Electronic Colloquium on Computational Complexity, Report No. 1 (2015)

On the Cryptographic Hardness of Finding a Nash Equilibrium

Nir Bitansky* Omer Paneth† Alon Rosen‡

January 1, 2015

Abstract

We prove that finding a Nash equilibrium of a game is hard, assuming the existence of indistinguishability obfuscation and injective one-way functions with sub-exponential hardness. We do so by showing how these cryptographic primitives give rise to a hard computational problem that lies in the complexity class PPAD, for which finding Nash equilibrium is known to be complete.

Previous proposals for basing PPAD-hardness on program obfuscation considered a strong "virtual black-box" notion that is subject to severe limitations and is unlikely to be realizable for the programs in question. In contrast, for indistinguishability obfuscation no such limitations are known, and recently, several candidate constructions of indistinguishability obfuscation were suggested based on different hardness assumptions on multilinear maps.

Our result provides further evidence of the intractability of finding a Nash equilibrium, one that is extrinsic to the evidence presented so far.

1 Introduction

The notion of *Nash equilibrium* is fundamental to game theory. While a mixed Nash equilibrium is guaranteed to exist in any game [Nas51], there is no known polynomial-time algorithm for finding one. The tractability of the problem has received much attention in the past decade, in large part due to its theoretical and philosophical significance. Prominent evidence for the hardness of finding a Nash equilibrium emerges from a line of works, originating with Papadimitriou [Pap94] and ultimately showing that the problem is complete for the complexity class PPAD [DGP09, CDT09]. The class PPAD contains several other search problems that are not known to be tractable, such as finding a fixed point of the kind guaranteed by Brouwer's Theorem. Akin to the phenomenon of NP-completeness, this could be interpreted as evidence to computational difficulty. However, unlike in the case of NP, currently known problems in PPAD appear to be of fairly restricted nature, and carry similar flavor to one another.

Seeking further evidence for the hardness of PPAD, we aim to to base its hardness on new types of problems. As observed in [MP91, Pap94] problems in PPAD cannot be NP-complete unless NP $=$ coNP. A potential source for such problems (already suggested in [Pap94]) is *cryptographic hardness*.

Towards identifying a suitable problem from the domain of cryptography, let us first take a closer look into the definition of PPAD. The class PPAD consists of all total search problems reducible to the following

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

‡Efi Arazi School of Computer Science, IDC Herzliya, Israel. Email: alon.rosen@idc.ac.il. Supported by ISF grant no. 1255/12 and by the ERC under the EU's Seventh Framework Programme (FP/2007-2013) ERC Grant Agreement n. 307952.

ISSN 1433-8092

END-OF-THE-LINE problem. We are given a program succinctly representing an exponential size directed graph over the nodes $\{0,1\}^n$, together with a source node x_s. The in-degree and out-degree of every node are at most one, and the in-degree of the source x_s is zero. Our goal is to find another node, other than x_s, with in-degree or out-degree zero. Such a node must exist by a simple parity argument.

Intuitively, solving the END-OF-THE-LINE problem may require some sort of ”reverse-engineering” of the program representing the graph. Indeed, under minimal cryptographic assumptions, solving the problem while only treating the program as a black-box is impossible. For instance, the program may internally invoke a pseudo-random permutation [LR88] to describe a “random looking” END-OF-THE-LINE graph that cannot be solved efficiently with only black-box access.[1]

This suggests a natural approach for constructing PPAD-hard problems based on *cryptographic obfuscation*, a compiler that transforms any program into an unintelligible one while preserving functionality. Ideally, an obfuscated program should be equivalent to a virtual black-box (VBB): it should reveal nothing more than what can be learned from its input-output behavior [BGI$^+$01]. In particular, a (pseudo) random END-OF-THE-LINE graph described by an obfuscated program should be unsolvable by any efficient algorithm *even given this obfuscated program* . Abbot, Kane and Valiant [AKV04] further show that PPAD-hardness can be based on VBB obfuscation of an even simpler program that essentially computes some natural pseudo random function.

Neither of the above programs, however, is known to be VBB obfuscatable. Indeed, VBB obfuscation is currently known only for a few simple functions based on strong assumptions. Moreover, certain functions are impossible to VBB obfuscate [BGI$^+$01], and VBB obfuscation of pseudo-random functions, including the one considered in [AKV04], is in particular subject to strong limitations [GK05, BCC$^+$14].

In light of the impossibilities for VBB obfuscation, Barak et al. [BGI$^+$01] defined *indistinguishability obfuscation* (iO), a relaxed notion requiring only that the obfuscations of any two equal-size circuits computing the same function are indistinguishable from one another. For iO, no impossibilities are known. Furthermore, starting with the work of Garg et al. several constructions of iO were recently suggested based on different assumptions related to cryptographic multilinear maps [GGH$^+$13, BR14, BGK$^+$14, PST14, GLSW14, Zim14]. A natural question is whether the comparatively weak security guarantees of iO suffice for establishing hardness of PPAD.

1.1 This Work

We show PPAD-hardness based on indistinguishability obfuscation and injective one-way functions with super-polynomial hardness.

Theorem 1.1 (Informal)**.** *Suppose that there exist sub-exponentially-hard injective one-way functions and quasi-polynomially-hard indistinguishability obfuscation for P/poly. Then, the* END-OF-THE-LINE *problem is hard for polynomial-time algorithms.*

In fact, under the above assumptions, we show that PPAD is *hard on average for quasi-polynomial algorithms*. Specifically, there exists an efficiently samplable distribution on END-OF-THE-LINE instances that fails probabilistic quasi-polynomial algorithms (for some quasi-polynomial function that depends on the iO hardness). We can get sub-exponential PPAD-hardness at the cost of assuming sub-exponential iO. We can also trade-off the security of the iO and the resulting PPAD-hardness with the security of the one-way function. Specifically, we show that assuming only polynomially-hard iO, but exponentially-hard injective one-way functions, PPAD is hard in the worst-case (rather than on average) for polynomial-time algorithms.

[1] If we do not require succinct representation of the graph, unconditional black-box hardness results are known [HPV89, Pap94].

One may also interpret our result in the converse: any algorithmic breakthrough resulting, say, in a sub-exponential algorithm for computing Nash equilibria will lead to a sub-exponential attack on the hardness of iO or injective one-way functions.

1.2 Main Ideas

To demonstrate the hardness of PPAD, we construct a hard distribution over END-OF-THE-LINE instances. Recall that an END-OF-THE-LINE instance is defined by a program representing an exponential size directed graph. Verifying that a given program indeed describes a valid END-OF-THE-LINE instance can be done efficiently (thus making the problem total [MP91, Pap94]).

In more detail, an END-OF-THE-LINE graph is described by a pair of circuits S and P. Given an input node $x \in \{0,1\}^n$, $\mathsf{S}(x)$ outputs a "candidate successor" and $\mathsf{P}(x)$ outputs a "candidate predecessor" of x. We say that there is an edge from x to x' in the graph if $\mathsf{S}(x) = x'$ and $\mathsf{P}(x') = x$ (this guarantees that the in-degree and out-degree of every node are at most one). The instance also defines a source node $x_s \in \{0,1\}^n$ and we require that $\mathsf{P}(x_s) = x_s \neq \mathsf{S}(x_s)$ (this guarantees that the in-degree of the source node x_s is zero). The solution is any node other than x_s with in-degree or out-degree zero.

A simplified construction. To convey the main ideas behind the hardness proof, we first describe a simplified construction of END-OF-THE-LINE instances that will only satisfy a weak form of hardness, and then extend it to obtain the sought-after result. In the constructed (distribution over) instances, the circuits S and P contain the description a function $\mathsf{PRF} : [T] \rightarrow \{0,1\}^m$ sampled from a family of pseudo-random functions, where T is of super-polynomial size.

Nodes x in the graph are of the form $(i, \sigma) \in [T] \times \{0,1\}^m$. We say that a node (i, σ) is *valid* if $\sigma = \mathsf{PRF}(i)$. The graph defined by S and P contains a single path passing though all valid nodes. Every invalid node will be connected to itself by a self-loop. The graph contains a single source node $x_s = (1, \mathsf{PRF}(1))$ with in-degree zero and a single sink node $(T, \mathsf{PRF}(T))$ with out-degree zero.

Given a node (i, σ), the circuit S computes the candidate successor as follows:

1. If the node is valid and $i < T$, S outputs the node $(i+1, \mathsf{PRF}(i+1))$ as the candidate successor.
2. If the node is invalid or if $i = T$, S outputs the node (i, σ) unchanged.

The function P is defined analogously in the reverse direction. The END-OF-THE-LINE instance is the triplet $(\tilde{\mathsf{S}}, \tilde{\mathsf{P}}, (1, \mathsf{PRF}(1)))$, where $\tilde{\mathsf{S}}$ and $\tilde{\mathsf{P}}$ are indistinguishability obfuscations of the circuits S and P respectively and $(1, \mathsf{PRF}(1))$ is the source node. Note that this instance has a unique solution $(T, \mathsf{PRF}(T))$.

Intuition. The path from the source $(1, \mathsf{PRF}(1))$ to the sink $(T, \mathsf{PRF}(T))$ should be thought of as an authenticated chain where a signature σ corresponding to some valid node (i, σ) cannot be obtained without first obtaining all previous signatures. It is not difficult to show that any efficient algorithm that only invokes the circuits S and P (and thus also the pseudo-random function PRF) as a black box cannot find the signature $\mathsf{PRF}(T)$, and thus cannot solve the instance. We would like to prove that the same hardness holds *even given the obfuscated circuits* $\tilde{\mathsf{S}}$ *and* $\tilde{\mathsf{P}}$. However, we first prove something weaker: *finding the sink* $(T, \mathsf{PRF}(T))$ *is hard given only the successor circuit* $\tilde{\mathsf{S}}$. We then extend the proof to show that the sink is hard to find given *both* $\tilde{\mathsf{S}}$ and $\tilde{\mathsf{P}}$, which requires modifying the construction.

To prove that finding the signature $\mathsf{PRF}(T)$ corresponding to the sink is hard, we show that the obfuscated $\tilde{\mathsf{S}}$ is computationally indistinguishable from a circuit that on input $(T, \mathsf{PRF}(T))$ returns some fixed value $\perp$, rather than $(T, \mathsf{PRF}(T))$ itself as $\tilde{\mathsf{S}}$ would. This implies that an efficient algorithm would not be able to obtain $\mathsf{PRF}(T)$ from either one of the circuits, or it could distinguish the two. We next go in more detail into how indistinguishability of these two circuits is shown.

For every $j \leq k \leq T$, we consider the circuit $\mathsf{S}_{j,k}$ that is identical to S, except that for every $i \in [j,k]$, $\mathsf{S}_{j,k}$ on the input $(i, \mathsf{PRF}(i))$ outputs the fixed value $\perp$. The argument proceeds in two steps. First, we show that for a *random* $u \in [T]$, an obfuscation $\tilde{\mathsf{S}}_{u,u}$ of $\mathsf{S}_{u,u}$ is computationally indistinguishable from $\tilde{\mathsf{S}}$. Intuitively this "splits" the authenticated chain into two parts. While given the source node it is possible to compute additional signatures in the first part of the chain $[1, u-1]$, we show that is it hard to find a signature for any i in the second part of the chain $[u, T]$. More concretely, in the second step, we prove that the obfuscated circuits $\tilde{\mathsf{S}}_{u,u}$ and $\tilde{\mathsf{S}}_{u,T}$ are computationally indistinguishable by a sequence of hybrids. For every $j \in [u, T]$, we show that the obfuscations $\tilde{\mathsf{S}}_{u,j}$ and $\tilde{\mathsf{S}}_{u,j+1}$ are computationally indistinguishable. In total, we have T hybrids; relying on injective one-way functions (which only come in during the proof) and iO with super-polynomial hardness (related to the size of T), we show that each two obfuscations are $T^{-\Theta(1)}$-indistinguishable. Overall we deduce that the obfuscations $\tilde{\mathsf{S}}$ and $\tilde{\mathsf{S}}_{u,T}$ are also computationally indistinguishable as required. To summarize, the hardness proof follows two steps:

1. **Split the chain into two parts:** For a random $u \in [T]$, prove that $\tilde{\mathsf{S}}$ and $\tilde{\mathsf{S}}_{u,T}$ are indistinguishable.

2. **Erase second part:** For every $u \leq j < T$, prove that $\tilde{\mathsf{S}}_{u,j}$ and $\tilde{\mathsf{S}}_{u,j+1}$ are $T^{-\Theta(1)}$-indistinguishable.

We next explain how two the steps described above are proven. For simplicity, we shall assume the existence of a length-doubling pseudo-random generator $\mathsf{PRG} : [T] \to [T^2]$ that is injective; in the body, we relax this assumption and rely only on injective one-way functions. The two steps rely the ideas of *hidden triggers and punctured programs* introduced by Sahai and Waters [SW14].

First step. To prove that $\tilde{\mathsf{S}}$ is indistinguishable from $\tilde{\mathsf{S}}_{u,u}$, we first note that $\tilde{\mathsf{S}}$ is indistinguishable from an obfuscation of a circuit $\mathsf{S}_v(i, \sigma)$ that has an extra "if statement":

1. if $\sigma = \mathsf{PRF}(i)$ and $\mathsf{PRG}(i) = v$, return $\perp$.

2. Otherwise return $(i+1, \mathsf{PRF}(i+1))$ (as S would);

here v is chosen at random from the range $[T^2]$ of PRG. Observe that, with overwhelming probability $1 - \frac{1}{T}$, v is not in the image of PRG, the condition in (1) is never met, and the alternative behavior is never triggered. Thus, S and S_v compute the same function and their obfuscations are indistinguishable. Next, relying on the pseudo-randomness guarantee of PRG, we can indistinguishably replace the uniformly random v with a pseudo-random value $\mathsf{PRG}(u)$. It is left to note that, because PRG is injective, S_v, with $v = \mathsf{PRG}(u)$, computes the exact same function as $\mathsf{S}_{u,u}$. Indeed, both compute the same function as S except on $(u, \mathsf{PRF}(u))$, where an alternative behavior is triggered and $\perp$ is returned.

Second step. To prove that $\tilde{\mathsf{S}}_{u,j}$ and $\tilde{\mathsf{S}}_{u,j+1}$ are indistinguishable, we require that PRF is *puncturable*. This means that for every input $i \in [T]$, we can sample a punctured $\mathsf{PRF}_{\{i\}}$ that agrees with PRF on all inputs $j \neq i$, but computationally hides any information on the value $\mathsf{PRF}(i)$; namely $\mathsf{PRF}(i)$ is pseudo-random, even given a program for evaluating the punctured $\mathsf{PRF}_{\{i\}}$. Such puncturable pseudo-random functions are known to exist based on any one-way function [BW13, KPTZ13, BGI14]. Now, we note that $\tilde{\mathsf{S}}_{u,j}$ and $\tilde{\mathsf{S}}_{u,j+1}$ differ only on input $(j+1, \mathsf{PRF}(j+1))$: while the first returns $(j+2, \mathsf{PRF}(j+2))$, the second returns $\perp$. In particular, the two circuits must hide $\mathsf{PRF}(j+1)$ to guarantee indistinguishability. What enables hiding the value $\mathsf{PRF}(j+1)$ is that both circuits never output $\mathsf{PRF}(j+1)$, but rather, on input $(j, \mathsf{PRF}(j))$ both return $\perp$. The value $\mathsf{PRF}(j+1)$ is only used to test if an input $(j+1, \sigma)$ is valid by comparing σ to $\mathsf{PRF}(j+1)$. Relying on puncturing, this comparison can be performed in "encrypted form" while hiding $\mathsf{PRF}(j+1)$.

Concretely, we first move from $\tilde{\mathsf{S}}_{u,j}$ to $\tilde{\mathsf{S}}^{(1)}_{u,j}$ that has a punctured $\mathsf{PRF}_{\{j+1\}}$. The circuit has $\sigma_{j+1} = \mathsf{PRF}(j+1)$ hardwired, and given $(j+1, \sigma)$ it directly compares σ to σ_{j+1}. The circuit computes the same function, and indistinguishability holds by iO. Then, relying on pseudo-randomness at the punctured point $j+1$, we move to $\tilde{\mathsf{S}}^{(2)}_{u,j}$ where σ_{j+1} is replaced with a truly random value in $\{0,1\}^m$. Next, we move to $\tilde{\mathsf{S}}^{(3)}_{u,j}$ where the comparison of σ_{j+1} and σ is not done in the clear, but rather under an injective (length-doubling) pseudo-random generator $\mathsf{PRG} : \{0,1\}^m \to \{0,1\}^{2m}$; in particular, σ_{j+1} is not stored in the clear, but rather $\mathsf{PRG}(\sigma_{j+1})$ is stored. This does not change functionality and thus indistinguishability is again preserved. Now, using pseudo-randomness of PRG, we move to yet another circuit $\tilde{\mathsf{S}}^{(4)}_{u,j}$, where $\mathsf{PRG}(\sigma_{j+1})$ is replaced by a truly random string $v \in \{0,1\}^{2m}$. Finally, note that as in the proof of the first step, v is not in the image of PRG with overwhelming probability $1-2^{-m}$. Thus we can indistinguishably change the circuit to return $\perp$ when given the input $(j+1, \sigma_{j+1})$. We can then reverse the above steps and go back to computing $\sigma_{j+1} = \mathsf{PRF}(j+1)$, using the non-punctured PRF.

To guarantee the required indistinguishability gap between the different hybrids, we should take care in choosing the parameters T, the output length m of PRF, and the hardness of each of the cryptographic primitives. Choosing these parameters yields different hardness tradeoffs (further discussed in Section 5.5).

The full construction. So far we only proved that that finding $\mathsf{PRF}(T)$ is hard given $\tilde{\mathsf{S}}$. Proving that the same hardness holds given also $\tilde{\mathsf{P}}$ encounters a barrier. Indeed, to be able to gradually erase the output of $\tilde{\mathsf{S}}$ on valid inputs $(i, \mathsf{PRF}(i))$, we crucially relied on the fact that in the hybrid circuit $\tilde{\mathsf{S}}_{u,j}$, the value $\mathsf{PRF}(j+1)$ is never returned and is *only used in comparison*, which could be done in encrypted form. However, in the presence of $\tilde{\mathsf{P}}$ this is no longer the case since the signature $\mathsf{PRF}(j+1)$ can also be reached "from the other direction"; namely, on input $(j+2, \mathsf{PRF}(j+2))$, $\tilde{\mathsf{P}}$ does return $\mathsf{PRF}(j+1)$ in the clear.

To overcome this barrier, we modify the construction. We transform the original END-OF-THE-LINE instance $(\tilde{\mathsf{S}}, \tilde{\mathsf{P}}, x_s)$ into a new instance $(\mathsf{S}', \mathsf{P}', x'_s)$. Similarly to the original instance, the graph defined by the new instance contains a single path (every node outside this path is a self-loop). Intuitively, a walk on the path in the new graph "emulates" a walk on the path in the original graph. In particular, the new source x'_s can be computed from the original source x_s, and the new sink contains the original sink $(T, \mathsf{PRF}(T))$. The key property of the new construction is that both the new successor circuit S' and predecessor circuit P' can be efficiently constructed based *only on the original successor circuit* $\tilde{\mathsf{S}}$. Thus, the already proven hardness of obtaining $\mathsf{PRF}(T)$ from $\tilde{\mathsf{S}}$ carries over when given both the circuits S' and P', and therefore the resulting instance distribution has the required hardness.

Abbot, Kane and Valiant [AKV04] describe a construction of suitable circuits S' and P' given any circuit for $\tilde{\mathsf{S}}$ satisfying a certain *verifiability* property: for every $i < T$, there should be an efficient way to test if a given node is the i-th node on the path defined by $\tilde{\mathsf{S}}$. In our construction of $\tilde{\mathsf{S}}$ this can done by testing that the node is of the form (i, σ) and that $\mathsf{S}(i, \sigma)$ is of the form $(i+1, \sigma')$. The idea behind the construction is to rely on *reversible computation*, which allows to simulate any computation in a reversible way [Ben89]. [2]

In the body, we formulate the SINK-OF-VERIFIABLE-LINE problem, which captures the required properties for the Abbot et al. construction. For completeness, we also describe in detail the reduction to the END-OF-THE-LINE problem, based on reversible computation.

1.3 Related Work

In addition to the already mentioned works on PPAD and the PPAD-completeness of finding Nash equilibria, various works show hardness results for different variants on the problem of finding Nash equilibria, and explore other notions of equilibria with related hardness. We refer the reader to [Gol11] and to the related work sections of [OPR14, Rub14] for further details.

[2] Reversible computation was also used in [Pap94] to prove that a variant of PPAD contains PSPACE.

1.4 Organization

In Section 2, we define the complexity class PPAD (via the END-OF-THE-LINE PROBLEM), and the SINK-OF-VERIFIABLE-LINE problem. In Section 3, we describe the reduction between the problems based on reversible computation. In Section 5.1, we construct and analyze the hard distribution on SINK-OF-VERIFIABLE-LINE instances, based on iO and injective one-way functions.

2 PPAD and the Sink-of-Verifiable-Line Problem

A search problem (I, R) is defined by a set of instances $I \subseteq \{0,1\}^*$ and by an NP relation R. A search problem is *total* if testing membership in I is efficient and for every every $z \in I$ the set of witnesses R is non-empty. We say that a search problem (I, R) is reducible to a search problem (I', R') if there exists a pair of polynomial-time computable functions f, g such that for every $z \in I$, $f(z) \in I'$, and for every $w \in R'(f(z))$, $g(w) \in R(z)$.

2.1 PPAD

The class PPAD [Pap94] contains all total search problems that are reducible to the END-OF-THE-LINE problem (EOL).

Definition 2.1 (END-OF-THE-LINE)**.** *An instance $(x_s, \mathsf{S}, \mathsf{P})$ is defined by a starting node $x_s \in \{0,1\}^n$ and a pair of circuits S, P with inputs and outputs in $\{0,1\}^n$ such that $\mathsf{P}(x_s) = x_s \neq \mathsf{S}(x_s)$. A string $w \in \{0,1\}^n$ is a valid witness iff:*

$$\mathsf{P}(\mathsf{S}(w)) \neq w \quad \vee \quad \mathsf{S}(\mathsf{P}(w)) \neq w \neq x_s \ .$$

Intuitively, the circuits S, P define a directed graph over vertices $\{0,1\}^n$ where the in-degree and out-degree of every node is at most one. For a pair of nodes x, y there is an edge from x to y iff $\mathsf{S}(x) = y$ and $\mathsf{P}(y) = x$. The condition $\mathsf{P}(x_s) = x_s \neq \mathsf{S}(x_s)$ grantees that the starting node x_s has in-degree zero. A witness w is any other node with in-degree or out-degree zero. Such node must exists by a simple parity argument.

2.2 The Sink-of-Verifiable-Line Problem

The SINK-OF-VERIFIABLE-LINE problem (SVL) is a search problem described in [AKV04] (there it is not given any specific name).

Definition 2.2 (SINK-OF-VERIFIABLE-LINE)**.** *An instance $(\mathsf{S}, \mathsf{V}, x_s, T)$ consists of a source $x_s \in \{0,1\}^n$, a target index $T \in [2^n]$, and a pair of circuits $\mathsf{S} : \{0,1\}^n \to \{0,1\}^n$, $\mathsf{V} : \{0,1\}^n \times [T] \to \{0,1\}$, with the guarantee that, for $(x, i) \in \{0,1\}^n \times [T]$, it holds that $\mathsf{V}(x, i) = 1$ iff $x = x_i := \mathsf{S}^{i-1}(x_s)$, where $x_1 := x_s$. A string $w \in \{0,1\}^n$ is a valid witness iff $\mathsf{V}(w, T) = 1$.*

Intuitively, S defines a path over a subset of vertices in $\{0,1\}^n$ starting at x_s, progressing according to $x_i = \mathsf{S}(x_{i-1})$, and ending at the target $x_T = \mathsf{S}^{T-1}(x_s)$. Unlike in the EOL problem, an SVL instance defines a single directed path, and there is no algorithm describing backward edges on this path. Furthermore, it is possible to test whether a given node x is the i-th node on the path. Note that while every valid SVL instance has a single witness, the problem may not be total, since we do not know how to efficiently test if an instance $(\mathsf{S}, \mathsf{V}, x_s, T)$ is valid. Specifically it is hard to verify that $\mathsf{V}(x, i) = 1$ iff $x = x_i$. (Note that for the very same reason, the decision problem corresponding to SVL may not be in NP).

3 Reducing Sink-of-Verifiable-Line to End-of-the-Line

In this section, we give a reduction from the SVL problem to the EOL problem. This reduction was sketched in [AKV04], for completeness we describe the reduction in full. The reduction is an efficient mapping from any SVL instance to an EOL instance such that from any witness to the EOL problem, the witness to the SVL problem can be efficiently computed. Note that the SVL problem is not total and therefore (although reducible to EOL) it is not in PPAD. However, by our reduction, the existence of a hard distribution on SVL instances implies the existence of a hard distribution on EOL instances.

As the main step of mapping an SVL instance to an EOL instance, the reduction has to come up with an efficient implementation of the predecessor function. This relies on ideas from *reversible computation* [Ben89]. Following [Ben89, AKV04], we start by describing a simple *pebble game* capturing the main ideas used in the reduction.

3.1 The pebble game

Consider the following pebble game, also known as *the east model* [JE91, CDG01]. We are given a set of t pebbles meant to be placed in a set of positions represented by positive integers. In a *valid* move, a pebble can either be placed or removed from position i provided that either $i = 1$ or position $i - 1$ is occupied by a pebble. In particular, valid moves are reversible, and any sequence of valid moves can be reversed by another sequence of valid moves. The goal of the game is to place a pebble in position $2^t - 1$.

An efficient strategy can be described recursively as follows. Assume there is a sequence of moves, using only the first $t - 1$ pebbles, and resulting in a pebble placed in position $2^{t-1} - 1$. Now, place the t-th pebble in position 2^{t-1}. Next, free the first $t - 1$ pebbles by reversing the original sequence of moves. Once the first $t - 1$ pebbles are free, repeat the original sequence of moves shifting every position up by 2^{t-1} resulting in a pebble placed in position $2^t - 1$.

The above strategy essentially allows to simulate any computation in a reversible way, while incurring only polynomial blowup in running time. The reduction from SVL to EOL relies on the above idea. We next describe it in detail.

3.2 The reduction

Let $(\mathsf{S}, \mathsf{V}, x_s, T)$ be an SVL instance where $x_s \in \{0,1\}^n$ and let $t = \log(T)$ (assume without loss of generality that t is an integer). We construct an instance $(x_s, \mathsf{S}', \mathsf{P}')$ for the EOL problem where $x_s \in \{0,1\}^m$ and $m = t \cdot (n + t)$. We interpret every node in $\{0,1\}^m$ as a sequence $(u_1, \ldots, u_t)$ of t *states* where, for every $j \in [t]$, the state u_j is of the form $(x, i) \in \{0,1\}^n \times [T]$. We say that a state (x, i) is *valid* if $\mathsf{V}(x, i) = 1$ and denote the i-th valid state $(\mathsf{S}^{i-1}(x_s), i)$ by $v(i)$. Given $u = (x, i)$, we abuse notation (overloading the function S) and denote $\mathsf{S}(u) := (\mathsf{S}(x), i + 1)$. Given $u = (x, i)$ and $u' = (x', i')$, we say that $u < u'$ if $i < i'$.

Intuitively, the EOL instance will correspond to a graph with a single path. Every node $(u_1, \ldots, u_t) \in \{0,1\}^m$ on this path describes a configuration of the pebble game in the efficient strategy above. To express that the j-th pebble is in position i we set $u_j = v(i + 1)$. If the j-th pebble is not used we set $u_j = v(1)$ and say that the sate u_j is *free*. The starting node x_s describes the starting configuration of the pebbling game where all states are free. The last node on the path describes the final configuration and contains the state $v(T)$. The circuits S' and P' traverse the path following moves of the winning strategy. To place a the j-th pebble in position i, given that the k-th pebble in position $i - 1$ we set $u_j \leftarrow \mathsf{S}(u_k)$. The j-th pebble is removed to free the state u_j by setting $u_j = v(1)$. Given any node that does not describe a configuration reached by the strategy, the circuits S' and P' output the node unchanged indicating that the node is a self-loop. The the only witness for the the instance $(x_s, \mathsf{S}', \mathsf{P}')$ is the final node on the path, and it is possible to compute x_T from this node. (We note that if we apply the reduction to a tuple $(\mathsf{S}, \mathsf{V}, x_s, T)$ that is not a valid SVL instance, we will still get an EOL instance with one or many solutions, but there is no guarantee on their relation to the tuple $(\mathsf{S}, \mathsf{V}, x_s, T)$ we started from.)

We continue with a formal description of the reduction. For $j \in [t]$, we define the functions $(\mathsf{S}_j, \mathsf{P}_j)$ that traverse some segment of the entire path. The nodes in this segment differ only on the first j states, whereas the last $t-j$ states of all nodes in the segment are the same. In the first node of the segment, the first j states are all free. In the last node of the segment the value of the first j states will depend on the *base state* of the segment. The base state of the segment u_b can be any one of the states $\{u_{j+1}, \ldots, u_t\}$ (that are fixed along the segment) or the free state $v(1)$. If $u_b = v(i)$ for some $i \in [T]$, then the last node in the segment is such that for every $k \in [j]$, $u_k = v(i + 2^j - 2^{k-1})$.

The function S_j takes as input the base state u_b and the current node N and outputs the next node in the segment. If N is not in the segment, or if it is the last node in the segment, the function S_j outputs N unchanged. The function P_j behaves analogously. The functions S' and P' that traverse the entire path are simply the functions S_t and P_t executed with the free base state $v(1)$.

Next, we describe the implementation of the function S_j in more detail, but still at high-level; the function P_j is analogous to S_j *only in reverse*. Pseudo-code for the functions S_j and P_j is given in Section A in the appendix.

The function S_j: the function S_j takes as input a base state $u_b = v(i)$ and a node $N = (u_1, \ldots, u_t)$. We start by describing the behavior of S_j when N is indeed in the segment traversed by S_j. Later we explain how nodes that are not in the segment are recognized. The function S_j traverses the segment using the functions S_{j-1} and P_{j-1} following this strategy:

1. In the first node of the segment, the states $u_1, \ldots, u_j$ are all free.

2. The first part of the segment is traversed using the function S_{j-1} with the base node u_b. When the function S_{j-1} reaches the end of its segment we have that for every $k \in [j-1]$:

$$u_k = v(i + 2^{j-1} - 2^{k-1}) \ ;$$

   in particular, $u_1 = v(i + 2^{j-1} - 1)$.

3. The next node on the path is obtained by setting the free state u_j (this state is not changed by S_{j-1}) to the state $\mathsf{S}(u_1) = v(i + 2^{j-1})$.

4. The second part of the segment is traversed using the function P_{j-1} with the base node u_b. This "reverses" the changes to the first $j-1$ states made in the first part of the segment. When the function P_{j-1} reaches the end of its segment we have that $u_1, \ldots, u_{j-1}$ are all free (while $u_j = v(i + 2^{j-1})$ was not changed).

5. The third part of the segment is traversed using the function S_{j-1} with the base node $u_j = v(i+2^{j-1})$. When the function S_{j-1} reaches the end of its segment we have that for every $k \in [j-1]$:

$$u_k = v(i + 2^{j-1} + 2^{j-1} - 2^{k-1}) = v(i + 2^j - 2^{k-1}) \ ,$$

   as required.

Above we described the sequential execution of S_j on the nodes in each of the three parts of the segment. We now explain how S_j identifies which segment is currently being traversed (we continue to assume that the input node is indeed in the segment). Given an input node $N = (u_1, \ldots, u_t)$:

1. If the state u_j is free the node belongs the the first part of the segment.
2. If the state u_j is not free and for every $k \in [j-1]$, $u_k < u_j$ then the node belongs the second part of the segment.
3. If the state u_j is not free and for every $k \in [j-1]$, $u_k > u_j$ then the node belongs the third part of the segment.

Finally, we explain how S_j recognizes that the input node $N = (u_1, \ldots, u_t)$ is not in the segment, in which case it will output the node unchanged. A node is not in the segment iff one of the following occurs:

1. The node contains an invalid state $u_i = (x, i)$ such that $\mathsf{V}(x, i) \neq 1$.
2. The state u_j is not free and it is also not the state $v(i + 2^{j-1})$ where $u_b = v(i)$ is the base state.
3. There exist $k, k' \in [j-1]$ such that $u_k \leq u_j \leq u_{k'}$.
4. The call to S_{j-1} or to P_{j-1} did not modify the node N even though the node is not the end point of the segment traversed by S_{j-1} or by P_{j-1}.

4 Cryptographic Definitions

We define the cryptographic primitives that are required for our result. The definitions 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 each circuit $\mathcal{A}_\lambda$ is of polynomial size $\lambda^{O(1)}$ with $\lambda^{O(1)}$ input and output bits; we shall also consider adversaries of some super polynomial size $t(\lambda) = \lambda^{\omega(1)}$. We often omit the subscript λ when it is clear from the context. The resulting hardness will accordingly be against non-uniform algorithms. The result can be cast into the uniform setting, with some adjustments to the analysis.

4.1 Indistinguishability obfuscation

We define indistinguishability obfuscation (iO) with respect to a give class of circuits. The definition is formulated as in [BGI+01].

Definition 4.1 (Indistinguishability obfuscation [BGI+01])**.** *A PPT algorithm $i\mathcal{O}$ is said to be an* indistinguishability obfuscator *for a class of circuits $\mathcal{C}$, if it satisfies:*

1. **Functionality:** *for any $C \in \mathcal{C}$,*

$$\Pr_{i\mathcal{O}}\left[\forall x : i\mathcal{O}(C)(x) = C(x)\right] = 1 \ .$$

2. **Indistinguishability:** *for any polysize distinguisher $\mathcal{D}$ there exists a negligible function $\mu(\cdot)$, such that for any two circuits $C_0, C_1 \in \mathcal{C}$ that compute the same function and are of the same size λ:*

$$\left|\Pr[\mathcal{D}(i\mathcal{O}(C_0)) = 1] - \Pr[\mathcal{D}(i\mathcal{O}(C_1)) = 1]\right| \leq \mu(\lambda) \ ,$$

where the probability is over the coins of $\mathcal{D}$ and $i\mathcal{O}$.

We further say that $i\mathcal{O}$ is (t, δ)-secure, for some function $t(\cdot)$ and concrete negligible function $\delta(\cdot)$, if for all $t(\lambda)^{O(1)}$ distinguishers the above indistinguishability gap $\mu(\lambda)$ is smaller than $\delta(\lambda)^{\Omega(1)}$.

4.2 Puncturable Pseudorandom Functions

We consider a simple case of the puncturable pseudo-random functions (PRFs) where any PRF may be punctured at a single point. The definition is formulated as in [SW14], and is satisfied by the GGM [GGM86] PRF [BW13, KPTZ13, BGI14],

Definition 4.2 (Puncturable PRFs)**.** *Let n, k be polynomially bounded length functions. An efficiently computable family of functions*

$$\mathcal{PRF} = \left\{ \mathsf{PRF}_S : \{0,1\}^{n(\lambda)} \to \{0,1\}^{\lambda} \;\middle|\; S \in \{0,1\}^{k(\lambda)}, \lambda \in \mathbb{N} \right\} \enspace ,$$

associated with an efficient (probabilistic) key sampler $\mathcal{K}_{\mathcal{PRF}}$, is a puncturable PRF if there exists a poly-time puncturing algorithm Punc *that takes as input a key S, and a point x^*, and outputs a punctured key $S\{x^*\}$, so that the following conditions are satisfied:*

1. **Functionality is preserved under puncturing:** *For every $x^* \in \{0,1\}^{n(\lambda)}$,*

$$\Pr_{S \leftarrow \mathcal{K}_{\mathcal{PRF}}(1^\lambda)} \left[\forall x \neq x^* : \mathsf{PRF}_S(x) = \mathsf{PRF}_{S\{x^*\}}(x) \;\middle|\; S\{x^*\} = \mathsf{Punc}(S, x^*) \right] = 1 \enspace .$$

2. **Indistinguishability at punctured points:** *for any polysize distinguisher $\mathcal{D}$ there exists a negligible function $\mu(\cdot)$, such that for all $\lambda \in \mathbb{N}$, and any $x^* \in \{0,1\}^{n(\lambda)}$,*

$$|\Pr[\mathcal{D}(x^*, S\{x^*\}, \mathsf{PRF}_S(x^*)) = 1] - \Pr[\mathcal{D}(x^*, S\{x^*\}, u) = 1]| \leq \mu(\lambda) \enspace ,$$

where $S \leftarrow \mathcal{K}_{\mathcal{PRF}}(1^\lambda), S\{x^*\} = \mathsf{Punc}(S, x^*)$, and $u \leftarrow \{0,1\}^\lambda$.

We further say that $\mathcal{PRF}$ is (t, δ)-secure, for some function $t(\cdot)$ and concrete negligible function $\delta(\cdot)$, if for all $t(\lambda)^{O(1)}$ distinguishers the above indistinguishability gap $\mu(\lambda)$ is smaller than $\delta(\lambda)^{\Omega(1)}$.

4.3 Injective One-Way Functions

We shall also rely on (possibly keyed) injective one-way functions.

Definition 4.3 (Injective OWF)**.** *Let ℓ, k be polynomially bounded length functions. An efficiently computable family of functions*

$$\mathcal{OWF} = \left\{ \mathsf{OWF}_K : \{0,1\}^{\lambda} \to \{0,1\}^{\ell(\lambda)} \;\middle|\; K \in \{0,1\}^{k(\lambda)}, \lambda \in \mathbb{N} \right\} \enspace ,$$

associated with an efficient (probabilistic) key sampler $\mathcal{K}_{\mathcal{OWF}}$, is an injective OWF if every function in the family is injective and for any polysize inverter $\mathcal{A}$ there exists a negligible function $\mu(\cdot)$, such that for all $\lambda \in \mathbb{N}$,

$$\Pr \left[\mathcal{A}(K, \mathsf{OWF}_K(x)) = x \;\middle|\; \begin{array}{c} K \leftarrow \mathcal{K}_{\mathcal{OWF}}(1^\lambda) \\ x \leftarrow \{0,1\}^\lambda \end{array} \right] \leq \mu(\lambda) \enspace .$$

We further say that $\mathcal{OWF}$ is (t, δ)-secure, for some function $t(\cdot)$ and concrete negligible function $\delta(\cdot)$, if for all $t(\lambda)^{O(1)}$ inverters the above inversion probability $\mu(\lambda)$ is smaller than $\delta(\lambda)^{\Omega(1)}$.

5 Hardness of Sink-of-Verifiable-Line

In this section, we show that the SVL problem is hard, assuming indistinguishability obfuscation (iO). As a corollary, we deduce that the EOL problem is hard.

Our basic construction will show that SVL is not only hard in the worst-case, but also in the average case. Concretely, we construct a PPT sampler $\mathcal{I}(1^\lambda)$ for SVL instances of $\lambda^{O(1)}$ size such that no of $\lambda^{O(1)}$-size $\mathcal{A}$ can solve an instance sampled by $\mathcal{I}$ accept with some negligible probability $\lambda^{-\omega(1)}$. We rely on super-polynomial hardness assumptions; for a convenient setting of parameters we describe the basic sampler assuming that all underlying cryptographic primitives are sub-exponentially hard. Accordingly we get sub-exponential hardness of SVL (albeit with some loss in parameters). In Section 5.5, we discuss relaxations to more mild (but still super-polynomial) hardness.

5.1 Ingredients

Fix any constant $\varepsilon < 1$, and let $T = T(\lambda) = 2^{\lambda^{\varepsilon/2}}$. We require the following primitives:

- $i\mathcal{O}$ is a $(2^{\lambda^\varepsilon}, 2^{-\lambda^\varepsilon})$-secure indistinguishability obfuscator for P/poly.
- $\mathcal{PRF}$ is a $(2^{\lambda^\varepsilon}, 2^{-\lambda^\varepsilon})$-secure family of puncturable pseudo-random functions, which for $\lambda \in \mathbb{N}$ maps $[T]$ to $\{0,1\}^\lambda$.
- $\mathcal{OWF}$ is a $(2^{\lambda^\varepsilon}, 2^{-\lambda^\varepsilon})$-secure family of injective one-way functions, which for $\lambda \in \mathbb{N}$ maps $\{0,1\}^\lambda$ to $\{0,1\}^{\ell(\lambda)}$, for some $\lambda \leq \ell(\lambda) \leq \lambda^{O(1)}$.

5.2 Obfuscated verify-and-sign

The core of the hard SVL distribution produced by $\mathcal{I}$ will be an obfuscated *verify and sign* circuit that given a valid signature on an index i produces a signature on the next index $i+1$, where signatures will be implemented by the puncturable PRF. The circuit is formally described in Figure 1.

```
                              $\mathsf{VS}_S$

Hardwired: a PRF key $S \leftarrow \mathcal{K}_{\mathcal{PRF}}(1^\lambda)$.

Input: index $i \in [T]$, string $\sigma \in \{0,1\}^\lambda$.

        1. If $\sigma \neq \mathsf{PRF}_S(T)$, return $\perp$.
        2. If $i = T$, return $\mathsf{SOLVED}$.
        3. Return $i+1, \mathsf{PRF}_S(i+1)$.

Padding: The circuit is padded so that its total size is $s(\lambda)$, for a fixed polynomial $s(\cdot)$ specified later.
```

Formally $\perp$ and SOLVED are represented by some canonical strings in $\{0,1\}^{\log T + \lambda}$.

Figure 1: The circuit VS_S.

5.3 The hard SVL distribution

A random instance $\Phi_{\widetilde{\mathsf{VS}},\sigma_1} \leftarrow \mathcal{I}(1^\lambda)$ is associated with a (random) obfuscation $\widetilde{\mathsf{VS}}$ of a verify-and-sign circuit (with respect to a random PRF seed) and a signature σ_1 on $1 \in [T]$. This induces a SVL instance $(\mathsf{S}, \mathsf{V}, x_s, T)$ where the successor circuit S computes $\widetilde{\mathsf{VS}}$, the verification circuit V uses $\widetilde{\mathsf{VS}}$ to test inputs along the chain from the source input $x_s = (1, \sigma_1)$ to the target input (T, σ_T). The SVL distribution is formally described in Figure 2.

$$\Phi_{\widetilde{\mathsf{VS}},\sigma_1} \leftarrow \mathcal{I}(1^\lambda)$$

Sampling: Given security parameter 1^λ,

1. Sample $S \leftarrow \mathcal{K}_{\mathcal{PRF}}(1^\lambda)$.
2. Compute an obfuscation $\widetilde{\mathsf{VS}} \leftarrow i\mathcal{O}(\mathsf{VS}_S)$.
3. Compute a signature $\sigma_1 = \mathsf{PRF}_S(1)$.

The induced instance: for $(\widetilde{\mathsf{VS}}, \sigma_1)$ sampled as above, let $\Phi_{\widetilde{\mathsf{VS}},\sigma_1} = (\mathsf{S}, \mathsf{V}, x_s, T)$, where

1. $\mathsf{S} : \{0,1\}^{\log T + \lambda} \to \{0,1\}^{\log T + \lambda}$,
   $\mathsf{S}(i, \sigma) = \widetilde{\mathsf{VS}}(i, \sigma)$.
2. $\mathsf{V} : \{0,1\}^{\log T + \lambda} \times [T] \to \{0,1\}$,
   $\mathsf{V}((i, \sigma), j) = 1$ if and only if $i = j$ and $\widetilde{\mathsf{VS}}(i, \sigma) \neq \perp$.
3. $x_s = (1, \sigma_1)$.
4. $T = T(\lambda)$.

Figure 2: Sampling the hard SVL distribution.

$\mathcal{I}$ is an SVL sampler. To show that $\mathcal{I}$ indeed samples SVL instances $(\mathsf{S}, \mathsf{V}, x_s, T)$ according to Definition 2.2, we only need to check that, for $i \in [T]$ and $x \in \{0,1\}^{\log T + \lambda}$, $\mathsf{V}(x, i) = 1$ if and only if $x = \mathsf{S}^{i-1}(x_s)$; the rest of the requirements are purely syntactic and are satisfied by the way we have defined our sampler. To see that the verification requirement is satisfied, note that by the definition of VS_S, S, V, and x_s it holds that $\mathsf{S}^{i-1}(x_s) = \mathsf{VS}_S^{i-1}(1, \sigma_1) = (i, \mathsf{PRF}_S(i))$, and $\mathsf{V}((j, \sigma), i) = 1$ if and only if $j = i$ and $\mathsf{VS}(j, \sigma) \neq \perp$, implying that $(j, \sigma) = (i, \mathsf{PRF}_S(i))$.

5.4 Hardness

We now show that $\mathcal{I}$ samples hard SVL instances; namely instances $(\mathsf{S}, \mathsf{V}, x_s, T)$ for which circuits of some sub-exponential-size cannot find the target $x_T = \mathsf{S}^{T-1}(x_s)$. We prove the following proposition.

Proposition 5.1. *For any $\mathcal{A}$ of size $2^{O(\lambda^{\varepsilon^2})}$ and every $\lambda \in \mathbb{N}$,*

$$\Pr\left[\mathsf{PRF}_S(T) \leftarrow \mathcal{A}(\widetilde{\mathsf{VS}}, \sigma_1) \;\middle|\; \begin{array}{c} S \leftarrow \mathcal{K}_{\mathcal{PRF}}(1^\lambda) \\ \widetilde{\mathsf{VS}} \leftarrow i\mathcal{O}(\mathsf{VS}_S) \\ \sigma_1 \leftarrow \mathsf{PRF}_S(1) \end{array}\right] \leq 2^{-\Omega(\lambda^{\varepsilon^2})} \;.$$

Proof. Fix any such $\mathcal{A}$. To prove the proposition we show that except with sub-exponentially-small probability $\mathcal{A}(\widetilde{\mathsf{VS}}, \sigma_1)$ cannot output σ^* such that $\widetilde{\mathsf{VS}}(T, \sigma^*) \neq \perp$, which is equivalent to showing that $\sigma^* \neq \mathsf{PRF}_S(T)$. We prove this via a sequence of indistinguishable hybrid experiments where the obfuscated $\widetilde{\mathsf{VS}}$ is gradually augmented to return $\perp$ on an increasing interval, until it eventually returns $\perp$ on some interval $[u, T]$ (for every possible signature), meaning in particular that $\mathcal{A}(\widetilde{\mathsf{VS}}, \sigma_1)$ cannot find an accepting signature σ^* for T. The second input σ_1 remains $\mathsf{PRF}_S(1)$ throughout all hybrids.

Hyb_1**:** The original experiment, where $\widetilde{\mathsf{VS}}$ is an $i\mathcal{O}$ of $\mathsf{VS}_S = \mathsf{VS}_S^{(1)}$.

Hyb_2**:** Here $\widetilde{\mathsf{VS}}$ is an $i\mathcal{O}$ of a circuit $\mathsf{VS}^{(2)}_{v,S,K'}$. The circuit has a random one-way function image $v = \mathsf{OWF}_{K'}(u)$, and on any input (i, σ), it returns $\perp$ if $\mathsf{OWF}_{K'}(i) = v$. The circuit is formally described in Figure 3.

$$\mathsf{VS}^{(2)}_{v,S,K'}$$

Hardwired: a PRF key $S \leftarrow \mathcal{K}_{\mathcal{PRF}}(1^\lambda)$, an injective OWF key $K' \leftarrow \mathcal{K}_{\mathcal{OWF}}(1^{\lambda'})$ for $\lambda' = \log T$, an image $v = \mathsf{OWF}_{K'}(u)$, for $u \leftarrow [T]$.

Input: index $i \in [T]$, string $\sigma \in \{0,1\}^\lambda$.

```
1. If $\mathsf{OWF}_{K'}(i) = v$, return $\perp$.
2. If $\sigma \neq \mathsf{PRF}_S(i)$, return $\perp$.
3. If $i = T$, return SOLVED.
4. Return $i + 1, \mathsf{PRF}_S(i + 1)$.
```

Padding: The circuit is padded so that its total size is $s(\lambda)$, for a fixed polynomial $s(\cdot)$ specified later.

Figure 3: The circuit $\mathsf{VS}^{(2)}_{v,S,K'}$.

$\mathsf{Hyb}_{3,j}, j \in [T+1]$**:** Here $\widetilde{\mathsf{VS}}$ is an $i\mathcal{O}$ of a circuit $\mathsf{VS}^{(3,j)}_{u,S}$. The circuit has a random index u, and on any input (i, σ), it returns $\perp$ if $i \in [u, u+j]$. The circuit is formally described in Figure 4.

$\mathsf{VS}^{(3,j)}_{u,S}$

Hardwired: a PRF key $S \leftarrow \mathcal{K}_{\mathcal{PRF}}(1^\lambda)$, a random index $u \leftarrow [T]$, and $j = \min\{j, T-u\}$.[a]

Input: index $i \in [T]$, string $\sigma \in \{0,1\}^\lambda$.

1. If $i \in [u, u+j-1]$, return $\perp$.
2. If $\sigma \neq \mathsf{PRF}_S(i)$, return $\perp$.
3. If $i = T$, return SOLVED.
4. Return $i+1, \mathsf{PRF}_S(i+1)$.

Padding: The circuit is padded so that its total size is $s(\lambda)$, for a fixed polynomial $s(\cdot)$ specified later.

[a]This is a convenient abuse of notation, which should be interpreted as "if $j > T-u$, truncate it".

Figure 4: The circuit $\mathsf{VS}^{(3,j)}_{u,S}$.

$\mathsf{Hyb}_{4,j}, j \in [T]$**:** Here $\widetilde{\mathsf{VS}}$ is an iO of a circuit $\mathsf{VS}^{(4,j)}_{u,S\{u+j\},\sigma_{u+j}}$. The circuit is the same as $\mathsf{VS}^{(3,j)}_{u,S}$, only that is has a punctured PRF key $S\{u+j\}$, and the value $\sigma_{u+j} = \mathsf{PRF}_S(u+j)$ is hardwired. The circuit is formally described in Figure 5.

$\mathsf{VS}^{(4,j)}_{u,S\{u+j\},\sigma_{u+j}}$

Hardwired: a random index $u \leftarrow [T]$, $j = \min\{j, T-u\}$, a punctured PRF key $S\{u+j\} \leftarrow \mathsf{Punc}(S, u+j)$, the PRF value $\sigma_{u+j} = \mathsf{PRF}_S(u+j)$, where $S \leftarrow \mathcal{K}_{\mathcal{PRF}}(1^\lambda)$.

Input: index $i \in [T]$, string $\sigma \in \{0,1\}^\lambda$.

1. If $i \in [u, u+j-1]$, return $\perp$.
2. If $i = u+j$ and $\sigma \neq \sigma_{u+j}$, return $\perp$.
3. If $\sigma \neq \mathsf{PRF}_S(i)$, return $\perp$.
4. If $i = T$, return SOLVED.
5. Return $i+1, \mathsf{PRF}_S(i+1)$.

Padding: The circuit is padded so that its total size is $s(\lambda)$, for a fixed polynomial $s(\cdot)$ specified later.

Figure 5: The circuit $\mathsf{VS}^{(4,j)}_{u,S\{u+j\},\sigma_{u+j}}$.

$\mathsf{Hyb}_{5,j}, j \in [T]$**:** Here $\widetilde{\mathsf{VS}}$ is an iO of a circuit $\mathsf{VS}^{(5,j)}_{u,S\{u+j\},\sigma_{u+j}}$. The circuit is the same as $\mathsf{VS}^{(4,j)}_{u,S\{u+j\},\sigma_{u+j}}$, only that the hardwired σ_{u+j} is not set to $\mathsf{PRF}_S(u+j)$, but sampled uniformly at random from $\{0,1\}^\lambda$,

$\mathsf{Hyb}_{6,j}, j \in [T]$**:** Here $\widetilde{\mathsf{VS}}$ is an iO of a circuit $\mathsf{VS}^{(6,j)}_{u,S,v,K}$. The circuit is the same as $\mathsf{VS}^{(5,j)}_{u,S\{u+j\},\sigma_{u+j}}$, only that instead of storing σ_{u+j} in the clear $v = \mathsf{OWF}_K(\sigma_{u+j})$ is stored, and comparison of σ and σ_{u+j} is done by comparing $\mathsf{OWF}_K(\sigma)$ and $\mathsf{OWF}_K(\sigma_{u+j})$. Here K is a key for an injective OWF from the family $\mathcal{OWF}$. In addition, the PRF seed S is no longer punctured. The circuit is formally described in Figure 6.

$$\mathsf{VS}^{(6,j)}_{u,S,v,K}$$

Hardwired: a random index $u \leftarrow [T]$, $j = \min\{j, T-u\}$, a PRF key $S \leftarrow \mathcal{K}_{\mathcal{PRF}}(1^\lambda)$, an image $v = \mathsf{OWF}_K(\sigma_{u+j})$, where $K \leftarrow \mathcal{K}_{\mathcal{OWF}}(1^\lambda)$, $\sigma_{u+j} \leftarrow \{0,1\}^\lambda$.

Input: index $i \in [T]$, string $\sigma \in \{0,1\}^\lambda$.

1. If $i \in [u, u+j-1]$, return $\perp$.
2. If $i = u+j$ and $\mathsf{OWF}_K(\sigma) \neq v$, return $\perp$.
3. If $\sigma \neq \mathsf{PRF}_S(i)$, return $\perp$.
4. If $i = T$, return SOLVED.
5. Return $i+1, \mathsf{PRF}_S(i+1)$.

Padding: The circuit is padded so that its total size is $s(\lambda)$, for a fixed polynomial $s(\cdot)$ specified later.

Figure 6: The circuit $\mathsf{VS}^{(6,j)}_{u,S,v,K}$.

The padding parameter $s(\lambda)$**.** We choose $s(\lambda)$ so that each of the circuits $\widetilde{\mathsf{VS}}^{\cdots}_{\cdots}$ considered above can be implemented by a circuit of size at most $s(\lambda)/3$. (The extra $1/3$ slack is taken to satisfy Lemma 5.1 in the analysis below.)

We prove the following:

Claim 5.1. *For any $2^{O(\lambda^{\varepsilon^2})}$-size distinguisher $\mathcal{D}$, all $\lambda \in \mathbb{N}$, and all $j \in [T]$:*

1. $|\Pr[\mathcal{D}(\mathsf{Hyb}_1) = 1] - \Pr[\mathcal{D}(\mathsf{Hyb}_2) = 1]| \leq 2^{-\Omega(\lambda^{\varepsilon^2})}$,
2. $\left|\Pr[\mathcal{D}(\mathsf{Hyb}_2) = 1] - \Pr[\mathcal{D}(\mathsf{Hyb}_{3,1}) = 1]\right| \leq 2^{-\Omega(\lambda^{\varepsilon})}$,
3. $\left|\Pr[\mathcal{D}(\mathsf{Hyb}_{3,j}) = 1] - \Pr[\mathcal{D}(\mathsf{Hyb}_{4,j}) = 1]\right| \leq 2^{-\Omega(\lambda^{\varepsilon})}$,
4. $\left|\Pr[\mathcal{D}(\mathsf{Hyb}_{4,j}) = 1] - \Pr[\mathcal{D}(\mathsf{Hyb}_{5,j}) = 1]\right| \leq 2^{-\Omega(\lambda^{\varepsilon})}$,
5. $\left|\Pr[\mathcal{D}(\mathsf{Hyb}_{5,j}) = 1] - \Pr[\mathcal{D}(\mathsf{Hyb}_{6,j}) = 1]\right| \leq 2^{-\Omega(\lambda^{\varepsilon})}$,
6. $\left|\Pr[\mathcal{D}(\mathsf{Hyb}_{6,j}) = 1] - \Pr[\mathcal{D}(\mathsf{Hyb}_{3,j+1}) = 1]\right| \leq 2^{-\Omega(\lambda^{\varepsilon})}$,

where the view of $\mathcal{D}$ in each hybrid consists of the corresponding obfuscated $\widetilde{\mathsf{VS}}$ and $\sigma_1 = \mathsf{PRF}_S(1)$.

Proving the above claim will conclude the proof of Proposition 5.1 since it implies that

$$\begin{aligned}
&\Pr\left[\begin{array}{c}\sigma \leftarrow \mathcal{A}(\widetilde{\mathsf{VS}},\sigma_1)\\ \widetilde{\mathsf{VS}}(T,\sigma)\neq\bot\end{array}\middle|\, \widetilde{\mathsf{VS}} \leftarrow i\mathcal{O}(\mathsf{VS}_S)\right] \leq \\
&\Pr\left[\begin{array}{c}\sigma \leftarrow \mathcal{A}(\widetilde{\mathsf{VS}},\sigma_1)\\ \widetilde{\mathsf{VS}}(T,\sigma)\neq\bot\end{array}\middle|\, \widetilde{\mathsf{VS}} \leftarrow i\mathcal{O}(\mathsf{VS}^{(3,T+1)}_{S,u})\right] + 2^{-\Omega(\lambda^{\varepsilon^2})} \qquad +T\cdot 2^{-\Omega(\lambda^{\varepsilon})} = \\
&\qquad\qquad 0 + 2^{-\Omega(\lambda^{\varepsilon^2})} \qquad +2^{\lambda^{\varepsilon/2}}\cdot 2^{-\Omega(\lambda^{\varepsilon})} = 2^{-\Omega(\lambda^{\varepsilon^2})} \ ,
\end{aligned}$$

where the first to last equality follows from the fact that $\mathsf{VS}^{(3,T+1)}_{S,u}(T,\sigma)=\bot$ for any σ.

Proof of Claim 5.1. We now prove each of the items in the claim.

Proof of 1 and 6. Recall that here we need to show that

1. $|\Pr[\mathcal{D}(\mathsf{Hyb}_1)=1]-\Pr[\mathcal{D}(\mathsf{Hyb}_2)=1]|\leq 2^{-\Omega(\lambda^{\varepsilon^2})}$,

6. $\left|\Pr[\mathcal{D}(\mathsf{Hyb}_{6,j})=1]-\Pr[\mathcal{D}(\mathsf{Hyb}_{3,j+1})=1]\right|\leq 2^{-\Omega(\lambda^{\varepsilon})}$.

In both cases, one obfuscated program differs from the other on exactly a single point, which is the unique (random) preimage of the corresponding image v (in the first case $v=\mathsf{OWF}_{K'}(u)$, and in the second $v=\mathsf{OWF}_K(\sigma_{u+j})$).

To prove the claim, we rely on a lemma proven in [BCP14] that roughly shows that, for circuits that only differ on a single input, iO implies what is known as *differing input obfuscation* [BGI+01], where it is possible to efficiently extract from any iO distinguisher an input on which the underlying circuits differ.

Lemma 5.1 (special case of [BCP14]). *Let $i\mathcal{O}$ be a (t,δ)-secure indistinguishability obfuscator for P/poly. There exists a PPT oracle-aided extractor $\mathcal{E}$, such that for any $t^{O(1)}$-size distinguisher $\mathcal{D}$, and two equal size circuits C_0,C_1 differing on exactly one input x^*, the following holds. Let C_0',C_1' be padded versions of C_0,C_1 of size $s\geq 3\cdot|C_0|$.*

$$\begin{array}{ll}
\textit{If} & |\Pr[\mathcal{D}(i\mathcal{O}(C_0')=1]-\Pr[\mathcal{D}(i\mathcal{O}(C_1')=1]|=\eta\geq\delta(s)^{o(1)} \ , \\
\textit{then} & \Pr\left[x^*\leftarrow\mathcal{E}^{\mathcal{D}(\cdot)}(1^{1/\eta},C_0,C_1)\right]\geq 1-2^{-\Omega(s)} \ .
\end{array}$$

Using the lemma, we show that if either item 1 or 6 do not hold, we can invoke the distinguisher $\mathcal{D}$ to invert the underlying one-way function. The argument is similar in both cases up to different parameters; for concreteness, we focus on the first. Assume that for infinitely many $\lambda\in\mathbb{N}$, $\mathcal{D}$ distinguishes Hyb_1 from Hyb_2 with gap $\eta(\lambda)=2^{-o(\lambda^{\varepsilon^2})}$. Then, by averaging, with probability $\eta(\lambda)/2$ over the choice of u,K', $\mathcal{D}$ distinguishes the two distributions conditioned on these choices with gap $\eta(\lambda)/2$. Thus, we can invoke the extractor $\mathcal{E}$ given by Lemma 5.1 to invert the one-way function family $\mathcal{OWF}$ with probability $\frac{\eta(\lambda)}{2}\cdot(1-2^{-\Omega(\lambda)})\geq 2^{-o(\lambda^{\varepsilon^2})}$ in time $t_{\mathcal{E}}(\lambda)\cdot t_{\mathcal{D}}(\lambda)\leq\eta(\lambda)^{-O(1)}\cdot 2^{O(\varepsilon^2)}=2^{O(\lambda^{\varepsilon^2})}$. Note that, indeed, given the image and the one-way function key, the inverter can construct the corresponding circuits efficiently. Recall that OWF'_K is defined on inputs of size $\lambda'=\log T=\lambda^{\varepsilon/2}$, and is $(2^{-\lambda'^{\varepsilon}},2^{\lambda'^{\varepsilon}})$-secure. Thus we get a contradiction to its one-wayness.

Proof of 2. Recall that here we need to show that

2. $\left|\Pr[\mathcal{D}(\mathsf{Hyb}_2)=1]-\Pr[\mathcal{D}(\mathsf{Hyb}_{3,1})=1]\right|\leq 2^{-\Omega(\lambda^{\varepsilon})}$.

Here the two obfuscated programs compute the exact same function. Specifically, a comparison in the clear of two values i and u is replaced by comparison of their corresponding values under an injective one-way function. Thus, the required indistinguishability follows from the $i\mathcal{O}$ $(2^{\lambda^\varepsilon}, 2^{-\lambda^\varepsilon})$-security.

Proof of 3. Recall that here we need to show that

3. $\left|\Pr[\mathcal{D}(\mathsf{Hyb}_{3,j}) = 1] - \Pr[\mathcal{D}(\mathsf{Hyb}_{4,j}) = 1]\right| \leq 2^{-\Omega(\lambda^\varepsilon)}$.

Here also, the two obfuscated programs compute the exact same function. Specifically, rather than computing $\sigma_{u+j} = \mathsf{PRF}_S(u + j)$ using the PRF key S, the value σ_{u+j} is hardwired and directly compared to σ. For any other index, the punctured key $S[u + j]$ is used. Thus, by the functionality guarantee of puncturing the two functions are the same, and indistinguishability follows from iO.

Proof of 4. Recall that here we need to show that

3. $\left|\Pr[\mathcal{D}(\mathsf{Hyb}_{4,j}) = 1] - \Pr[\mathcal{D}(\mathsf{Hyb}_{5,j}) = 1]\right| \leq 2^{-\Omega(\lambda^\varepsilon)}$.

The only difference between the two obfuscated circuit distributions is that in the first the hardwired value σ_{u+j} is $\mathsf{PRF}_S(u + j)$, whereas in the second it is sampled independently uniformly at random. Indistinguishability follows from the $(2^{\lambda^\varepsilon}, 2^{-\lambda^\varepsilon})$-pseudo-randomness at the punctured point guarantee. Note that, indeed, given punctured key $S[u + j]$ and σ_{u+j}, a distinguisher can construct the corresponding circuits efficiently.

Proof of 5. Recall that here we need to show that

5. $\left|\Pr[\mathcal{D}(\mathsf{Hyb}_{5,j}) = 1] - \Pr[\mathcal{D}(\mathsf{Hyb}_{6,j}) = 1]\right| \leq 2^{-\Omega(\lambda^\varepsilon)}$.

Here also, the two obfuscated programs compute the exact same function. First, the comparison of σ and σ_{u+j} is replaced by comparison of their corresponding values under an injective one-way function. In addition, the punctured key $S[u + j]$ is replaced with a non-punctured key S. This does not affect functionality as the two keys compute the same function on all points except $u + j$, and the circuits in the two hybrids treat any input $u + j, \sigma$, independently of the PRF key. Thus, indistinguishability follows from iO.

This concludes the proof of the Claim 5.1 and Proposition 5.1. □

□

5.5 Hardness Tradeoffs

In the hard SVL distribution constructed above, we assumed all cryptographic primitives are sub-exponentially hard. We now explain how this can be relaxed, and what are the tradeoffs between the hardness of the different primitives (and the SVL hardness itself). Let $f(\cdot), g(\cdot), h(\cdot)$ be sub-linear functions and assume that $\mathcal{OWF}$ is $(2^{f(\lambda)}, 2^{-f(\lambda)})$-secure, $\mathcal{PRF}$ is $(2^{g(\lambda)}, 2^{-g(\lambda)})$-secure, and $i\mathcal{O}$ is $(2^{-h(\lambda)}, 2^{-h(\lambda)})$-secure. We can restate Claim 5.1 as follows.

Claim 5.2 (Claim 5.1 generalized)**.** *For any distinguisher $\mathcal{D}$ of size at most $2^{O(m(\lambda))}$ where $m(\lambda) = \min(f(\lambda), g(\lambda), h(\lambda))$, all $\lambda \in \mathbb{N}$, and all $j \in [T]$:*

1. $|\Pr[\mathcal{D}(\mathsf{Hyb}_1) = 1] - \Pr[\mathcal{D}(\mathsf{Hyb}_2) = 1]| \leq 2^{-\Omega(f(\log T))} + 2^{-\Omega(h(\lambda))}$,
2. $\left|\Pr[\mathcal{D}(\mathsf{Hyb}_2) = 1] - \Pr[\mathcal{D}(\mathsf{Hyb}_{3,1}) = 1]\right| \leq 2^{-\Omega(h(\lambda))}$,
3. $\left|\Pr[\mathcal{D}(\mathsf{Hyb}_{3,j}) = 1] - \Pr[\mathcal{D}(\mathsf{Hyb}_{4,j}) = 1]\right| \leq 2^{-\Omega(h(\lambda))}$,

4. $\left|\Pr[\mathcal{D}(\mathsf{Hyb}_{4,j})=1]-\Pr[\mathcal{D}(\mathsf{Hyb}_{5,j})=1]\right| \le 2^{-\Omega(g(\lambda))}$,
5. $\left|\Pr[\mathcal{D}(\mathsf{Hyb}_{5,j})=1]-\Pr[\mathcal{D}(\mathsf{Hyb}_{6,j})=1]\right| \le 2^{-\Omega(h(\lambda))}$,
6. $\left|\Pr[\mathcal{D}(\mathsf{Hyb}_{6,j})=1]-\Pr[\mathcal{D}(\mathsf{Hyb}_{3,j+1})=1]\right| \le 2^{-\Omega(f(\lambda))}+2^{-\Omega(h(\lambda))}$.

The overall probability of solving the SVL can be then bounded by

$$2^{-\Omega(f(\log T))}+T\cdot\left(2^{-\Omega(f(\lambda))}+2^{-\Omega(g(\lambda))}+2^{-\Omega(h(\lambda))}\right) .$$

In particular, for $m(\cdot)$ defined as above, we can guarantee $(2^{m(\lambda)},2^{-m(\lambda)})$-hardness of SVL as long as

1. $m(\lambda)=\log(T)+\omega(\log m(\lambda))$.
2. $f(\log T)=\omega(\log m(\lambda))$.

For instance, for any constant $\varepsilon<1$, we can set

- $T=2^{(\log\lambda)^{2/\varepsilon}}$,
- $f(\lambda)=\lambda^{\varepsilon}$ ($\mathcal{OWF}$ is still sub-exponential),
- $g(\lambda)=h(\lambda)=(\log\lambda)^{2+2/\varepsilon}$ ($\mathcal{PRF}$ and $i\mathcal{O}$ are qausipolynomial).

Alternatively, we can set

- $T=2^{2^{(\log\lambda)^{\varepsilon}}}$,
- $f(\lambda)=g(\lambda)=h(\lambda)=2^{(\log\lambda)^{\frac{1+\varepsilon}{2}}}$ (all primitives are only $2^{\lambda^{o(1)}}$-secure).

Can we rely only on polynomial hardness? We do not know how to show SVL hardness based only on polynomially secure primitives. In particular, with the approach described above $\mathcal{OWF}$ cannot be quasi-polynomially (let alone polynomially) secure, since $f(f(\lambda))=\omega(\log m(\lambda))=\omega(\log f(\lambda))$. In addition, $T(\lambda)$ cannot be polynomial as long as we aim to deal with distinguishers of arbitrary polysize $\lambda^{O(1)}$, which in turn implies that $\mathcal{PRF}$ and $i\mathcal{O}$ cannot be polynomially secure.

We could, however, relax $i\mathcal{O},\mathcal{PRF}$ to be polynomially secure if we assume $\mathcal{OWF}$ is *exponentially-secure*; concretely, that $\mathcal{OWF}$ cannot be inverted with probability greater than $2^{-\varepsilon\lambda}$ in time less than $2^{\varepsilon\lambda}$ for some constant $\varepsilon<1$. Moreover, doing so we need to settle for worst-case hardness of SVL, rather than average-case as above. Specifically, rather than considering SVL instances with respect to one fixed function $T(\lambda)$, we can consider SVL instances with respect to every $T(\lambda)\in\left\{\lambda,\lambda^2,\ldots,\lambda^{\log\lambda}\right\}$. In the resulting distribution, for any λ^c-size circuit family, there exists a sequence of instances (or distributions on instances) on which it fails; these are the instances with $T\gg\lambda^{c'\cdot c/\varepsilon}$, where c' is a constant that depends on polynomial overhead of the extractor given by Lemma 5.1.

Acknowledgements

We thank Andrej Bogdanov, Ran Canetti, Tal Moran, Moni Naor, Guy Rothblum and Paul Valiant for insightful conversations. We are especially grateful to Moni for discussing reversible computation and for pointing out the manuscript of Abbot, Kane and Valiant.

References

[AKV04] Tim Abbot, Daniel Kane, and Paul Valiant. On algorithms for nash equilibria. Unpublished manuscript. `http://web.mit.edu/tabbott/Public/final.pdf`, 2004.

[BCC$^+$14] Nir Bitansky, Ran Canetti, Henry Cohn, Shafi Goldwasser, Yael Tauman Kalai, Omer Paneth, and Alon Rosen. The impossibility of obfuscation with auxiliary input or a universal simulator. In *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part II*, pages 71–89, 2014.

[BCP14] Elette Boyle, Kai-Min Chung, and Rafael Pass. On extractability obfuscation. In *TCC*, pages 52–73, 2014.

[Ben89] Charles H. Bennett. Time/space trade-offs for reversible computation. *SIAM J. Comput.*, 18(4):766–776, 1989.

[BGI$^+$01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In *CRYPTO*, pages 1–18, 2001.

[BGI14] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In *Public Key Cryptography*, pages 501–519, 2014.

[BGK$^+$14] Boaz Barak, Sanjam Garg, Yael Tauman Kalai, Omer Paneth, and Amit Sahai. Protecting obfuscation against algebraic attacks. In *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, pages 221–238, 2014.

[BR14] Zvika Brakerski and Guy N. Rothblum. Virtual black-box obfuscation for all circuits via generic graded encoding. In *Theory of Cryptography - 11th Theory of Cryptography Conference, TCC 2014, San Diego, CA, USA, February 24-26, 2014. Proceedings*, pages 1–25, 2014.

[BW13] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In *ASIACRYPT (2)*, pages 280–300, 2013.

[CDG01] Fan Chung, Persi Diaconis, and Ronald Graham. Combinatorics for the east model. *Adv. in Appl. Math*, 27:200–1, 2001.

[CDT09] Xi Chen, Xiaotie Deng, and Shang-Hua Teng. Settling the complexity of computing two-player nash equilibria. *J. ACM*, 56(3), 2009.

[DGP09] Constantinos Daskalakis, Paul W. Goldberg, and Christos H. Papadimitriou. The complexity of computing a nash equilibrium. *SIAM J. Comput.*, 39(1):195–259, 2009.

[GGH$^+$13] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *FOCS*, 2013.

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *J. ACM*, 33(4):792–807, 1986.

[GK05] Shafi Goldwasser and Yael Tauman Kalai. On the impossibility of obfuscation with auxiliary input. In *FOCS*, pages 553–562, 2005.

[GLSW14] Craig Gentry, Allison B. Lewko, Amit Sahai, and Brent Waters. Indistinguishability obfuscation from the multilinear subgroup elimination assumption. *IACR Cryptology ePrint Archive*, 2014:309, 2014.

[Gol11] Paul W. Goldberg. A survey of ppad-completeness for computing nash equilibria. *CoRR*, abs/1103.2709, 2011.

[HPV89] Michael D. Hirsch, Christos H. Papadimitriou, and Stephen A. Vavasis. Exponential lower bounds for finding brouwer fix points. *J. Complexity*, 5(4):379–416, 1989.

[JE91] J. Jckle and S. Eisinger. A hierarchically constrained kinetic ising model. *Zeitschrift fr Physik B Condensed Matter*, 84(1):115–124, 1991.

[KPTZ13] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In *ACM Conference on Computer and Communications Security*, pages 669–684, 2013.

[LR88] Michael Luby and Charles Rackoff. How to construct pseudorandom permutations from pseudorandom functions. *SIAM J. Comput.*, 17(2):373–386, 1988.

[MP91] Nimrod Megiddo and Christos H. Papadimitriou. On total functions, existence theorems and computational complexity. *Theor. Comput. Sci.*, 81(2):317–324, 1991.

[Nas51] John Nash. Non-cooperative Games. *The Annals of Mathematics*, 54(2):286–295, 1951.

[OPR14] Abraham Othman, Christos H. Papadimitriou, and Aviad Rubinstein. The complexity of fairness through equilibrium. In *ACM Conference on Economics and Computation, EC '14, Stanford , CA, USA, June 8-12, 2014*, pages 209–226, 2014.

[Pap94] Christos H. Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. *J. Comput. Syst. Sci.*, 48(3):498–532, 1994.

[PST14] Rafael Pass, Karn Seth, and Sidharth Telang. Indistinguishability obfuscation from semantically-secure multilinear encodings. In *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I*, pages 500–517, 2014.

[Rub14] Aviad Rubinstein. Inapproximability of nash equilibrium. *CoRR*, abs/1405.3322, 2014.

[SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: Deniable encryption, and more. In *STOC*, 2014.

[Zim14] Joe Zimmerman. How to obfuscate programs directly. *IACR Cryptology ePrint Archive*, 2014:776, 2014.

A Pseudo-code Descriptions of S_j and P_j

Algorithm A.1 The function S_1:

```
Input: Base state $u_b = (x, i)$ and the current node $N = (u_1, \dots, u_t)$
1: if $N$ contain an invalid state then
2:    return $N$ unchanged {Not on path (Condition 1)}
3: end if
4: if $u_j$ is free then
5:    Set $u_1 \leftarrow \mathsf{S}(u_b)$
6:    return $(u_1, \dots, u_t)$
7: else
8:    return $N$ unchanged {End of path or not on path (Condition 2)}
9: end if
```

Algorithm A.2 The function S_j for $j > 1$:

```
Input: Base state $u_b = (x, i)$ and the current node $N = (u_1, \dots, u_t)$
1: if $N$ contain an invalid state then
2:    return $N$ unchanged {Not on path (Condition 1)}
3: end if
4: if $u_j$ is free then
5:    $N' \leftarrow \mathsf{S}_{j-1}(u_b, N)$ {First part}
6:    if $N' \neq N$ then
7:       return $N'$
8:    else if for all $k \in [j-1]$, $u_k = v(i + 2^{j-1} - 2^{k-1})$ then
9:       Set $u_j \leftarrow \mathsf{S}(u_1)$ {First part ended, first step of second part}
10:      return $(u_1, \dots, u_t)$
11:   end if
12:   return $N$ unchanged {Not on path (Condition 4)}
13: else if $u_j = v(i + 2^{j-1})$ then
14:   if for every $k \in [j-1]$, $u_k$ is either free or $u_k < u_j$ then
15:      $N' \leftarrow \mathsf{P}_{j-1}(u_b, N)$ {Second part}
16:      if $N' \neq N$ then
17:         return $N'$
18:      else if for all $k \in [j-1]$, $u_k$ is free then
19:         return $\mathsf{S}_{j-1}(u_j, N)$ {Second part ended, first step of third part}
20:      end if
21:      return $N$ unchanged {Not on path (Condition 4)}
22:   else if for every $k \in [j-1]$, $u_k$ is either free or $u_k > u_j$ then
23:      return $\mathsf{S}_{j-1}(u_j, N)$ {Third part}
24:   end if
25: end if
26: return $N$ unchanged {Not on path (Conditions 2 and 3)}
```

Algorithm A.3 The function P_1:

Input: Base state $u_b = (x, i)$ and the current node $N = (u_1, \ldots, u_t)$

```
1: if $N$ contain an invalid state then
2:     return $N$ unchanged $\{$Not on path (Condition 1)$\}$
3: end if
4: if $u_j = v(i+1)$ then
5:     Set $u_1 \leftarrow v(1)$
6:     return $(u_1, \ldots, u_t)$
7: else
8:     return $N$ unchanged $\{$End of path or not on path (Condition 2)$\}$
9: end if
```

Algorithm A.4 The function P_j for $j > 1$:

Input: Base state $u_b = (x, i)$ and the current node $N = (u_1, \ldots, u_t)$

```
 1: if $N$ contain an invalid state then
 2:     return $N$ unchanged $\{$Not on path (Condition 1)$\}$
 3: end if
 4: if $u_j$ is free then
 5:     return $\mathsf{P}_{j-1}(u_b, N)$ $\{$Third part$\}$
 6: else if $u_j = v(i + 2^{j-1})$ then
 7:     if for every $k \in [j-1]$, $u_k$ is either free or $u_k < u_j$ then
 8:         $N' \leftarrow \mathsf{S}_{j-1}(u_b, N)$ $\{$Second part$\}$
 9:         if $N' \neq N$ then
10:             return $N'$
11:         else if for all $k \in [j-1]$, $u_k = v(i + 2^{j-1} - 2^{k-1})$ then
12:             Set $u_j \leftarrow v(1)$ $\{$Second part ended, first step of third part$\}$
13:             return $(u_1, \ldots, u_t)$
14:         end if
15:         return $N$ unchanged $\{$Not on path (Condition 4)$\}$
16:     else if for every $k \in [j-1]$, $u_k$ is either free or $u_k > u_j$ then
17:         $N' \leftarrow \mathsf{P}_{j-1}(u_j, N)$ $\{$First part$\}$
18:         if $N' \neq N$ then
19:             return $N'$
20:         else if for all $k \in [j-1]$, $u_k$ is free then
21:             return $\mathsf{S}_{j-1}(u_b, N)$ $\{$First part ended, first step of third part$\}$
22:         end if
23:     end if
24: end if
25: return $N$ unchanged $\{$Not on path (Conditions 2, 3, and 4)$\}$
```