1 : \widehat{M}^t_{s_b}(0^\lambda) \leftarrow \mathsf{RE.Encode}(M^t_{s_b}, 0^\lambda, t(\lambda), 1^\lambda)\right] = \\
&\Pr\left[\mathcal{A}'_\lambda(\widehat{S}_{s_b}) = 1 : \widehat{S}_{s_b} \leftarrow \mathsf{Sim}\left(s_b, 1^{3\lambda}, 1^\lambda, t(\lambda), 1^\lambda\right)\right] \pm \mu(\lambda) \ .
\end{aligned}$$

It follows by our assumption towards contradiction (Equation 1) that for large enough λ and any $x \in \mathcal{L}_\lambda$, $\bar{x} \notin \mathcal{L}_\lambda$:

$$\left|\Pr[\mathcal{B}'_\lambda(x) = 1] - \Pr[\mathcal{B}'_\lambda(\bar{x}) = 1]\right| \geq \frac{2}{p(\lambda)} - 2\mu(\lambda) \geq \frac{1}{p(\lambda)} \ .$$

To obtain the required $\mathcal{B}$ that deterministically works for any $x \in \{0,1\}^\lambda$, we rely on the standard parallel repetition argument showing that BPP/*poly* $\subseteq$ P/*poly* [Gol01b, Theorem 1.3.7]. Note that following this argument, $|\mathcal{B}| = \text{poly}(\mathcal{B}') = \text{poly}(\lambda)$ and $\mathsf{dep}(\mathcal{B}) = \mathsf{dep}(\mathcal{B}) + \text{polylog}(\lambda) = o(t^\varepsilon(\lambda))$ contradicting the fact that $\mathcal{L}$ is non-parallelizing.

3.5 Weak Time-Lock Puzzles from One-Way Functions

In this section, we provide a construction of *weak* time-lock puzzles based on one-way functions. Weak puzzle are defined similarly to standard puzzles (Definition 3.1) except that we only require that the puzzle can be generated in fast *parallel* time even though the sequential time to generate a puzzle may be as much or even more than the time to solve it. The security requirement for weak time-lock puzzles is unchanged (see Definition 3.2).

Definition 3.8 (Weak Puzzles)**.** *A weak puzzle is defined by a pair of algorithms* $(\mathsf{Puzzle.Gen}, \mathsf{Puzzle.Sol})$ *satisfying the syntax and completeness requirements as per Definition 3.1, and the following weak efficiency requirement.*

Weak Efficiency:

- $\mathsf{Puzzle.Gen}(t, s)$ *can be computed by a uniform circuit of size* $\text{poly}(t, \lambda)$ *and depth* $\text{poly}(\log t, \lambda)$.
- $\mathsf{Puzzle.Sol}(Z)$ *can be computed in time* $t \cdot \text{poly}(\lambda)$.

Our construction of weak time-lock puzzles is essentially the same as Construction 3.5 of that of standard time-lock puzzles except that instead of using a succinct randomized encoding, we use a standard *non-succinct* randomized encoding that can be obtained from any one-way function.

Definition 3.9 (Randomized Encoding)**.** *A randomized encoding scheme* RE *consists of two algorithms* $(\mathsf{RE.Encode}, \mathsf{RE.Decode})$ *satisfying the syntax, functionality and security requirements as per Definition 3.3, and the following efficiency requirement.*

Efficiency: For any machine M *that on input* x *produces a* λ*-bit output in* t *steps:*

- $\widehat{M}(x) \leftarrow \mathsf{RE.Encode}(M, x, t, 1^\lambda)$ *can be computed by a uniform circuit of depth* $\text{polylog}(t) \cdot \text{poly}(|M|, |x|, \lambda)$ *and total size* $t \cdot \text{poly}(|M|, \lambda)$.
- $\mathsf{RE.Decode}(\widehat{M}(x))$ *can be computed in (sequential) time* $t \cdot \text{poly}(|M|, |x|, \lambda)$.

Theorem 3.10. *Let* $\varepsilon < 1$. *Assume that, for every polynomial bounded function* $t(\cdot)$, *there exists a non-parallelizing language* $\mathcal{L} \in \mathit{Dtime}(t(\cdot))$ *with gap* ε. *Then, for any* $\underline{\varepsilon} < \varepsilon$, *Construction 3.5 instantiated with a standard randomized encoding is a weak time-lock puzzle with gap* $\underline{\varepsilon}$.

The proof of Theorem 3.10 follows closely the proof of Theorem 3.7 in Section 3.4 with the following modifications. The weak efficiency of the puzzle follows from the efficiency property of randomized encodings since $\mathsf{RE.Encode}(M_{t,s}, 0^\lambda, t, 1^\lambda)$ runs in *parallel* time $\text{polylog}(t) \cdot \text{poly}(|M_{t,s}|, \lambda) = \text{polylog}(t) \cdot \text{poly}(\lambda)$. In security proof we now think of $q_{\mathsf{RE}}(\lambda)$ as bounding the depth of the circuit $\mathsf{RE.Encode}$ instead of its sequential running time.

4 Time-Lock Puzzles with Pre-processing

In this section, we consider time-lock puzzles with *pre-processing*. Generating such puzzles requires a one-time preprocessing phase that is as expensive as solving the puzzle; however, subsequent to the pre-processing phase, one can generate any polynomial number of puzzles inexpensively.

We provide a construction of time-lock puzzles with pre-processing based on *reusable* randomized encodings. While known constructions of succinct randomized encodings (used in Section 3) are based on indistinguishability obfuscation, reusable randomized encodings can be constructed based on sub-exponential LWE [GKP+13]. Unlike succinct randomized encodings, the time to compute a reusable randomized encoding $\widehat{M}$ of a Turing machine M does depend on the running time of M. However, the encoded machine $\widehat{M}$ can then be evaluated on many encoded inputs and therefore the cost of encoding M is amortized over many evaluations.

4.1 Definitions

We start with the definition of puzzles with pre-processing and then define security of time-lock puzzles with pre-processing.

Puzzles with pre-processing. We first define puzzles with pre-processing whose efficiency lies in between that of standard (succinct) puzzles and weak puzzles. Concretely, in puzzles with pre-processing, most of the puzzle is generated ahead of time in a preprocessing phase, independently of any specific desired solution s. Subsequently, a puzzle with any solution s can be generated as efficiently as in standard (succinct) puzzles.

Definition 4.1 (Puzzles with Pre-processing)**.** *A puzzle with pre-processing is defined by a triple of algorithms*
$(\mathsf{Puzzle.Preproc}, \mathsf{Puzzle.Gen}, \mathsf{Puzzle.Sol})$ *satisfying the following requirements.*

- *Syntax:*
  - $(P, K) \leftarrow \mathsf{Puzzle.Preproc}(1^t, 1^\lambda)$ *is a probabilistic algorithm that takes as input a difficulty parameter t and a security parameter λ, and outputs a state P and a short $K \in \{0,1\}^\lambda$.*
  - $Z \leftarrow \mathsf{Puzzle.Gen}(s, K)$ *is a probabilistic algorithm that takes as input a solution $s \in \{0,1\}^\lambda$ and secret key K and outputs a puzzle Z.*
  - $s \leftarrow \mathsf{Puzzle.Sol}(P, Z)$ *is a deterministic algorithm that takes as input a state P and puzzle Z and outputs a solution s.*
- *Completeness: For every security parameter λ, difficulty parameter t, solution $s \in \{0,1\}^\lambda$, state P, key K in the support of $\mathsf{Puzzle.Preproc}(1^t, 1^\lambda)$, and puzzle Z in the support of $\mathsf{Puzzle.Gen}(s, K)$, $\mathsf{Puzzle.Sol}(P, Z)$ outputs s.*
- *Efficiency:*
  - $(P, K) \leftarrow \mathsf{Puzzle.Preproc}(1^t, 1^\lambda)$ *can be computed by a uniform circuit of depth* $\mathrm{poly}(\log t, \lambda)$ *and total size* $t \cdot \mathrm{poly}(\lambda)$.
  - $Z \leftarrow \mathsf{Puzzle.Gen}(s, K)$ *can be computed in (sequential) time* $\mathrm{poly}(\log t, \lambda)$.
  - $\mathsf{Puzzle.Sol}(Z)$ *can be computed in time* $t \cdot \mathrm{poly}(\lambda)$.

Time-Lock Puzzles with Pre-processing. We now define time-lock puzzles with pre-processing where the adversary is given *multiple* challenge puzzles. We require that the parallel time required to solve any of the puzzles is proportional to the time it takes to solve a puzzle honestly, up to some fixed polynomial loss. To the best of our knowledge, such a notion of time-lock puzzles was not considered previously.

Definition 4.2 (Time-Lock Puzzles with Pre-processing)**.** *A puzzle with pre-processing* (Puzzle.Preproc, Puzzle.Gen, Puzzle.Sol) *is a time-lock puzzle with gap* $\varepsilon < 1$ *if there exists a polynomial* $\underline{t}(\cdot)$*, such that for every polynomial* $t(\cdot) \geq \underline{t}(\cdot)$ *and every polysize adversary* $\mathcal{A} = \{\mathcal{A}_\lambda\}_{\lambda\in\mathbb{N}}$ *of depth* $\mathsf{dep}(\mathcal{A}_\lambda) \leq t^\varepsilon(\lambda)$*, there exists a negligible function* μ*, such that for every* $\lambda \in \mathbb{N}$*, and pairs of solutions* $(s_{1,0}, s_{1,1}), \ldots, (s_{k,0}, s_{k,1}) \in \{0,1\}^{\lambda+\lambda}$ *for* $k = \mathrm{poly}(\lambda)$*:*

$$\Pr\left[b \leftarrow \mathcal{A}_\lambda\left(P, \{Z_i\}_{i=1}^k\right) : \begin{array}{l} b \leftarrow \{0,1\}\ , \\ (P,K) \leftarrow \mathsf{Puzzle.Preproc}(1^t, 1^\lambda), \\ Z_i \leftarrow \mathsf{Puzzle.Gen}(s_{i,b}, K) \end{array}\right] \leq \frac{1}{2} + \mu(\lambda)\ .$$

Remark 4.3 (*On the security of time-lock puzzles with pre-processing*)*.* It is instructive to compare Definition 4.2 with a weaker definition where the adversary is given only *one* challenge puzzle. Indeed, there might exist a construction that satisfies this weaker definition but is insecure w.r.t. Definition 4.2.

Remark 4.4 (*On the efficiency of pre-processing phase*)*.* In Definition 4.1, we require that the pre-processing algorithm must be computable by a uniform circuit of dept $\mathrm{poly}(\log t, \lambda)$. One could consider an alternative, natural definition where this efficiency requirement is removed. We note that this definition is technically incomparable to weak time-lock puzzles (see Definition 3.8).

4.2 Reusable Randomized Encodings

Here we define reusable randomized encodings whose efficiency lies in between that of standard randomized encodings and succinct ones. Concretely, in reusable randomized encodings, most of the expensive encoding phase is done ahead of time in a preprocessing phase, independently of any specific (M, x). Subsequently, any (M, x) can be encoded as efficiently as in succinct randomized encodings.

Definition 4.5 (Reusable Randomized Encoding)**.** *A reusable randomized encoding scheme* RE *consists of a triple of algorithms* (RE.Preproc, RE.Encode, RE.Decode) *satisfying the following requirements.*

- *Syntax:*
  - $(\widehat{U}, K) \leftarrow \mathsf{RE.Preproc}(m, n, t, 1^\lambda)$ *is a probabilistic algorithm that takes as input bounds* m, n, t *on machine size, input size, and time, as well as a security parameter* 1^λ*. The algorithm outputs an encoded state* $\widehat{U}$ *and a short secret key* $K \in \{0,1\}^\lambda$*.*
  - $\widehat{M}(x) \leftarrow \mathsf{RE.Encode}(M, x, K)$ *is a probabilistic algorithm that takes as input a machine* M*, input* x*, and secret key* $K \in \{0,1\}^\lambda$*. The algorithm outputs a randomized encoding* $\widehat{M}(x)$*.*
  - $y \leftarrow \mathsf{RE.Decode}(\widehat{U}, \widehat{M}(x))$ *is a deterministic algorithm that takes as input an encoded state* $\widehat{U}$ *and a randomized encoding* $\widehat{M}(x)$*, and computes an output* $y \in \{0,1\}^\lambda$*.*
- *Functionality: for every* m, n, t*, security parameter* λ*,* n*-bit input* x*, and* m*-bit machine* M *such that* $M(x)$ *halts in* t *steps, it holds that* $y = M(x)$ *with overwhelming probability over the coins of* RE.Preproc, RE.Encode*.*

- *Security: there exists a PPT simulator* Sim *satisfying: for any poly-size distinguisher* $\mathcal{D} = \{\mathcal{D}_\lambda\}_{\lambda\in\mathbb{N}}$ *and polynomials* $m(\cdot), n(\cdot), t(\cdot)$*, there exists a negligible* $\mu(\cdot)$*, such that for any* $\lambda \in \mathbb{N}$*, machines and inputs* $(M_1, x_1,) \ldots, (M_k, x_k) \in \{0,1\}^{n(\lambda)+m(\lambda)}$*:*

$$\left| \Pr\left[\mathcal{D}_\lambda(\widehat{U}, \widehat{M}_1(x_1), \ldots, \widehat{M}_k(x_k)) = 1 : \begin{array}{r}(\widehat{U}, K) \leftarrow \mathsf{RE.Preproc}(m(\lambda), n(\lambda), t(\lambda), 1^\lambda) \\ \widehat{M}_i(x_i) \leftarrow \mathsf{RE.Encode}(M_i, x_i, K)\end{array}\right] - \right.$$
$$\left. \Pr\left[\mathcal{D}_\lambda(\widehat{U}, \widehat{S}_{y_1}, \ldots, \widehat{S}_{y_k}) = 1 : \widehat{U}, \left\{\widehat{S}_{y_i}\right\} \leftarrow \mathsf{Sim}\left(\{y_i\}, m(\lambda), n(\lambda), t(\lambda), 1^\lambda\right)\right]\right| \leq \mu(\lambda) \ ,$$

  where y_i *is the output of* $M_i(x_i)$ *after* $t(\lambda)$ *steps.*

- *Efficiency: For any* m, n, t *and machine* $M \in \{0,1\}^m$ *that on input* $x \in \{0,1\}^n$ *produces a* λ*-bit output in* t *steps:*
  - $(\widehat{U}, K) \leftarrow \mathsf{RE.Preproc}(m, n, t, 1^\lambda)$ *can be computed by a uniform circuit of depth* $\mathrm{polylog}(t) \cdot \mathrm{poly}(m, n, \lambda)$ *and total size* $t \cdot \mathrm{poly}(m, \lambda)$*.*
  - $\widehat{M}(x) \leftarrow \mathsf{RE.Encode}(M, x, K)$ *can be computed in sequential time (rather than just parallel time)* $\mathrm{polylog}(t) \cdot \mathrm{poly}(m, n, \lambda)$*.*
  - $\mathsf{RE.Decode}(\widehat{U}, \widehat{M}(x))$ *can be computed in (sequential) time* $t \cdot \mathrm{poly}(m, n, \lambda)$*.*

Theorem 4.6 ([GKP+13])**.** *Assuming sub-exponential hardness of the LWE problem, there exists a reusable randomized encoding scheme.*

4.3 Construction

We now describe our construction of time-lock puzzles with pre-processing from reusable randomized encodings.

Construction 4.7 (Time-Lock Puzzles with Pre-processing). Let RE be a reusable randomized encoding scheme. For $s \in \{0,1\}^\lambda$ and $t \leq 2^\lambda$, let M_s^t be a machine that, on any input $x \in \{0,1\}^\lambda$, outputs the string s after t steps (here we assume that $t \geq \lambda + \omega(1)$). Further assume that M_s^t is described by 3λ bits (which is possible for large enough λ).

The time-lock puzzle with pre-processing is constructed as follows:

- $\mathsf{Puzzle.Preproc}(1^t, 1^\lambda)$ samples $(\widehat{U}, K') \leftarrow \mathsf{RE.Preproc}(3\lambda, \lambda, t, 1^\lambda)$ and outputs $(P = \widehat{U}, K = K')$.
- $\mathsf{Puzzle.Gen}(s, K)$ samples $\widehat{M}_s^t(0^\lambda) \leftarrow \mathsf{RE.Encode}(M_{t,s}, 0^\lambda, t, 1^\lambda)$ and outputs $Z = \widehat{M}_s^t(0^\lambda)$.
- $\mathsf{Puzzle.Sol}(Z)$ outputs $\mathsf{RE.Decode}(P, Z)$.

Theorem 4.8. *Let* $\varepsilon < 1$*. Assume that, for every polynomial bounded function* $t(\cdot)$*, there exists a non-parallelizing language* $\mathcal{L} \in \mathit{Dtime}(t(\cdot))$ *with gap* ε*. Then, for any* $\underline{\varepsilon} < \varepsilon$*, Construction 4.7 is a time-lock puzzle with pre-processing with gap* $\underline{\varepsilon}$*.*

4.4 Proof of Security

The completeness and efficiency properties of the puzzle follow directly from the completeness and efficiency properties of the reusable randomized encoding scheme. We proceed to argue that the puzzle is a secure time-lock puzzle with pre-processing. Let $q_{\mathsf{RE}}(\lambda)$ be the fixed polynomial given by the efficiency property of the reusable randomized encoding scheme, bounding the time

required to compute a randomized encoding with machine size 3λ, input size λ, and any time bound $t \leq 2^\lambda$. Let $\underline{t}(\lambda) := (q_{\mathsf{RE}}(\lambda))^{1/\underline{\varepsilon}}$.

Assume towards contradiction that there exists a polysize adversary $\mathcal{A} = \{\mathcal{A}_\lambda\}_{\lambda\in\mathbb{N}}$, and a polynomially bounded function $t(\cdot) \geq \underline{t}(\cdot)$ such that $\mathsf{dep}(\mathcal{A}_\lambda) < t^{\underline{\varepsilon}}(\lambda)$ and for some polynomial $p(\cdot)$ and infinitely many $\lambda \in \mathbb{N}$ there exist pairs of solutions $(s_{1,0}, s_{1,1}), \ldots, (s_{k,0}, s_{k,1}) \in \{0,1\}^{\lambda+\lambda}$ such that:

$$\Pr\left[b \leftarrow \mathcal{A}_\lambda(Z) : \begin{array}{l} b \leftarrow \{0,1\} \ , \\ (P,K) \leftarrow \mathsf{Puzzle.Preproc}(1^t, 1^\lambda), \\ Z \leftarrow \mathsf{Puzzle.Gen}(s_{i,b}, K) \end{array}\right] \geq \frac{1}{2} + \frac{1}{p(\lambda)} \ . \tag{2}$$

Let $\mathcal{L} \in \mathrm{Dtime}(t(\cdot))$ be a non-parallelizing language with gap ε, which exists by assumption. We construct a polysize circuit family $\mathcal{B} = \{\mathcal{B}_\lambda\}_{\lambda\in\mathbb{N}}$ of depth $\mathsf{dep}(\mathcal{B}_\lambda) \leq t^\varepsilon(\lambda)$ that decides $\mathcal{L}_\lambda = \mathcal{L} \cap \{0,1\}^\lambda$ for any λ as above, contradicting the fact that $\mathcal{L}$ is non-parallelizing.

We start be constructing a *probabilistic* polysize adversary $\mathcal{B}'$ such that that $\mathsf{dep}(\mathcal{B}'_\lambda) = o(t^\varepsilon(\lambda))$ and $\mathcal{B}'_\lambda$ decides $\mathcal{L}_\lambda$ *with some noticeable advantage*. Fix any λ with corresponding $(s_{1,0}, s_{1,1}), \ldots, (s_{k,0}, s_{k,1}) \in \{0,1\}^{\lambda+\lambda}$ as above. For every $i \in [k]$, let $M^{\mathcal{L},t}_{s_{i,0},s_{i,1}}$ be a machine that, on input $x \in \{0,1\}^\lambda$, outputs $s_{i,1}$ if $x \in \mathcal{L}$ and and $s_{i,0}$ if $x \notin \mathcal{L}$, after exactly $t(\lambda)$ steps. Such machines indeed exist since $\mathcal{L} \in \mathrm{Dtime}(t(\cdot))$. Further assume that $M^{\mathcal{L},t}_{s_{i,0},s_{i,1}}$ is described by 3λ bits (which is possible for large enough λ), and thus has the same description length as $M^t_{s_{i,b}}$.

Given input $x \in \{0,1\}^\lambda$, to decide if $x \in \mathcal{L}$, the randomized $\mathcal{B}'_\lambda$ acts as follows:

- Sample $(\widehat{U}, K) \leftarrow \mathsf{RE.Preproc}(1^{3\lambda}, 1^\lambda, t, 1^\lambda)$.
- Sample $Z_i := \widehat{M}^{\mathcal{L},t}_{s_{i,0},s_{i,1}}(x) \leftarrow \mathsf{RE.Encode}(M^{\mathcal{L},t}_{s_{i,0},s_{i,1}}, x, K)$ for every $i \in [k]$.
- Obtain $b \leftarrow \mathcal{A}_\lambda\left(\widehat{U}, \{Z_i\}_{i=1}^k\right)$ and output b.

First, note that $\mathcal{B}'$ is of polynomial size and its depth is given by:

$$\mathsf{dep}(\mathcal{B}'_\lambda) = q_{\mathsf{RE}}(\lambda) + \mathsf{dep}(\mathcal{A}_\lambda) = \underline{t}^{\underline{\varepsilon}}(\lambda) + t^{\underline{\varepsilon}}(\lambda) \leq 2t^{\underline{\varepsilon}}(\lambda) = o(t^\varepsilon(\lambda)) \ .$$

We next show that $\mathcal{B}'$ distinguishes instances $x \in \mathcal{L}$ from instances $x \notin \mathcal{L}$ with noticeable advantage. For any $x \in \{0,1\}^\lambda$, let $b \in \{0,1\}$ indicate whether $x \in \mathcal{L}_\lambda$ we have that $s_{i,b} = M^{\mathcal{L},t}_{s_{i,0},s_{i,1}}(x) = M^t_{s_{i,b}}(0^\lambda)$. Therefore, by the security of the reusable randomized encoding scheme there exists a PPT simulator Sim and a negligible function $\mu(\cdot)$ such that for any $x \in \{0,1\}^\lambda$:

$$\begin{aligned}
&\Pr[\mathcal{B}'_\lambda(x) = 1] = \\
&\Pr\left[\mathcal{A}_\lambda\left(\widehat{U}, \left\{\widehat{M}^{\mathcal{L},t}_{s_{i,0},s_{i,1}}(x)\right\}_{i=1}^k\right) = 1 : \begin{array}{l} (\widehat{U}, K) \leftarrow \mathsf{RE.Preproc}\left(1^{3\lambda}, 1^\lambda, t, 1^\lambda\right), \\ \widehat{M}^{\mathcal{L},t}_{s_{i,0},s_{i,1}}(x) \leftarrow \mathsf{RE.Encode}\left(M^{\mathcal{L},t}_{s_{i,0},s_{i,1}}, x, K\right) \end{array}\right] = \\
&\Pr\left[\mathcal{A}_\lambda\left(\widehat{U}, \left\{\widehat{S}_{s_{i,b}}\right\}_{i=1}^k\right) = 1 : \widehat{U}, \left\{\widehat{S}_{s_{i,b}}\right\}_{i=1}^k \leftarrow \mathsf{Sim}\left(\{s_{i,b}\}_{i=1}^k, 1^{3\lambda}, 1^\lambda, t(\lambda), 1^\lambda\right)\right] \pm \mu(\lambda) \ .
\end{aligned}$$

And:

$$\begin{aligned}
&\Pr\left[\mathcal{A}_\lambda\left(P, \{Z_i\}_{i=1}^k\right) = 1 : \begin{array}{l} (P,K) \leftarrow \mathsf{Puzzle.Preproc}(1^t, 1^\lambda), \\ Z_i \leftarrow \mathsf{Puzzle.Gen}(s_{i,b}, K) \end{array}\right] = \\
&\Pr\left[\mathcal{A}_\lambda\left(\widehat{U}, \left\{\widehat{M}^t_{s_{i,b}}(0^\lambda)\right\}_{i=1}^k\right) = 1 : \begin{array}{l} (\widehat{U}, K) \leftarrow \mathsf{RE.Preproc}\left(1^{3\lambda}, 1^\lambda, t, 1^\lambda\right), \\ \widehat{M}^t_{s_{i,b}}(0^\lambda) \leftarrow \mathsf{RE.Encode}(M^t_{s_{i,b}}, x, K) \end{array}\right] = \\
&\Pr\left[\mathcal{A}_\lambda\left(\widehat{U}, \left\{\widehat{S}_{s_{i,b}}\right\}_{i=1}^k\right) = 1 : \widehat{U}, \left\{\widehat{S}_{s_{i,b}}\right\}_{i=1}^k \leftarrow \mathsf{Sim}\left(\{s_{i,b}\}_{i=1}^k, 1^{3\lambda}, 1^\lambda, t(\lambda), 1^\lambda\right)\right] \pm \mu(\lambda) \ .
\end{aligned}$$

It follows by our assumption towards contradiction (Equation 2) that for large enough λ and any $x \in \mathcal{L}_\lambda$, $\bar{x} \notin \mathcal{L}_\lambda$:

$$\left|\Pr[\mathcal{B}'_\lambda(x) = 1] - \Pr[\mathcal{B}'_\lambda(\bar{x}) = 1]\right| \geq \frac{2}{p(\lambda)} - 2\mu(\lambda) \geq \frac{1}{p(\lambda)} \ .$$

To obtain the required $\mathcal{B}$ that deterministically works for any $x \in \{0,1\}^\lambda$, we rely on the standard parallel repetition argument showing that BPP/*poly* $\subseteq$ P/*poly* [Gol01b, Theorem 1.3.7]. Note that following this argument, $|\mathcal{B}| = \text{poly}(\mathcal{B}') = \text{poly}(\lambda)$ and $\mathsf{dep}(\mathcal{B}) = \mathsf{dep}(\mathcal{B}) + \text{polylog}(\lambda) = o(t^\varepsilon(\lambda))$ contradicting the fact that $\mathcal{L}$ is non-parallelizing.

5 Proofs of Work

In this section, we define proofs of work and construct them based on succinct randomized encodings. The security of the construction also assumes the existence of a hard language that is non-amortizing in the worst case.

Definition. In proofs of work, we require that the minimal time required to solve k random puzzles is proportional to k times the time it takes to solve the puzzle honestly, up to some fixed polynomial loss. We give the formal definition below.

Definition 5.1 (Proofs of Work)**.** *A puzzle* $(\mathsf{Puzzle.Gen}, \mathsf{Puzzle.Sol})$ *is a proof of work with gap* $\varepsilon < 1$ *if there exists a polynomial* $\underline{t}(\cdot)$*, such that for every polynomials* $t(\cdot) \geq \underline{t}(\cdot)$ *and* $k(\cdot)$*, and every polysize adversary* $\mathcal{A} = \{\mathcal{A}_\lambda\}_{\lambda\in\mathbb{N}}$ *such that* $|\mathcal{A}_\lambda| \leq k(\lambda) \cdot t^\varepsilon(\lambda)$*, there exists a negligible function* μ *such that for every* $\lambda \in \mathbb{N}$*:*

$$\Pr\left[(s_1, \ldots, s_{k(\lambda)}) \leftarrow \mathcal{A}_\lambda(Z_1, \ldots, Z_{k(\lambda)}) : \begin{array}{l} s_i \leftarrow \{0,1\}^\lambda \ , \\ Z_i \leftarrow \mathsf{Puzzle.Gen}(t(\lambda), s_i) \end{array}\right] \leq \mu(\lambda) \ .$$

Construction. Our construction of proofs of work is the same as the construction of time-lock puzzles given in Section 3.3.

5.1 Security

In order to prove security of our construction, we will rely on a specific kind of language with *worst-case hardness* that we refer to as "non-amortizing" language. We first present its definition. The definition addresses non-uniform families of circuits, and can be naturally augmented to the uniform case.

Non-amortizing language. A poly-time decidable language $\mathcal{L}$ is *non-amortizing* if solving multiple instances together is not significantly easier than solving each of them separately. That is, the circuit size of any family of circuits deciding the k-fold direct product $\mathcal{L}^k$ is as large as k times the time required to decide $\mathcal{L}$ up to some fixed polynomial loss.

Definition 5.2 (Non-Amortizing Language)**.** *A language* $\mathcal{L} \in \mathit{Dtime}(t(\cdot))$ *is non-amortizing with gap* $\varepsilon < 1$ *if for every polynomial* $k(\cdot)$ *and family of non-uniform polysize circuits* $\mathcal{B} = \{\mathcal{B}_\lambda\}_{\lambda\in\mathbb{N}}$ *such that* $\mathcal{B}_\lambda$ *operates on inputs in* $\{0,1\}^{\lambda\times k(\lambda)}$ *and* $|\mathcal{B}_\lambda| \leq k(\lambda) \cdot (t(\lambda))^\varepsilon$*, and every large enough* λ*,* $\mathcal{B}_\lambda$ *fails to decide* $\mathcal{L}^k_\lambda = \mathcal{L}^{k(\lambda)} \cap \{0,1\}^{\lambda\times k(\lambda)}$*.*

The security of our construction is stated in the following theorem.

Theorem 5.3. *Let* $\varepsilon < 1$*. Assume that, for every polynomial bounded function* $t(\cdot)$*, there exists a non-amortizing language* $\mathcal{L} \in \mathit{Dtime}(t(\cdot))$ *with gap* ε*. Then, for any* $\underline{\varepsilon} < \varepsilon$*, Construction 3.5 is a proof of work with gap* $\underline{\varepsilon}$*.*

5.2 Proof of Theorem 5.3

The completeness and efficiency properties of the puzzle follows directly from the completeness and efficiency properties of the succinct randomized encoding scheme. We proceed to argue that that the puzzle is a secure proof of work. Let $q_{\mathsf{RE}}(\lambda)$ be the fixed polynomial given by the efficiency property of the succinct randomized encoding scheme, bounding the time required to compute a randomized encoding with machine size 3λ, input size λ, and any time bound $t \leq 2^{\lambda}$. Let $\underline{t}(\lambda) := (q_{\mathsf{RE}}(\lambda))^{1/\underline{\varepsilon}}$.

Assume towards contradiction that there exists an adversary $\mathcal{A} = \{\mathcal{A}_\lambda\}_{\lambda\in\mathbb{N}}$, and polynomially bounded functions $t(\cdot) \geq \underline{t}(\cdot)$ and $k(\cdot)$ such that for some polynomial $p(\cdot)$ and infinitely many $\lambda \in \mathbb{N}$, $|\mathcal{A}_\lambda| < k(\lambda) \cdot (t(\lambda))^{\underline{\varepsilon}}$, and:

$$\Pr\left[\left(s_1,\ldots,s_{k(\lambda)}\right) \leftarrow \mathcal{A}_\lambda(Z_1,\ldots,Z_{k(\lambda)}) : \begin{array}{c} s_i \leftarrow \{0,1\}^\lambda \ , \\ Z_i \leftarrow \mathsf{Puzzle.Gen}(t(\lambda), s_i) \end{array}\right] \geq \frac{1}{p(\lambda)} \ .$$

Let $\mathcal{L} \in \mathrm{Dtime}(t(\cdot))$ be a non-amortizing language with gap ε, which exists by assumption. We construct a polysize circuit family $\mathcal{B} = \{\mathcal{B}_\lambda\}_{\lambda\in\mathbb{N}}$ operating on inputs in $\{0,1\}^{\lambda\times k(\lambda)}$ and of size $|\mathcal{B}_\lambda| \leq k(\lambda)\cdot(t(\lambda))^{\varepsilon}$ such that, for any λ as above, $\mathcal{B}$ decides $\mathcal{L}^k_\lambda = \mathcal{L}^{k(\lambda)} \cap \{0,1\}^{\lambda\times k(\lambda)}$, contradicting the fact that $\mathcal{L}$ is non-amortizing.

We start be constructing a *probabilistic* adversary $\mathcal{B}'$ such that that $|\mathcal{B}'_\lambda| = o\left(k\left(\lambda\right)\cdot(t(\lambda))^{\varepsilon}\right)$ and $\mathcal{B}'_\lambda$ decides $\mathcal{L}_\lambda$ *with some noticeable advantage*. Then, we conclude the proof using a standard parallel repetition argument.

Fix any λ as above and choose random $s_1,\ldots,s_k \in \{0,1\}^\lambda$. Let $M^{\mathcal{L},t}_{s_1},\ldots,M^{\mathcal{L},t}_{s_k}$ be k machines where the ith machine $M^{\mathcal{L},t}_{s_i}$, on input $x_i \in \{0,1\}^\lambda$, outputs s_i if $x_i \in \mathcal{L}$ and a randomly chosen $r'_i \in \{0,1\}^\lambda$ if $x_i \notin \mathcal{L}$, after exactly $t(\lambda)$ steps. Note that such machines indeed exist since $\mathcal{L} \in \mathrm{Dtime}(t(\cdot))$. Further assume that $M^{\mathcal{L},t}_{s_i}$ is described by the same number of bits as $M^t_{s_b}$, e.g. 3λ bits (which is possible for large enough λ).

Given input $(x_1,\ldots,x_k) \in \{0,1\}^{\lambda\times k(\lambda)}$, to decide if $(x_1,\ldots,x_k) \in \mathcal{L}^k_\lambda$, the randomized $\mathcal{B}'_\lambda$ acts as follows:

- Sample $Z_i := \widehat{M}^{\mathcal{L},t}_{s_i}(x) \leftarrow \mathsf{RE.Encode}(M^{\mathcal{L},t}_{s_i}, x_i, t(\lambda), 1^\lambda)$.
- Obtain $(s'_1,\ldots,s'_k) \leftarrow \mathcal{A}_\lambda(Z_1,\ldots,Z_k)$. If $s'_i = s_i$ for every $i \in [k]$, output 1; else output 0.

First, note that the size of $\mathcal{B}'$ is given by:

$$|\mathcal{B}'_\lambda| = k(\lambda)\cdot q_{\mathsf{RE}}(\lambda) + |\mathcal{A}_\lambda| = k(\lambda)\cdot(\underline{t}(\lambda))^{\underline{\varepsilon}} + k(\lambda)\cdot(t(\lambda))^{\underline{\varepsilon}} \leq 2k(\lambda)\cdot(t(\lambda))^{\underline{\varepsilon}} = o\left(k\left(\lambda\right)\cdot\left(t\left(\lambda\right)\right)^{\varepsilon}\right) \ .$$

We next show that $\mathcal{B}'$ distinguishes instances $(x_1,\ldots,x_k) \in \mathcal{L}^k$ from instances $(x_1,\ldots,x_k) \notin \mathcal{L}^k$ with noticeable advantage. By the security of the randomized encoding scheme there exists a PPT simulator Sim and a negligible function $\mu(\cdot)$ such that:

$$\begin{aligned}
&\Pr[\mathcal{B}'_\lambda(x_1,\ldots,x_k) = 1] = \\
&\Pr\left[\mathcal{A}_\lambda\left(\left\{\widehat{M}^{\mathcal{L},t}_{s_i}(x_i)\right\}_{i=1}^k\right) = (s_1,\ldots,s_k) : \widehat{M}^{\mathcal{L},t}_{s_i}(x_i) \leftarrow \mathsf{RE.Encode}(M^{\mathcal{L},t}_{s_i}, x_i, t(\lambda), 1^\lambda)\right] = \\
&\Pr\left[\mathcal{A}_\lambda\left(\left\{\widehat{S}_{a_i}\right\}_{i=1}^k\right) = (s_1,\ldots,s_k) : \begin{array}{c} a_i = M^{\mathcal{L},t}_{s_i}(x_i) \\ \widehat{S}_{a_i} \leftarrow \mathsf{Sim}\left(a_i, 1^{3\lambda}, 1^\lambda, t(\lambda), 1^\lambda\right) \end{array}\right] \pm k\cdot\mu(\lambda),
\end{aligned}$$

and:

$$\Pr\left[\mathcal{A}_\lambda\left(\{Z_i\}_{i=1}^k\right) = (s_1,\ldots,s_k) : Z_i \leftarrow \mathsf{Puzzle.Gen}(t(\lambda), a_i)\right] =$$
$$\Pr\left[\mathcal{A}_\lambda\left(\left\{\widehat{M}^t_{a_i}\left(0^\lambda\right)\right\}_{i=1}^k\right) = (s_1,\ldots,s_k) : \widehat{M}^t_{a_i}(0^\lambda) \leftarrow \mathsf{RE.Encode}(M^t_{a_i}, 0^\lambda, t(\lambda), 1^\lambda)\right] =$$
$$\Pr\left[\mathcal{A}_\lambda\left(\left\{\widehat{S}_{a_i}\right\}_{i=1}^k\right) = (s_1,\ldots,s_k) : \widehat{S}_{a_i} \leftarrow \mathsf{Sim}\left(a_i, 1^{3\lambda}, 1^\lambda, t(\lambda), 1^\lambda\right)\right] \pm k\cdot\mu(\lambda).$$

It follows by our assumption towards contradiction that for large enough λ and any $(x_1,\ldots,x_k) \in \mathcal{L}^k_\lambda$, $(\bar{x}_1,\ldots,\bar{x}_k) \notin \mathcal{L}^k_\lambda$:

$$\left|\Pr[\mathcal{B}'_\lambda(x_1,\ldots,x_k)=1] - \Pr[\mathcal{B}'_\lambda(\bar{x}_1,\ldots,\bar{x}_k)=1]\right| \geq \frac{2}{p(\lambda)} - 2k\cdot\mu(\lambda) \geq \frac{1}{p(\lambda)} \ .$$

To obtain the required $\mathcal{B}$ that deterministically works for any $(x_1,\ldots,x_k) \in \{0,1\}^{\lambda\times k(\lambda)}$, we rely on the standard parallel repetition argument showing that BPP/*poly* $\subseteq$ P/*poly* [Gol01b, Theorem 1.3.7].

Acknowledgments. We thank Ryan Williams for his patient and thorough answers to our questions. We thank Benny Applebaum for valuable comments. We also thank Juan Garay for referring us to [GMPY11].

References

[AIK06] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Computationally private randomizing polynomials and their applications. *Computational Complexity*, 15(2):115–162, 2006.

[AJ15] Prabhanjan Ananth and Abhishek Jain. Indistinguishability obfuscation from compact functional encryption. In *Crypto*, 2015.

[Bar02] Boaz Barak. A probabilistic-time hierarchy theorem for "slightly non-uniform" algorithms. In *Randomization and Approximation Techniques, 6th International Workshop, RANDOM 2002, Cambridge, MA, USA, September 13-15, 2002, Proceedings*, pages 194–208, 2002.

[BGL+15] Nir Bitansky, Sanjam Garg, Huijia Lin, Rafael Pass, and Sidharth Telang. Succinct randomized encodings and their applications. In *Symposium on Theory of Computing, STOC 2015*, 2015.

[BN00] Dan Boneh and Moni Naor. Timed commitments. In *Advances in Cryptology - CRYPTO 2000, 20th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2000, Proceedings*, pages 236–254, 2000.

[BV15] Nir Bitansky and Vinod Vaikuntanathan. Indistinguishability obfuscation from functional encryption. In *FOCS*, 2015.

[CHJV15] Ran Canetti, Justin Holmgren, Abhishek Jain, and Vinod Vaikuntanathan. Indistinguishability obfuscation of iterated circuits and ram programs. In *Symposium on Theory of Computing, STOC 2015*, 2015.

[CKV10] Kai-Min Chung, Yael Tauman Kalai, and Salil P. Vadhan. Improved delegation of computation using fully homomorphic encryption. In *Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings*, pages 483–501, 2010.

[DN92] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In *Advances in Cryptology - CRYPTO '92, 12th Annual International Cryptology Conference, Santa Barbara, California, USA, August 16-20, 1992, Proceedings*, pages 139–147, 1992.

[GHR95] Raymond Greenlaw, H. James Hoover, and Walter L. Ruzzo. *Limits to Parallel Computation: P-completeness Theory.* Oxford University Press, Inc., New York, NY, USA, 1995.

[GKP+13] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Reusable garbled circuits and succinct functional encryption. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 555–564, 2013.

[GMPY11] Juan A. Garay, Philip D. MacKenzie, Manoj Prabhakaran, and Ke Yang. Resource fairness and composability of cryptographic protocols. *J. Cryptology*, 24(4):615–658, 2011.

[Gol01a] Oded Goldreich. *The Foundations of Cryptography - Volume 1, Basic Techniques.* Cambridge University Press, 2001.

[Gol01b] Oded Goldreich. *The Foundations of Cryptography - Volume 1, Basic Techniques.* Cambridge University Press, 2001.

[HS65] Juris Hartmanis and Richard Stearns. On the computational complexity of algorithms. 117:285–306, 1965.

[IK00] Yuval Ishai and Eyal Kushilevitz. Randomizing polynomials: A new representation with applications to round-efficient secure computation. In *41st Annual Symposium on Foundations of Computer Science, FOCS 2000, 12-14 November 2000, Redondo Beach, California, USA*, pages 294–304, 2000.

[IW01] Russell Impagliazzo and Avi Wigderson. Randomness vs time: Derandomization under a uniform assumption. *J. Comput. Syst. Sci.*, 63(4):672–688, 2001.

[IW14] Yuval Ishai and Hoeteck Wee. Partial garbling schemes and their applications. In *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, pages 650–662, 2014.

[Jag15] Tibor Jager. How to build time-lock encryption. *IACR Cryptology ePrint Archive*, 2015:478, 2015.

[JJ99] Markus Jakobsson and Ari Juels. Proofs of work and bread pudding protocols. In *Secure Information Networks: Communications and Multimedia Security, IFIP TC6/TC11 Joint Working Conference on Communications and Multimedia Security (CMS '99), September 20-21, 1999, Leuven, Belgium*, pages 258–272, 1999.

[KLW15] Venkata Koppula, Allison Bishop Lewko, and Brent Waters. Indistinguishability obfuscation for turing machines with unbounded memory. In *Symposium on Theory of Computing, STOC 2015*, 2015.

[LGR15] Jia Liu, Flavio Garcia, and Mark Ryan. Time-release protocol from bitcoin and witness encryption for SAT. *IACR Cryptology ePrint Archive*, 2015:482, 2015.

[LPST15] Huijia Lin, Rafael Pass, Karn Seth, and Sidharth Telang. Output-compressing randomized encodings and applications. *IACR Cryptology ePrint Archive*, 2015:720, 2015.

[May93] Timothy C. May. Timed-release crypto, 1993. http://www.hks.net/cpunks/cpunks-0/1460.html.

[MMV11] Mohammad Mahmoody, Tal Moran, and Salil P. Vadhan. Time-lock puzzles in the random oracle model. In *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, pages 39–50, 2011.

[Nak00] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2000. http://bitcoin.org/bitcoin.pdf.

[RSW00] Ronald L. Rivest, Adi Shamir, and David A. Wagner. Time-lock puzzles and timed-release crypto. Technical Report MIT/LCS/TR-684, MIT, February 2000.

[Yao82] Andrew Chi-Chih Yao. Theory and applications of trapdoor functions (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science, Chicago, Illinois, USA, 3-5 November 1982*, pages 80–91, 1982.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th Annual Symposium on Foundations of Computer Science, Toronto, Canada, 27-29 October 1986*, pages 162–167. IEEE Computer Society, 1986.

A Time-Lock Puzzles from Message-hiding Encodings

We observe that a different construction of time-lock puzzles can be obtained from a relaxation of randomized encodings called *message-hiding encodings* [IW14, KLW15], at the price of assuming stronger non-parallelizing languages with *average-case hardness*.

A.1 Message-Hiding Encodings

In message-hiding encodings, a secret message m is encoded with respect to some public predicate P and input x. Decoding m is possible only if $P(x) = 1$, and otherwise the encoding computationally hides m. As in succinct randomized encodings, the complexity of encoding the message is essentially independent of the complexity of P.

Definition A.1 (Message-Hiding Encoding [KLW15])**.** *A message-hiding encoding scheme* MHE *consists of two algorithms* $(\mathsf{MHE.Encode}, \mathsf{MHE.Decode})$ *satisfying the following requirements.*

- *Syntax:*
  - $\widehat{m} \leftarrow \mathsf{MHE.Encode}(P, x, t, m)$ *is a probabilistic algorithm that takes as input a machine* P*, input* x*, time bound* t*, and message* $m \in \{0,1\}^\lambda$*, where* λ *is the security parameter. The algorithm outputs an encoding* $\widehat{m}$*.*

– $\tilde{m} \leftarrow \mathsf{MHE.Decode}(\widehat{m})$ *is a deterministic algorithm that takes as input an encoding* $\widehat{m}$ *and computes a message* $\tilde{m} \in \{0,1\}^{\lambda}$.

- *Functionality: for every input* x *and machine* M *such that* P *accepts* x *after* t *steps, it holds that* $\tilde{m} = m$ *with overwhelming probability over the coins of* MHE.Encode.

- *Security: for any poly-size distinguisher* $\mathcal{D} = \{\mathcal{D}_\lambda\}_{\lambda\in\mathbb{N}}$ *and polynomials* $m(\cdot), n(\cdot), t(\cdot)$, *there exists a negligible* $\mu(\cdot)$, *such that for any* $\lambda \in \mathbb{N}$, *machine* $P \in \{0,1\}^{m(\lambda)}$, *messages* m_0, m_1, *and input* $x \in \{0,1\}^{n(\lambda)}$ *such that* P *rejects* x *after* t *steps:*

$$\Big|\Pr[\mathcal{D}_\lambda(\widehat{m}_0) = 1 : \widehat{m}_0 \leftarrow \mathsf{MHE.Encode}(P, x, t(\lambda), m_0)] - \Pr[\mathcal{D}_\lambda(\widehat{m}_1) = 1 : \widehat{m}_1 \leftarrow \mathsf{MHE.Encode}(P, x, t(\lambda), m_1)]\Big| \leq \mu(\lambda) \ .$$

- *Efficiency: For any machine* M *that on input* x *halts and outputs a bit in* t *steps, and any* $m \in \{0,1\}^{\lambda}$:
  - $\mathsf{MHE.Encode}(P, x, t, m)$ *can be computed in (sequential) time* $\mathrm{polylog}(t)\cdot\mathrm{poly}(|P|, |x|, \lambda)$.
  - $\mathsf{RE.Decode}(\widehat{m})$ *can be computed in (sequential) time* $t \cdot \mathrm{poly}(|P|, |x|, \lambda)$.

Message-hiding encodings are a relaxation of succinct randomized encoding and can thus be constructed from *indistinguishability obfuscation* [KLW15].

A.2 Non-Parallelizing Languages with Average-Case Hardness

In addition to message-hiding encodings, we also assume the existence of non-parallelizing languages with *average-case hardness*. These are defined similarly to standard non-parallelizing languages (Definition 3.6) except that we require that shallow circuits fail to decide such languages on instances samples from some distribution $\mathcal{X}$ with probability that is significantly higher than half. We require that for every difficulty parameter t and input length there exists a non-parallelizing language $\mathcal{L}$ decidable in time t and that there is a uniform way to decide the language and sample hard instances for every t.

Definition A.2 (Average-Case Non-Parallelizing Language Ensemble)**.** *A collection of languages* $\{\mathcal{L}_{\lambda,t}\}_{\lambda,t\in\mathbb{N}}$ *is average-case non-parallelizing with gap* $\varepsilon < 1$ *if the following properties are satisfied.*

- *Completeness: For every* $\lambda, t \in \mathbb{N}$, *we have that* $\mathcal{L}_{\lambda,t} \subseteq \{0,1\}^{\lambda}$ *and there exists a decision algorithm* $\mathcal{L}$ *such that for every* $\lambda, t \in \mathbb{N}$ *and every* $x \in \{0,1\}^{\lambda}$, $\mathcal{L}(t, x)$ *runs in time* t *and outputs* 1 *iff* $x \in \mathcal{L}_{\lambda,t}$.

- *Average-Case Non-Parallelizing: There exists an efficient sampler* $\mathcal{X}$ *such that for every family of polysize circuits* $\mathcal{B} = \{\mathcal{B}_\lambda\}_{\lambda\in\mathbb{N}}$ *there exists a negligible function* μ *such that for every* $\lambda, t \in \mathbb{N}$ *if* $\mathsf{dep}(\mathcal{B}_\lambda) \leq t^{\varepsilon}$ *then:*

$$\Pr_{x\leftarrow\mathcal{X}_\lambda(1^\lambda,t)}[\mathcal{B}_\lambda(x) = \mathcal{L}(t, x)] \leq \frac{1}{2} + \mu(\lambda) \ .$$

A.3 Construction

We describe the construction of time-lock puzzles based on a message-hiding encoding scheme and an average-case non-parallelizing language ensemble. Note that unlike Construction 3.5, this construction does depend on the specific choice of non-parallelizing language.

Construction A.3 (Time-Lock Puzzles from Message-Hiding Encodings). Let MHE be a message-hiding encoding scheme, let $\{\mathcal{L}_{\lambda,t}\}_{\lambda,t\in\mathbb{N}}$ be a collection of average-case non-parallelizing languages with decision algorithm $\mathcal{L}$ and input sampler $\mathcal{X}$. For $s \in \{0,1\}^\lambda$ and $t \leq 2^\lambda$, let P^t be the decision predicate $\mathcal{L}(t,x)$ for the language $\mathcal{L}_{\lambda,t}$.

The time-lock puzzle is constructed as follows:

- $\mathsf{Puzzle.Gen}(t,s)$ samples a hard instance $x \leftarrow \mathcal{X}_\lambda(1^\lambda,t)$, samples:

$$\widehat{s}_1 \leftarrow \mathsf{MHE.Encode}(P^t,x,t,s) \quad , \quad \widehat{s}_0 \leftarrow \mathsf{MHE.Encode}(1-P^t,x,t,s) \ ,$$

  and outputs $Z = (t,x,\widehat{s}_0,\widehat{s}_1)$.

- $\mathsf{Puzzle.Sol}(t,x,\widehat{s}_0,\widehat{s}_1)$ computes $b = P^t(x)$ and outputs $\mathsf{MHE.Decode}(\widehat{s}_b)$.

A.4 Proof of Security

The security of Construction A.3 is stated in the following theorem.

Theorem A.4. *If $\{\mathcal{L}_{\lambda,t}\}_{\lambda,t\in\mathbb{N}}$ is average-case non-parallelizing with gap $\varepsilon < 1$, then, for any $\underline{\varepsilon} < \varepsilon$, Construction A.3 is a time-lock puzzle with gap $\underline{\varepsilon}$.*

A.4.1 Proof of Theorem A.4

The completeness and efficiency properties of the puzzle follow directly from the completeness and efficiency properties of the message-hiding encoding scheme. We proceed to argue that the puzzle is a secure time-lock puzzle. Let $q_{\mathsf{MHE}}(\lambda)$ be the fixed polynomial given by the efficiency property of the message-hiding encoding scheme, bounding the time required to compute an encoding of the predicate P^t, an input of size λ, and any time bound $t \leq 2^\lambda$. Let $\underline{t}(\lambda) := (q_{\mathsf{MHE}}(\lambda))^{1/\underline{\varepsilon}}$.

Assume towards contradiction that there exists a polysize adversary $\mathcal{A} = \{\mathcal{A}_\lambda\}_{\lambda\in\mathbb{N}}$, and a polynomially bounded function $t(\cdot) \geq \underline{t}(\cdot)$ such that $\mathsf{dep}(\mathcal{A}_\lambda) < t^{\underline{\varepsilon}}(\lambda)$ and for some polynomial p_1 and infinitely many $\lambda \in \mathbb{N}$ there exists a pair of solutions $s_0, s_1 \in \{0,1\}^\lambda$ such that:

$$\Pr\left[b \leftarrow \mathcal{A}_\lambda(Z) : \begin{array}{l} b \leftarrow \{0,1\} \ , \\ Z \leftarrow \mathsf{Puzzle.Gen}(t(\lambda),s_b) \end{array}\right] \geq \frac{1}{2} + \frac{1}{p_1(\lambda)} \ .$$

By construction we can rewrite the above as:

$$\Pr\left[\begin{array}{l} b \leftarrow \{0,1\} \ , \\ x \leftarrow \mathcal{X}_\lambda(1^\lambda,t(\lambda)) \\ \widehat{s}_1 \leftarrow \mathsf{MHE.Encode}(P^{t(\lambda)},x,t(\lambda),s_b) \\ \widehat{s}_0 \leftarrow \mathsf{MHE.Encode}(1-P^{t(\lambda)},x,t(\lambda),s_b) \\ b \leftarrow \mathcal{A}_\lambda(t(\lambda),x,\widehat{s}_0,\widehat{s}_1) \end{array}\right] \geq \frac{1}{2} + \frac{1}{p_1(\lambda)} \ .$$

For every $x \in \{0,1\}^\lambda$ let $b_x = \mathcal{L}(t(\lambda),x)$. Since $\{\mathcal{L}_{\lambda,t}\}$ is average-case non-parallelizing, we have that for some negligible function μ_1:

$$\left|\Pr_{x\leftarrow\mathcal{X}_\lambda(1^\lambda,t(\lambda))}[b_x = 1] - \frac{1}{2}\right| \leq \mu_1(\lambda) \ ,$$

and therefore for some polynomial p_2:

$$\Pr\left[\begin{array}{l} x \leftarrow \mathcal{X}_\lambda(1^\lambda, t(\lambda)) \\ \widehat{s}_1 \leftarrow \mathsf{MHE.Encode}(P^{t(\lambda)}, x, t(\lambda), s_{b_x}) \\ \widehat{s}_0 \leftarrow \mathsf{MHE.Encode}(1 - P^{t(\lambda)}, x, t(\lambda), s_{b_x}) \\ b_x \leftarrow \mathcal{A}_\lambda(t(\lambda), x, \widehat{s}_0, \widehat{s}_1) \end{array}\right] \geq \frac{1}{2} + \frac{1}{p_2(\lambda)} \quad . \tag{3}$$

We construct a polysize circuit family $\mathcal{B} = \{\mathcal{B}_\lambda\}_{\lambda \in \mathbb{N}}$ of depth $\mathsf{dep}(\mathcal{B}_\lambda) \leq t^\varepsilon(\lambda)$ such that for some polynomial p_3 and for any λ as above:

$$\Pr_{x \leftarrow \mathcal{X}_\lambda(1^\lambda, t(\lambda))}[\mathcal{B}_\lambda(x) = \mathcal{L}(t(\lambda), x)] \geq \frac{1}{2} + \frac{1}{p_3(\lambda)} \quad . \tag{4}$$

contradicting the fact that $\{\mathcal{L}_{\lambda,t}\}$ is average-case non-parallelizing.

Note that it is sufficient to construct a *probabilistic* polysize adversary $\mathcal{B}$ as above. Fix any λ as above with corresponding $s_0, s_1 \in \{0,1\}^\lambda$. Given input $x \leftarrow \mathcal{X}_\lambda(1^\lambda, t(\lambda))$, to decide if $x \in \mathcal{L}_{\lambda,t}$ $\mathcal{B}_\lambda$ acts as follows:

- Sample:

$$\widehat{s}_1 \leftarrow \mathsf{MHE.Encode}(P^t, x, t, s_1) \quad , \quad \widehat{s}_0 \leftarrow \mathsf{MHE.Encode}(1 - P^t, x, t, s_0) \quad .$$

- Obtain $b \leftarrow \mathcal{A}_\lambda(t(\lambda), x, \widehat{s}_0, \widehat{s}_1)$ and output b.

First, note that $\mathcal{B}$ is of polynomial size and its depth is given by:

$$\mathsf{dep}(\mathcal{B}_\lambda) = q_{\mathsf{MHE}}(\lambda) + \mathsf{dep}(\mathcal{A}_\lambda) = \underline{t}^{\underline{\varepsilon}}(\lambda) + t^{\underline{\varepsilon}}(\lambda) \leq 2t^{\underline{\varepsilon}}(\lambda) = o(t^\varepsilon(\lambda)) \quad .$$

We next show that $\mathcal{B}$ satisfies (4). For every $x \in \{0,1\}^\lambda$ let $b_x = \mathcal{L}(t(\lambda), x)$. By the security of the message-hiding encoding scheme there exists a negligible function μ_2 such that:

$$\begin{aligned} &\Pr[\mathcal{B}_\lambda(x) = b_x : x \leftarrow \mathcal{X}_\lambda(1^\lambda, t(\lambda))] = \\ &\Pr\left[\begin{array}{l} x \leftarrow \mathcal{X}_\lambda(1^\lambda, t(\lambda)) \\ \widehat{s}_1 \leftarrow \mathsf{MHE.Encode}(P^t, x, t, s_1) \\ \widehat{s}_0 \leftarrow \mathsf{MHE.Encode}(1 - P^t, x, t, s_0) \\ b_x \leftarrow \mathcal{A}_\lambda(t, x, \widehat{s}_0, \widehat{s}_1) \end{array}\right] \geq \\ &\Pr\left[\begin{array}{l} x \leftarrow \mathcal{X}_\lambda(1^\lambda, t(\lambda)) \\ \widehat{s}_1 \leftarrow \mathsf{MHE.Encode}(P^t, x, t, s_{b_x}) \\ \widehat{s}_0 \leftarrow \mathsf{MHE.Encode}(1 - P^t, x, t, s_{b_x}) \\ b_x \leftarrow \mathcal{A}_\lambda(t, x, \widehat{s}_0, \widehat{s}_1) \end{array}\right] - \mu_2(\lambda) \quad . \end{aligned}$$

By (3) we have that for some polynomial p_3:

$$\begin{aligned} &\Pr\left[\begin{array}{l} x \leftarrow \mathcal{X}_\lambda(1^\lambda, t(\lambda)) \\ \widehat{s}_1 \leftarrow \mathsf{MHE.Encode}(P^t, x, t, s_{b_x}) \\ \widehat{s}_0 \leftarrow \mathsf{MHE.Encode}(1 - P^t, x, t, s_{b_x}) \\ b_x \leftarrow \mathcal{A}_\lambda(t, x, \widehat{s}_0, \widehat{s}_1) \end{array}\right] - \mu_2(\lambda) \geq \\ &\frac{1}{2} + \frac{1}{p_3(\lambda)} \quad . \end{aligned}$$

A.5 From Message-Hiding Encodings to Randomized Encodings via FHE

In this section, we sketch how succinct randomized encodings can be obtained from message-hiding encodings, assuming fully-homomorphic encryption (FHE). The transformation follows the same ideas introduced in [GKP+13] to convert attribute-based encryption to functional encryption.

The construction will rely on garbling schemes that are similar to the (standard) randomized encoding schemes given in Definition 3.9, only that the input x can be bit-wise encoded very fast and separately of the machine M. Like randomized encodings, such schemes can be constructed from one-way functions [Yao86] (and un particular from FHE).

The high-level idea is to encrypt the secret machine M and input x and provide a garbled decryption procedure $\widehat{M}_{\mathsf{SK}}$, where the the encoded input (i.e. the evaluated ciphertext corresponding to $M(x)$) will be given under message-hiding encodings that will guarantee that only the correct encoded input is revealed.

We now formally define garbling schemes and describe the construction in detail.

Definition A.5 (Garbling Scheme)**.** *A garbling scheme* GAR *consists of three algorithms* $(\mathsf{GAR.EncodeMach}, \mathsf{GAR.EncodeInp}, \mathsf{RE.Decode})$ *satisfying the following requirements.*

- *Syntax:*
  - $(\widehat{M}, K) \leftarrow \mathsf{GAR.EncodeMach}(M, n, t, 1^\lambda)$ *is a probabilistic algorithm that takes as input a machine M, input size n, time bound t, and a security parameter 1^λ. The algorithm outputs an encoding $\widehat{M}$ and secret key $K \in \{0,1\}^\lambda$.*
  - $\widehat{x} \leftarrow \mathsf{GAR.EncodeInp}(x_i, i, K)$ *is a probabilistic algorithm that takes an input bit x_i and index $i \in [n]$ key $K \in \{0,1\}^\lambda$. The algorithm outputs an encoding $\widehat{x}_i$.*
  - $y \leftarrow \mathsf{GAR.Decode}(\widehat{M}, \widehat{x})$ *is a deterministic algorithm that takes as input encodings $\widehat{M}$ and $\widehat{x} = (\widehat{x}_1, \ldots, \widehat{x}_n)$ and computes an output $y \in \{0,1\}^\lambda$.*
- *Functionality: for every input x and machine M such that, on input x, M halts in t steps and produces a λ-bit output, it holds that $y = M(x)$ with overwhelming probability over the coins of* GAR.EncodeMach, GAR.EncodeInp.
- *Security: there exists a PPT simulator* Sim *satisfying: for any poly-size distinguisher $\mathcal{D} = \{\mathcal{D}_\lambda\}_{\lambda \in \mathbb{N}}$ and polynomials $m(\cdot), n(\cdot), t(\cdot)$, there exists a negligible $\mu(\cdot)$, such that for any $\lambda \in \mathbb{N}$, machine $M \in \{0,1\}^{m(\lambda)}$, input $x \in \{0,1\}^{n(\lambda)}$:*

$$\left| \Pr\left[\mathcal{D}_\lambda(\widehat{M}, \widehat{x}) = 1 : \begin{array}{c} (\widehat{M}, K) \leftarrow \mathsf{GAR.EncodeMach}(M, n(\lambda), t(\lambda), 1^\lambda) \\ \widehat{x} \leftarrow \{\mathsf{GAR.EncodeInp}(x_i, i, K) : i \in [n(\lambda)]\} \end{array} \right] - \Pr[\mathcal{D}_\lambda(\widehat{S}, \widehat{s}) = 1 : (\widehat{S}, \widehat{s}) \leftarrow \mathsf{Sim}(y, 1^{m(\lambda)}, 1^{n(\lambda)}, t(\lambda), 1^\lambda)] \right| \leq \mu(\lambda) \ ,$$

  where y is the output of $M(x)$ after $t(\lambda)$ steps.
- *Efficiency: For any machine M that on input $x \in \{0,1\}^n$ produces a λ-bit output in t steps:*
  - $\mathsf{GAR.EncodeMach}(M, n, t, 1^\lambda)$ *can be computed in time $t \cdot \mathrm{poly}(|M|, n, \lambda)$.*
  - $\mathsf{GAR.EncodeInp}(x_i, i, K)$ *can be computed in time $\mathrm{poly}(\log n, \lambda)$.*
  - $\mathsf{GAR.Decode}(\widehat{M}, \widehat{x})$ *can be computed in time $t \cdot \mathrm{poly}(|M|, n, \lambda)$.*

Construction A.6 (Succinct Randomized Encodings from Message-Hiding Encodings). Let $\mathsf{GAR} = (\mathsf{GAR.EncodeMach}, \mathsf{GAR.EncodeInp}, \mathsf{GAR.Decode})$ be a garbling scheme. Let $\mathsf{FHE} = (\mathsf{FHE.Enc}, \mathsf{FHE.Dec}, \mathsf{FHE.Eval})$ be a fully-homomorphic encryption scheme.

For a ciphertext CT, interpreted as an encryption of machine M and input x, denote by $P^{t,b}_{\mathsf{CT},\mathsf{PK},i}$ an inputless machine that homomorphically evaluates M on x, obtains and evaluated cipher $\widehat{\mathsf{CT}}$ (interpreted as an encryption of $M(x)$), and accepts iff the i-th bit $\widehat{\mathsf{CT}}_i$ is b.

Also, for decryption key SK, denote by M_{SK} the machine that takes as input an evaluated ciphertext $\widehat{\mathsf{CT}}$ and decrypts it. We assume that the size of the underlying plaintext in $\widehat{\mathsf{CT}}$ is λ and that $\widehat{\mathsf{CT}}$ is of polynomial size $\ell(\lambda)$. We denote by $t_{\mathsf{Dec}}(\lambda)$ the running time of M_{SK}.

The succinct randomized encodings is constructed as follows:

- $\widehat{M}(x) \leftarrow \mathsf{RE.Encode}(M, x, t, 1^\lambda)$:
  - Sample $(\mathsf{PK}, \mathsf{SK}) \leftarrow \mathsf{FHE.Gen}(1^\lambda)$.
  - Garble decryption $(\widehat{M}_{\mathsf{SK}}, K) \leftarrow \mathsf{GAR.EncodeMach}(M_{\mathsf{SK}}, \ell(\lambda), t_{\mathsf{Dec}}, 1^\lambda)$.
  - Compute all possible garbled inputs $\left\{m^b_i := \widehat{x}^b_i \leftarrow \mathsf{GAR.EncodeInp}(b, i, K)\right\}_{i\in[\ell(\lambda)],b\in\{0,1\}}$.
  - Encrypt the computation $\mathsf{CT} \leftarrow \mathsf{FHE.Enc}(\mathsf{PK}, (M, x))$.
  - Compute message-hiding encodings $\left\{\widehat{m}^b_i \leftarrow \mathsf{MHE.Encode}(P^{t,b}_{\mathsf{CT},\mathsf{PK},i}, \bot, \widehat{x}^b_i)\right\}_{i\in[\ell(\lambda)],b\in\{0,1\}}$.
  - Output $\widehat{M}(x) := \widehat{M}_{\mathsf{SK}}, \{\widehat{m}^b_i\}$.
- $y \leftarrow \mathsf{RE.Decode}(\widehat{M}(x))$:
  - Obtain $\widehat{x}^{\widehat{\mathsf{CT}}} := \left\{m_i^{\widehat{\mathsf{CT}}_i} = \widehat{x}_i^{\widehat{\mathsf{CT}}_i} = \mathsf{MHE.Decode}(\widehat{m}_i^{\widehat{\mathsf{CT}}_i})\right\}_{i\in[\ell(\lambda)]}$.
  - Output $y := \mathsf{GAR.Decode}(\widehat{M}_{\mathsf{SK}}, \widehat{x}^{\widehat{\mathsf{CT}}})$.

Theorem A.7. *Construction A.6 is a succinct randomized encoding.*

The formal proof of the theorem closely follows the proof of [GKP+13, Theorem 3.1], where message-hiding encodings take the role of attribute-based encryption and randomized encodings take the role of functional encryption, and is omitted. Here we only give the intuition.

In terms of correctness, note that predicates $P^{t,b}_{\mathsf{CT},\mathsf{PK},i}$ underlying the message-hiding encodings are only satisfied for the values $\widehat{\mathsf{CT}}_i$ that correspond to a correct homomorphic evaluation of M on x. In terms of efficiency, the encoding procedure consists of message-hiding encodings, which are guaranteed to be computable fast independently of t, and garbling decryption, which is again a computation that is independent of t. The security follows by a combination of three underlying primitives. By the security of message-hiding encoding only the encoding of the correct $\widehat{\mathsf{CT}}$ is revealed and nothing else. This, in turn implies, that the garbled decryption reveals no information about decryption key SK and can be completely simulated from the output $M(x)$. Thus, we can safely rely on the semantic security of the encryption to claim that no information is leaked on M, x except from what can be simulated from $M(x)$.

B Necessity of One-Way Functions

In this section, we show that time-lock puzzles imply one-way functions. As a corollary, we can deduce that succinct randomized encodings and non-parallelizing languages imply one-way functions. Extended these ideas, we show that succinct randomized encodings alone imply

one-way functions against uniform adversaries (without the additional assumption regarding non-parallelizing languages).

A weak one-way function from time-lock puzzles. As a first step, we construct weak one-way functions from time-lock puzzles. Then, one can obtain strong one-way functions using standard amplification [Yao82].

Let $(\mathsf{Puzzle.Gen}, \mathsf{Puzzle.Sol})$ be a time-lock puzzle with gap $\varepsilon < 1$ and assume that for security parameter $\lambda \in \mathbb{N}$, and any $s \in \{0,1\}^\lambda$, $t \leq 2^\lambda$, $\mathsf{Puzzle.Gen}(t, s; r)$ uses random coins r of length $\ell(\lambda) = \lambda^{O(1)}$. We define a function f that takes as input random coins r and a random parameter τ sampled from some small set, and outputs a puzzle with some fixed solution $s_0 \in \{0,1\}^\lambda$ that opens in time $t = 2^\tau$. Formally, $f : \{0,1\}^{\ell(\lambda)} \times [\log^2 \lambda] \to \{0,1\}^*$ is defined by:

$$f(r, \tau) = \mathsf{Puzzle.Gen}(2^\tau, s_0; r) \ .$$

Claim B.1. *f is $\Omega\left(\frac{1}{\log^2 \lambda}\right)$-one-way.*

Proof sketch. Fix any poly-size $\mathcal{A} = \{\mathcal{A}_\lambda\}$ and let $d(\lambda)$ be a polynomial bound on $\mathcal{A}$'s depth. We essentially show that when $2^{\varepsilon\tau} > d(\lambda)$, $\mathcal{A}$ fails to distinguish the output of the one-way function $f(r, \tau)$ from a random puzzle with some other fixed solution $s_1 \neq s_0$, which has no preimage under f (by the completeness of the puzzle). Since $2^{\varepsilon\tau} > d(\lambda)$ with probability at least $\frac{1}{\log^2 \lambda}$, the result will follow.

Formally, recall that by the definition of time-lock puzzles there exists a polynomial $\underline{t}(\cdot)$, such that for every polynomial $t(\lambda) \geq \underline{t}(\lambda)$ such that $d(\lambda) \leq t^\varepsilon(\lambda)$:

$$\begin{aligned} \big| \Pr\left[\mathcal{A}_\lambda(Z) \in f^{-1}(Z) \mid Z \leftarrow \mathsf{Puzzle.Gen}(t(\lambda), s_0)\right] - \\ \Pr\left[\mathcal{A}_\lambda(Z) \in f^{-1}(Z) \mid Z \leftarrow \mathsf{Puzzle.Gen}(t(\lambda), s_1)\right] \big| \leq \mu(\lambda) \ , \end{aligned}$$

for some negligible μ and any $s_0, s_1 \in \{0,1\}^\lambda$.

We consider any polynomial $t^*(\lambda) = 2^{\tau^*(\lambda)}$ that satisfies both conditions in the above definition. Then

$$\begin{aligned} &\Pr\left[\mathcal{A}_\lambda(f(r,\tau)) \notin f^{-1}(f(r,\tau)) \mid (r,\tau) \leftarrow \{0,1\}^{\ell(\lambda)} \times [\log^2(\lambda)]\right] = \\ &\Pr\left[\mathcal{A}_\lambda(Z) \notin f^{-1}(Z) \mid \begin{array}{c}(r,\tau) \leftarrow \{0,1\}^{\ell(\lambda)} \times [\log^2(\lambda)] \\ Z = \mathsf{Puzzle.Gen}(2^\tau, s_0; r)\end{array}\right] \geq \\ &\frac{1}{\log^2 \lambda} \Pr\left[\mathcal{A}_\lambda(Z) \notin f^{-1}(Z) \mid \begin{array}{c} r \leftarrow \{0,1\}^{\ell(\lambda)} \\ Z = \mathsf{Puzzle.Gen}(t^*(\lambda), s_0; r)\end{array}\right] \geq \\ &\frac{1}{\log^2 \lambda}\left(\Pr\left[\mathcal{A}_\lambda(Z) \notin f^{-1}(Z) \mid \begin{array}{c} r \leftarrow \{0,1\}^{\ell(\lambda)} \\ Z = \mathsf{Puzzle.Gen}(t^*(\lambda), s_1; r)\end{array}\right] - \mu(\lambda)\right) \geq \\ &\frac{1}{\log^2 \lambda} - \lambda^{-\omega(1)} \ . \end{aligned}$$

□

B.1 Reducing Complexity Assumptions: One-Way Functions from Relaxed Time-Lock Puzzles

Note that, in the above proof, we did not really invoke the full power of time-lock puzzles. Concretely, time-lock puzzles that are secure against adversaries of a certain bounded *size* rather than bounded *depth* would have sufficed. For such time-lock puzzles non-parallelizing languages are not needed. Instead we can use languages that are decidable in uniform polynomial-time

t, but cannot be decided by circuits of significantly smaller size $s < t^{\varepsilon}$ (in the spirit of the assumptions made in the context of derandomizing BPP [IW01]).

We observe that if we settle for one-way functions against *uniform* adversaries, we can rely only on succinct randomized encodings and remove additional complexity assumptions altogether. The intuition behind this observation is that for the class of uniform polynomial time languages we have an unconditional *time hierarchy theorem* [HS65]. This intuition cannot be fulfilled as is since our reduction from breaking time-lock puzzles to violating the hierarchy theorem is a randomized, and a time hierarchy theorem is not known for BPP. However, the intuition can be salvaged using a hierarchy theorem for *slightly non-uniform BPP* [Bar02]. We elaborate below.

We first define the required relaxation of time-lock puzzles.

Definition B.2 (Relaxed Time-Lock Puzzles)**.** *A puzzle* $(\mathsf{Puzzle.Gen}, \mathsf{Puzzle.Sol})$ *is a relaxed time-lock puzzle with gap* $\varepsilon < 1$ *if there exists a polynomial* $\underline{t}(\cdot)$*, such that for every polynomial* $t(\cdot) \geq \underline{t}(\cdot)$ *and every probabilistic poly-time (uniform) adversary* $\mathcal{A}$ *with running time* $\mathsf{time}_{\mathcal{A}}(\lambda) \leq t^{\varepsilon}(\lambda)$*, there exists a negligible function* μ*, such that for every* $\lambda \in \mathbb{N}$*, and every pair of (uniformly computable) solutions* $s_0, s_1 \in \{0,1\}^{\lambda}$*:*

$$\Pr\left[b \leftarrow \mathcal{A}(Z) : \begin{array}{l} b \leftarrow \{0,1\} \ , \\ Z \leftarrow \mathsf{Puzzle.Gen}(t(\lambda), s_b) \end{array}\right] \leq \frac{1}{2} + \mu(\lambda) \ ,$$

where the probability is also over the random coin tosses of $\mathcal{A}$*.*

Claim B.3. *If* $(\mathsf{Puzzle.Gen}, \mathsf{Puzzle.Sol})$ *is a relaxed time-lock puzzle, then* f *defined above is* $\Omega\left(\frac{1}{\log^2 \lambda}\right)$*-one-way against uniform inverters.*

The proof of the claim is essentially identical to the proof of Claim B.1, except that instead of considering poly-size circuit inverters and their depth, we consider uniform inverters and their running time.

B.1.1 Constructing Relaxed Time-Lock Puzzles

The construction of relaxed time-lock puzzles is, in fact, identical to Construction 3.5 of (standard) time-lock puzzles. The existence of non-parallelizing languages is replaced by the following unconditional theorem by Barak.

Theorem B.4 ([Bar02])**.** *For any constant* $\varepsilon < 1$ *and any (uniformly computable) polynomially-bounded function* $t(\cdot)$*, there exists a language* $\mathcal{L} \in \mathit{Ptime}(t(\cdot))/\log\log(\cdot)$ *such that every probabilistic polynomial time* $\mathcal{B}$ *with running time* $\mathsf{time}_{\mathcal{B}}(\lambda) \leq t^{\varepsilon}(\lambda)$*, and non-uniform advice of size* $\log\log(\lambda)$*, and every large enough* λ*,* $\mathcal{B}$ *fails to decide* $\mathcal{L}_{\lambda} = \mathcal{L} \cap \{0,1\}^{\lambda}$*.*

Above $\mathrm{Ptime}(t(\cdot))/\log\log(\cdot)$ is the set of languages decidable by a BPP machine with non-uniform advice of size $\log\log(\lambda)$ in the input size λ. We shall assume w.l.o.g that the error of all BPP machines is bounded by $2^{-\lambda}$.

Theorem B.5. *For any* $\underline{\varepsilon} < 1$*, Construction 3.5 is a relaxed time-lock puzzle with gap* $\underline{\varepsilon}$*.*

The proof of the theorem is an adaptation of the proof of Theorem 3.7. At high-level, the only difference is that now when proving security, rather than constructing a circuit decider for a non-parallelizing language we construct a decider that only has slight non-uniform advice for a language given by Theorem B.4. For the sake of completeness, we give the details below.

B.1.2 Proof of Theorem B.5

The completeness and efficiency properties of the puzzle follow directly from the completeness and efficiency properties of the succinct randomized encoding scheme. We proceed to argue that the puzzle is a secure relaxed time-lock puzzle. Let $q_{\mathsf{RE}}(\lambda)$ be the fixed polynomial given by the efficiency property of the succinct randomized encoding scheme, bounding the time required to compute a randomized encoding with machine size 3λ, input size λ, and any time bound $t \leq 2^{\lambda}$. Let $\underline{t}(\lambda) := (q_{\mathsf{RE}}(\lambda))^{1/\underline{\varepsilon}}$.

Assume towards contradiction that there exists a uniform adversary $\mathcal{A}$, and a polynomially bounded function $t(\cdot) \geq \underline{t}(\cdot)$ such that $\mathsf{time}_{\mathcal{A}}(\lambda) < t^{\underline{\varepsilon}}(\lambda)$ and for some polynomial $p(\cdot)$ and infinitely many $\lambda \in \mathbb{N}$ there exists a pair of solutions $s_0, s_1 \in \{0,1\}^{\lambda}$ (computable in uniform polytime) such that:

$$\Pr\left[b \leftarrow \mathcal{A}(Z) : \begin{array}{l} b \leftarrow \{0,1\} \ , \\ Z \leftarrow \mathsf{Puzzle.Gen}(t(\lambda), s_b) \end{array}\right] \geq \frac{1}{2} + \frac{1}{p(\lambda)} \ . \tag{5}$$

Fix any $\underline{\varepsilon} < \varepsilon < 1$, and let $\mathcal{L} \in \mathrm{Ptime}(t(\cdot))/\log\log(\cdot)$ be the language with gap ε given by Theorem B.4. We construct a probabilistic machine $\mathcal{B}$ with running time $\mathsf{time}_{\mathcal{B}}(\lambda) \leq t^{\varepsilon}(\lambda)$ with advice of size $\log\log(\lambda)$ that decides $\mathcal{L}_{\lambda} = \mathcal{L} \cap \{0,1\}^{\lambda}$ for any λ as above, contradicting Theorem B.4.

Fix any λ as above with corresponding $s_0, s_1 \in \{0,1\}^{\lambda}$. Let $M^{\mathcal{L},t}_{s_0,s_1}$ be a probabilistic machine that, on input $x \in \{0,1\}^{\lambda}$, outputs s_1 if $x \in \mathcal{L}$ and and s_0 if $x \notin \mathcal{L}$, after exactly $t(\lambda)$ steps. Such a machine, with error $2^{-\lambda}$, can be uniformly constructed given $\log\log(\lambda)$ bits of non-uniform advice since $\mathcal{L} \in \mathrm{Ptime}(t(\cdot))/\log\log(\cdot)$ (and s_0, s_1 can be generated in uniform polytime). Further, denote by $M^{\mathcal{L},t,r}_{s_0,s_1}$ this machine with fixed random coins r, and assume that each such machine is described by 3λ bits (which is possible for large enough λ), and thus has the same description length as $M^{t}_{s_b}$.

Given input $x \in \{0,1\}^{\lambda}$, to decide if $x \in \mathcal{L}$, the $\mathcal{B}_{\lambda}$ acts as follows:

- Sample $Z := \widehat{M}^{\mathcal{L},t,r}_{s_0,s_1}(x) \leftarrow \mathsf{RE.Encode}(M^{\mathcal{L},t,r}_{s_0,s_1}, x, t(\lambda), 1^{\lambda})$, where r are uniformly random coins.
- Obtain $b \leftarrow \mathcal{A}_{\lambda}(Z)$ and output b.

First, note that $\mathcal{B}$ can be implemented with $\log\log(\lambda)$ bits of non-uniform advice and runs in polynomial time:

$$\mathsf{time}(\mathcal{B}_{\lambda}) = q_{\mathsf{RE}}(\lambda) + \mathsf{time}_{\mathcal{A}}(\lambda) = \underline{t}^{\underline{\varepsilon}}(\lambda) + t^{\underline{\varepsilon}}(\lambda) \leq 2t^{\underline{\varepsilon}}(\lambda) = o(t^{\varepsilon}(\lambda)) \ .$$

We next show that $\mathcal{B}$ distinguishes instances $x \in \mathcal{L}$ from instances $x \notin \mathcal{L}$ with noticeable advantage. For any $x \in \{0,1\}^{\lambda}$, let $b \in \{0,1\}$ indicate whether $x \in \mathcal{L}_{\lambda}$ we have that $s_b = M^{\mathcal{L},t}_{s_0,s_1}(x) = M^{t}_{s_b}(0^{\lambda})$. Therefore, by the security of the randomized encoding scheme there exists a PPT simulator Sim and a negligible function $\mu(\cdot)$ such that for any $x \in \{0,1\}^{\lambda}$:

$$\begin{aligned} &\Pr[\mathcal{B}_{\lambda}(x) = 1] = \\ &\Pr\left[\mathcal{A}_{\lambda}(\widehat{M}^{\mathcal{L},t,r}_{s_0,s_1}(x)) = 1 : \begin{array}{c} r \leftarrow \{0,1\}^{\mathrm{poly}(\lambda)} \\ \widehat{M}^{\mathcal{L},t}_{s_0,s_1}(x) \leftarrow \mathsf{RE.Encode}(M^{\mathcal{L},t}_{s_0,s_1}, x, t(\lambda), 1^{\lambda}) \end{array}\right] = \\ &\Pr\left[\mathcal{A}_{\lambda}(\widehat{S}_{s_b}) = 1 : \widehat{S}_{s_b} \leftarrow \mathsf{Sim}\left(s_b, 1^{3\lambda}, 1^{\lambda}, t(\lambda), 1^{\lambda}\right)\right] \pm (\mu(\lambda) + 2^{-\lambda}) \ , \end{aligned}$$

and:

$$\begin{aligned}
&\Pr\left[\mathcal{A}_\lambda(Z) = 1 : Z \leftarrow \mathsf{Puzzle.Gen}(t(\lambda), s_b)\right] = \\
&\Pr\left[\mathcal{A}_\lambda(\widehat{M}^t_{s_b}(0^\lambda)) = 1 : \widehat{M}^t_{s_b}(0^\lambda) \leftarrow \mathsf{RE.Encode}(M^t_{s_b}, 0^\lambda, t(\lambda), 1^\lambda)\right] = \\
&\Pr\left[\mathcal{A}_\lambda(\widehat{S}_{s_b}) = 1 : \widehat{S}_{s_b} \leftarrow \mathsf{Sim}\left(s_b, 1^{3\lambda}, 1^\lambda, t(\lambda), 1^\lambda\right)\right] \pm \mu(\lambda) \ .
\end{aligned}$$

It follows by our assumption towards contradiction (Equation 5) that for large enough λ and any $x \in \mathcal{L}_\lambda$, $\bar{x} \notin \mathcal{L}_\lambda$:

$$|\Pr[\mathcal{B}_\lambda(x) = 1] - \Pr[\mathcal{B}_\lambda(\bar{x}) = 1]| \geq \frac{2}{p(\lambda)} - 2\mu(\lambda) \geq \frac{1}{p(\lambda)} \ .$$

This completes the proof.