Electronic Colloquium on Computational Complexity, Revision 2 of Report No. 59 (2016)

Can PPAD Hardness be Based on Standard Cryptographic Assumptions?

Alon Rosen* Gil Segev† Ido Shahaf†‡

Abstract

We consider the question of whether PPAD hardness can be based on standard cryptographic assumptions, such as the existence of one-way functions or public-key encryption. This question is particularly well-motivated in light of new devastating attacks on obfuscation candidates and their underlying building blocks, which are currently the only known source for PPAD hardness.

Central in the study of obfuscation-based PPAD hardness is the SINK-OF-VERIFIABLE-LINE (SVL) problem, an intermediate step in constructing instances of the PPAD-complete problem SOURCE-OR-SINK. Within the framework of black-box reductions we prove the following results:

- Average-case PPAD hardness (and even SVL hardness) does not imply any form of cryptographic hardness (not even one-way functions). Moreover, even when assuming the existence of one-way functions, average-case PPAD hardness (and, again, even SVL hardness) does not imply any public-key primitive. Thus, strong cryptographic assumptions (such as obfuscation-related ones) are not essential for average-case PPAD hardness.
- Average-case SVL hardness cannot be based either on standard cryptographic assumptions or on average-case PPAD hardness. In particular, average-case SVL hardness is not essential for average-case PPAD hardness.
- Any attempt for basing the average-case hardness of the PPAD-complete problem SOURCE-OR-SINK on standard cryptographic assumptions must result in instances with a nearly-exponential number of solutions. This stands in striking contrast to the obfuscation-based approach, which results in instances having a unique solution.

Taken together, our results imply that it may still be possible to base PPAD hardness on standard cryptographic assumptions, but any such black-box attempt must significantly deviate from the obfuscation-based approach: It cannot go through the SVL problem, and it must result in SOURCE-OR-SINK instances with a nearly-exponential number of solutions.

*Efi Arazi School of Computer Science, IDC Herzliya, Israel. Email: alon.rosen@idc.ac.il. Supported by ISF grant No. 1399/17 and via Project PROMETHEUS (Grant No. 780701).

†School of Computer Science and Engineering, Hebrew University of Jerusalem, Jerusalem 91904, Israel. Email: {segev,ido.shahaf}@cs.huji.ac.il. Supported by the European Union's 7th Framework Program (FP7) via a Marie Curie Career Integration Grant (Grant No. 618094), by the European Union's Horizon 2020 Framework Program (H2020) via an ERC Grant (Grant No. 714253), by the Israel Science Foundation (Grant No. 483/13), by the Israeli Centers of Research Excellence (I-CORE) Program (Center No. 4/11), by the US-Israel Binational Science Foundation (Grant No. 2014632), and by a Google Faculty Research Award.

‡Supported by the Clore Israel Foundation via the Clore Scholars Programme.

ISSN 1433-8092

Contents

1 Introduction **1**
- 1.1 Our Contributions . . . 2
- 1.2 Open Problems . . . 4
- 1.3 Overview of Our Approach . . . 6
- 1.4 Paper Organization . . . 9

2 Preliminaries **9**
- 2.1 Complexity Classes and Total Search Problems . . . 9
- 2.2 One-Way Functions and Injective Trapdoor Functions . . . 11
- 2.3 Key-Agreement Protocols . . . 11

3 Average-Case SVL Hardness Does Not Imply One-Way Functions **12**
- 3.1 Proof Overview . . . 13
- 3.2 $\mathcal{O}_{\mathsf{SVL}}$ is a Hard-on-Average SVL Instance . . . 15
- 3.3 Inverting Oracle-Aided Functions Relative to $\mathcal{O}_{\mathsf{SVL}}$ . . . 16
- 3.4 Proof of Theorem 3.2 . . . 18

4 Average-Case PPAD Hardness Does Not Imply Unique-TFNP Hardness **19**
- 4.1 Proof Overview . . . 20
- 4.2 $\mathcal{O}_{\mathsf{PPAD}}$ is a Hard-on-Average Source-or-Sink Instance . . . 22
- 4.3 Solving Oracle-Aided Unique-TFNP Instances Relative to $\mathcal{O}_{\mathsf{PPAD}}$ . . . 23
- 4.4 Proof of Theorem 4.2 . . . 26

5 One-Way Functions Do Not Imply Bounded-TFNP Hardness **27**
- 5.1 Proof Overview . . . 28
- 5.2 f is a One-Way Function . . . 29
- 5.3 Solving Oracle-Aided Bounded-TFNP Instances Relative to f . . . 30
- 5.4 Proof of Theorem 5.2 . . . 32

6 Public-Key Cryptography Does Not Imply Bounded-TFNP Hardness **33**
- 6.1 Proof Overview . . . 35
- 6.2 $\mathcal{O}_{\mathsf{TDF}}$ is a Collection of Injective Trapdoor Functions . . . 36
- 6.3 Solving Oracle-Aided Bounded-TFNP Instances Relative to $\mathcal{O}_{\mathsf{TDF}}$ . . . 39
- 6.4 Proofs of Claims 6.8–6.10 . . . 43

References **45**

A Average-Case SVL Hardness and OWFs Do Not Imply Key Agreement **48**
- A.1 Proof Overview . . . 49
- A.2 Attacking Key-Agreement Protocols Relative to f and $\mathcal{O}_{\mathsf{SVL}}$ . . . 51

1 Introduction

In recent years there has been increased interest in the computational complexity of finding a Nash equilibrium. Towards this end, Papadimitriou defined the complexity class PPAD, which consists of all TFNP problems that are polynomial-time reducible to the SOURCE-OR-SINK problem [Pap94].[1] Papadimitriou showed that the problem of finding a Nash equilibrium is reducible to SOURCE-OR-SINK, and thus belongs to PPAD. He also conjectured that there exists a reduction in the opposite direction, and this was proved by Daskalakis, Goldberg and Papadimitriou [DGP09], and by Chen, Deng and Teng [CDT09]. Thus, to support the belief that finding a Nash equilibrium may indeed be computationally hard, it became sufficient to place a conjectured computationally-hard problem within the class PPAD.

Currently, no PPAD-complete problem is known to admit a sub-exponential-time algorithm. At the same time, however, we do not know how to generate instances that defeat known heuristics for these problems (see [HPV89] for oracle-based worst-case hard instances of computing Brouwer fixed points and [SvS04] for finding a Nash equilibrium). This leaves us in an intriguing state of affairs, in which we know of no efficient algorithms with provable worst-case guarantees, but we are yet to systematically rule out the possibility that known heuristic algorithms perform well on the average.

"Post-obfuscation" PPAD hardness. A natural approach for arguing hardness on the average would be to reduce from problems that originate from cryptography. Working in the realm of cryptography has at least two advantages. First of all, it enables us to rely on well-studied problems that are widely conjectured to be average-case hard. Secondly, and no less importantly, cryptography supplies us with frameworks for reasoning about average-case hardness. On the positive direction, such frameworks are highly suited for designing and analyzing reductions between average-case problems. On the negative direction, in some cases it is possible to argue that such "natural" reductions do not exist [Rud88, IR89].

Up until recently not much progress has been made in relating between cryptography and PPAD hardness. This has changed as a result of developments in the study of obfuscation [BGI+01, GGH+13], a strong cryptographic notion with connections to the hardness of SOURCE-OR-SINK. As shown by Bitansky, Paneth and Rosen [BPR15] the task of breaking sub-exponentially secure *indistinguishability obfuscation* can be reduced to solving SOURCE-OR-SINK. Beyond giving the first extrinsic evidence of PPAD hardness, the result of Bitansky et al. also provided the first method to sample potentially hard-on-average SOURCE-OR-SINK instances. Their result was subsequently strengthened by Garg, Pandey and Srinivasan, who based it on indistinguishability obfuscation with standard (i.e., polynomial) hardness [GPS16].

"Pre-obfuscation" PPAD hardness? Indistinguishability obfuscation has revealed to be an exceptionally powerful primitive, with numerous far reaching applications. However, its existence is far from being a well-established cryptographic assumption, certainly not nearly as well-established as the existence of one-way functions or public-key encryption. Recently, our confidence in existing indistinguishability obfuscation candidates has somewhat been shaken, following a sequence of devastating attacks on both candidate obfuscators and on their underlying building blocks (see, for example, [BGH+15, CGH+15, CHL+15, CLR15, HJ15, MF15, CFL+16, CJL16, MSZ16]). It thus became natural to ask:

Can average-case PPAD hardness be based on standard cryptographic assumptions?

[1] The name END-OF-LINE is more commonly used in the literature, however SOURCE-OR-SINK is more accurately descriptive [BCE+95].

By standard cryptographic assumptions we are in general referring to "pre-obfuscation" type of primitives, such as the existence of one-way functions or public-key cryptography. As mentioned above, such assumptions are currently by far more well-established than indistinguishability obfuscation, and basing average-case PPAD hardness on them would make a much stronger case.

For all we know PPAD hardness may be based on the existence of one-way functions. However, if it turned out that average-case PPAD hardness implies public-key encryption, then this would indicate that basing average-case PPAD hardness on one-way functions may be extremely challenging since we currently do not know how to base public-key encryption on one-way functions (and in fact cannot do so using black-box techniques [IR89]). Similarly, if it turned out that average-case PPAD hardness implies indistinguishability obfuscation, this would indicate that basing average-case PPAD hardness on any standard cryptographic assumption would require developing radically new techniques. More generally, the stronger the implication of PPAD hardness is, the more difficult it may be to base PPAD hardness on standard assumptions. This leads us to the following second question:

Does average-case PPAD hardness imply any form of cryptographic hardness?

As discussed above, a negative answer to the above question would actually be an encouraging sign. It would suggest, in particular, that program obfuscation is not essential for PPAD hardness, and that there may be hope to base PPAD hardness on standard cryptographic assumptions.

1.1 Our Contributions

Motivated by the above questions, we investigate the interplay between average-case PPAD hardness and standard cryptographic assumptions. We consider this interplay from the perspective of black-box reductions, the fundamental approach for capturing natural relations both among cryptographic primitives (e.g., [Rud88, IR89, Lub96]) and among complexity classes (e.g., [BCE+95, CIY97]).

Average-case PPAD hardness does not imply cryptographic hardness. Our first result shows that average-case PPAD hardness does not imply any form of cryptographic hardness in a black-box manner (not even a one-way function). In addition, our second result shows that, even when assuming the existence of one-way functions, average-case PPAD hardness does not imply any public-key primitive (not even key agreement).[2] In fact, we prove the following more general theorems by considering the SINK-OF-VERIFIABLE-LINE (SVL) problem, introduced by Abbot et al. [AKV04] and further studied by Bitansky et al. [BPR15] and Garg et al. [GPS16]:

Theorem 1.1. *There is no black-box construction of a one-way function from a hard-on-average distribution of SVL instances.*

Theorem 1.2. *There is no black-box construction of a key-agreement protocol from a one-way function and a hard-on-average distribution of SVL instances.*

Abbot et al. [AKV04] and Bitansky et al. [BPR15] showed that any hard-on-average distribution of SVL instances can be used in a black-box manner for constructing a hard-on-average distribution of instances to a PPAD-complete problem (specifically, instances of the SOURCE-OR-SINK problem). Thus, Theorem 1.1 implies, in particular, that there is no black-box construction of a one-way function from a hard-on-average distribution of instances to a PPAD-complete problem. Similarly,

[2] Recall that although indistinguishability obfuscation does not unconditionally imply the existence of one-way functions [BGI+12], it does imply public-key cryptography when assuming the existence of one-way functions [SW14].

Theorem 1.2 implies, in particular, that there is no black-box construction of a key-agreement protocol from a one-way function and a hard-on-average distribution of instances to a PPAD-complete problem.

As discussed in the previous section, the fact that average-case PPAD hardness does not naturally imply any form of cryptographic hardness is an encouraging sign in the pursuit of basing average-case PPAD hardness on standard cryptographic assumptions. For example, if average-case PPAD hardness would have implied program obfuscation, this would have indicated that extremely strong cryptographic assumptions are likely to be essential for average-case PPAD hardness. Similarly, if average-case PPAD hardness would have implied public-key cryptography, this would have indicated that well-structured cryptographic assumptions are essential for average-case PPAD hardness. The fact that average-case PPAD hardness does not naturally imply any form of cryptographic hardness hints that it may be possible to base average-case PPAD hardness even on the minimal (and unstructured) assumption that one-way functions exist.

Note that even if Theorems 1.1 and 1.2 do not hold when considering non-black-box reductions, this still does not rule out the possibility of basing average-case PPAD hardness on one-way functions in a black-box manner – given that the known barriers for constructions based on one-way functions are limited to black-box techniques (e.g., [IR89, Sim98]).

PPAD hardness vs. SVL hardness. The SVL problem played a central role in the recent breakthrough of Bitansky et al. [BPR15] and Garg et al. [GPS16] in constructing a hard-on-average distribution of instances to a PPAD-complete problem based on indistinguishability obfuscation. Specifically, they constructed a hard-on-average distribution of SVL instances, and then reduced it to a hard-on-average distribution of SOURCE-OR-SINK instances [AKV04, BPR15].

We show, however, that the SVL problem is in fact far from representing PPAD hardness: Whereas Abbot et al. [AKV04] and Bitansky et al. [BPR15] showed that the SVL problem can be efficiently reduced to the SOURCE-OR-SINK problem (even in the worst case), we show that there is no such reduction in the opposite direction (not even an average-case one). We prove the following theorem:

Theorem 1.3. *There is no black-box construction of a hard-on-average distribution of SVL instances from a hard-on-average distribution of* SOURCE-OR-SINK *instances. Moreover, this holds even if the underlying* SOURCE-OR-SINK *instances always have a unique solution.*

On basing average-case PPAD hardness on standard assumptions. Theorem 1.1 encouragingly shows that it may still be possible to base average-case PPAD hardness on standard cryptographic assumptions, but Theorem 1.3 shows that the obfuscation-based approach (which goes through the SVL problem) may not be the most effective one. Now, we show that in fact any attempt for basing average-case PPAD hardness on standard cryptographic assumptions (e.g., on one-way functions, public-key encryption, and even on injective trapdoor functions) in a black-box manner must significantly deviate from the obfuscation-based approach. Specifically, the SOURCE-OR-SINK instances resulting from that approach have exactly one solution[3], and we show that when relying on injective trapdoor functions in a black-box manner it is essential to have a nearly-exponential number of solutions. We prove the following theorem:

Theorem 1.4. *There is no black-box construction of a hard-on-average distribution of* SOURCE-OR-SINK *instances over* $\{0,1\}^n$ *with* $2^{n^{o(1)}}$ *solutions from injective trapdoor functions.*

[3]Unless, of course, one allows for artificial manipulations of the instances to generate multiple (strongly related) solutions.

In particular, since Abbot et al. [AKV04] and Bitansky et al. [BPR15] showed that hard-on-average SVL instances lead to hard-on-average SOURCE-OR-SINK instances *having a unique solution*, Theorem 1.4 implies the following corollary which, when combined with Theorem 1.1, shows that average-case SVL hardness is essentially incomparable to standard cryptographic assumptions.

Corollary 1.5. *There is no black-box construction of hard-on-average distribution of SVL instances from injective trapdoor functions.*

More generally, although Theorem 1.4 and Corollary 1.5 focus on injective trapdoor functions, our impossibility result holds for a richer and larger class of building blocks. Specifically, it holds for any primitive that exists relative to a random injective trapdoor function oracle. Thus, Theorem 1.4 and Corollary 1.5 hold, for example, also for collision-resistant hash functions (which are not implied by one-way functions or injective trapdoor functions in a black-box manner [Sim98, HHR$^+$15]).

Taken together, our results imply that it may be possible to base average-case PPAD hardness on standard cryptographic assumptions, but any black-box attempt must significantly deviate from the obfuscation-based approach: It cannot go through the SVL problem, and it must result in SOURCE-OR-SINK instances with a nearly-exponential number of solutions. See Figure 1 for an illustration of our results.

A wider perspective: From Rudich's impossibility to structured building blocks and bounded-TFNP hardness. Our results apply to a wide class of search problems, and not only to the specific SOURCE-OR-SINK and SVL problems. We consider the notion of TFNP instances with a guaranteed (non-trivial) upper bound on their number of existing solutions, to which we refer as *bounded*-TFNP instances. This captures, in particular, SOURCE-OR-SINK instances and (valid) SVL instances, and provides a more general and useful perspective for studying cryptographic limitations in constructing hard instances of search problems.

Equipped with such a wide perspective, our approach and proof techniques build upon, and significantly extend, Rudich's classic proof for ruling out black-box constructions of one-way permutations based on one-way functions [Rud88]. We extend Rudich's approach from its somewhat restricted context of one-way functions (as building blocks) and one-way permutations (as target objects) to provide a richer framework that considers: (1) *significantly more structured* building blocks, and (2) *significantly less restricted* target objects. Specifically, we bound the limitations of hard-on-average SOURCE-OR-SINK and SVL instances as building blocks (instead of one-way functions), and we rule out bounded-TFNP instances as target objects (instead of one-way permutations).

1.2 Open Problems

Several interesting open problems arise directly from our results, and here we point out some of them.

- The strong structural barrier put forward in Theorem 1.4 stands in stark contrast to the approach of Bitansky et al. [BPR15] and Garg et al. [GPS16]. Thus, an intriguing open problem is either to extend our impossibility result to rule out constructions with any number of solutions, or to circumvent our impossibility result by designing instances with an nearly-exponential number of solutions based on standard cryptographic assumptions.
- More generally, the question of circumventing black-box impossibility results by utilizing non-black-box techniques is always fascinating. In our specific context, already the obfuscation-based constructions of Bitansky et al. [BPR15] and Garg et al. [GPS16] involve non-black-box techniques (e.g., they apply an indistinguishability obfuscator to a circuit that uses a pseudo-random function). However, as recently shown by Asharov and Segev [AS15, AS16], as long

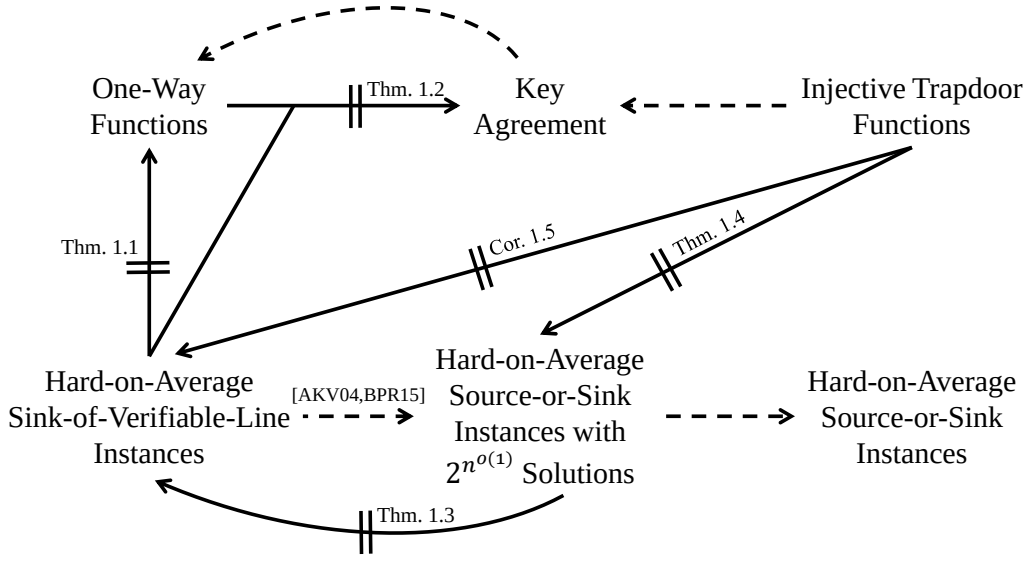

Figure 1: An illustration of our results. Dashed arrows correspond to known implications, and solid arrows correspond to our separations.

as the indistinguishability obfuscator itself is used in a black-box manner, such techniques can in fact be captured by refining the existing frameworks for black-box separations (specifically, the framework of Asharov and Segev captures the obfuscation-based constructions of Bitansky et al. [BPR15] and Garg et al. [GPS16]). Thus, an exciting open problem is to circumvent our results by utilizing non-black-box techniques while relying on standard cryptographic assumptions.

- Our impossibility results in Theorem 1.4 and Corollary 1.5 apply to any building block that exists relative to a random injective trapdoor function oracle (e.g., a collision-resistent hash function). It is not clear, however, whether similar impossibility results may apply to one-way *permutations*. Thus, an intriguing open problem is either to extend our impossibility results to rule out constructions based on one-way permutations, or to circumvent our impossibility results by designing hard-on-average instances based on one-way permutations. We note that by relying on one-way permutations it is rather trivial to construct some arbitrary hard-on-average TFNP distribution (even one with unique solutions), but it is not known how to construct less arbitrary forms of hardness, such as average-case PPAD or SVL hardness.
- The recent work of Hubácek, Naor, and Yogev [HNY17] proposes two elegant approaches for constructing hard-on-average TFNP instances. Their first approach is based on any hard-on-average NP relation (the existence of which is implied, for example, by any one-way function) in a black-box manner, and results in TFNP instances with a possibly exponential number of solutions. Their second approach is based on any injective one-way function and a non-interactive witness-indistinguishable proof system for NP (which can be constructed based on trapdoor permutations), and results in TFNP instances having at most two solutions. An interesting question is whether their approaches imply not only average-case TFNP hardness for the particular problems defined by their underlying one-way function and proof system, but also more specific forms of TFNP hardness, such as average-case PPAD or SVL hardness.

1.3 Overview of Our Approach

In this section we provide a high-level overview of the main ideas underlying our results. Each of our results is of the form "the existence of P does not imply the existence of Q in a black-box manner", where each of P and Q is either a cryptographic primitive (e.g., a one-way function) or a hard-on-average search problem (e.g., the source-or-sink problem). Intuitively, such a statement is proved by constructing a distribution over oracles relative to which there exists an implementation of P, but any implementation of Q can be "efficiently broken". Our formal proofs properly formalize this intuition via the standard framework of fully black-box reductions (e.g., [IR89, Lub96, Gol00, RTV04]), where for our purposes it suffices to measure the efficiency of an attacker via its number of queries to the relevant oracles. Moreover, we show that when our attackers are given access to an oracle that decides any PSPACE language, then we can also bound the amount of internal computation performed by our attackers, thus strengthening our result to rule out construction that require polynomial time efficiency.

Average-case SVL hardness does not imply OWFs. Theorem 1.1 is proved by presenting a distribution of oracles relative to which there exists a hard-on-average distribution of SVL instances, but there are no one-way functions. An SVL instance is of the form $\{(\mathsf{S}_n, \mathsf{V}_n, L(n))\}_{n\in\mathbb{N}}$, where for every $n \in \mathbb{N}$ it holds that $\mathsf{S}_n : \{0,1\}^n \to \{0,1\}^n$, $\mathsf{V}_n : \{0,1\}^n \times [2^n] \to \{0,1\}$, and $L(n) \in [2^n]$. Such an instance is *valid* if for every $n \in \mathbb{N}$, $x \in \{0,1\}^n$, and $i \in [2^n]$, it holds that $\mathsf{V}_n(x,i) = 1$ if and only if $x = \mathsf{S}_n^i(0^n)$. Intuitively, the circuit S_n can be viewed as implementing the successor function of a directed graph over $\{0,1\}^n$ that consists of a single line starting at 0^n, and the circuit V_n enables to efficiently test whether a given node x is of distance i from 0^n on the line. The goal is to find the node of distance $L(n)$ from 0^n (see Section 2.1 for the formal definition of the SVL problem).

We consider an oracle that is a valid SVL instance $\mathcal{O}_{\mathsf{SVL}}$ corresponding to a graph with a single line $0^n \to x_1 \to \cdots \to x_{L(n)}$ of length $L(n) = 2^{n/2}$. The line is chosen uniformly among all lines in $\{0,1\}^n$ of length $L(n)$ starting at 0^n (and all nodes outside the line have self loops and are essentially irrelevant). First, we show that the oracle $\mathcal{O}_{\mathsf{SVL}}$ is indeed a hard-on-average SVL instance. This is based on the following, rather intuitive, observation: Since the line $0^n \to x_1 \to \cdots \to x_{L(n)}$ is *sparse* and *uniformly sampled*, then any algorithm performing $q = q(n)$ oracle queries should not be able to query $\mathcal{O}_{\mathsf{SVL}}$ with any element on the line beyond the first q elements $0^n, x_1, \ldots, x_{q-1}$. In particular, for our choice of parameters, any algorithm performing at most, say, $2^{n/4}$ queries, has only an exponentially-small probability of reaching $x_{L(n)}$ (where the probability is taken over the choice of the oracle $\mathcal{O}_{\mathsf{SVL}}$).

Then, we show that any oracle-aided function $F^{\mathcal{O}_{\mathsf{SVL}}}(\cdot)$ can be inverted (with high probability over the choice of the oracle $\mathcal{O}_{\mathsf{SVL}}$) by an algorithm whose query complexity is polynomially-related to that of the function $F^{\mathcal{O}_{\mathsf{SVL}}}(\cdot)$. The proof is based on the following approach. Consider a value $y = F^{\mathcal{O}_{\mathsf{SVL}}}(x)$ that we would like to invert. If F performs at most $q = q(n)$ oracle queries, the above-mentioned observation implies that the computation $F^{\mathcal{O}_{\mathsf{SVL}}}(x)$ should not query $\mathcal{O}_{\mathsf{SVL}}$ with any elements on the line $0^n \to x_1 \to \cdots \to x_{L(n)}$ except for the first q elements $x_0, x_1, \ldots, x_{q-1}$. This observation gives rise to the following inverter $\mathcal{A}$: First perform q queries to $\mathcal{O}_{\mathsf{SVL}}$ for discovering $x_1, \ldots, x_q$, and then invert $y = F^{\mathcal{O}_{\mathsf{SVL}}}(x)$ relative to the oracle $\widetilde{\mathcal{O}_{\mathsf{SVL}}}$ defined via the following successor function $\widetilde{\mathsf{S}}$:

$$\widetilde{\mathsf{S}}(\alpha) = \begin{cases} x_{i+1} & \text{if } \alpha = x_i \text{ for some } i \in \{0, \ldots, q-1\} \\ \alpha & \text{otherwise} \end{cases} .$$

The formal proof is in fact more subtle, and requires a significant amount of caution when inverting $y = F^{\mathcal{O}_{\mathsf{SVL}}}(x)$ relative to the oracle $\widetilde{\mathcal{O}_{\mathsf{SVL}}}$. Specifically, the inverter $\mathcal{A}$ should find an input $\widetilde{x}$ such that the computations $F^{\widetilde{\mathcal{O}_{\mathsf{SVL}}}}(\widetilde{x})$ and $F^{\mathcal{O}_{\mathsf{SVL}}}(\widetilde{x})$ do not query the oracles $\widetilde{\mathcal{O}_{\mathsf{SVL}}}$ and $\mathcal{O}_{\mathsf{SVL}}$, respectively, with

any of $x_q, \ldots, x_{L(n)}$. In this case, we show that indeed $F^{\mathcal{O}_{\mathsf{SVL}}}(\widetilde{x}) = y$ and the inverter is successful. We refer the reader to Section 3 for more details and for the formal proof.

Average-case SVL hardness and OWFs do not imply key agreement. Theorem 1.2 is proved by showing that in any black-box construction of a key-agreement protocol based on a one-way function and a hard-on-average distribution of SVL instances, we can eliminate the protocol's need for using the SVL instances. This leads to a black-box construction of key-agreement protocol based on a one-way function, which we can then rule out by invoking the classic result of Impagliazzo and Rudich [IR89] and its refinement by Barak and Mahmoody-Ghidary [BM09].

Specifically, consider a key-agreement protocol $(\mathcal{A}^{f,\mathcal{O}_{\mathsf{SVL}}}, \mathcal{B}^{f,\mathcal{O}_{\mathsf{SVL}}})$ in which the parties have oracle access to a random function f and to the oracle $\mathcal{O}_{\mathsf{SVL}}$ used for proving Theorem 1.1. Then, if $\mathcal{A}$ and $\mathcal{B}$ perform at most $q = q(n)$ oracle queries, the observation underlying the proof of Theorem 1.1 implies that, during an execution $(\mathcal{A}^{f,\mathcal{O}_{\mathsf{SVL}}}, \mathcal{B}^{f,\mathcal{O}_{\mathsf{SVL}}})$ of the protocol, the parties should not query $\mathcal{O}_{\mathsf{SVL}}$ with any elements on the line $0^n \to x_1 \to \cdots \to x_{L(n)}$ except for the first q elements $x_0, x_1, \ldots, x_{q-1}$. This observation gives rise to a key-agreement protocol $(\widetilde{\mathcal{A}}^f, \widetilde{\mathcal{B}}^f)$ that does not require access to the oracle $\mathcal{O}_{\mathsf{SVL}}$: First, $\widetilde{A}$ samples a sequence $x_1, \ldots, x_q$ of q values, and sends these values to $\widetilde{B}$. Then, $\widetilde{\mathcal{A}}$ and $\widetilde{\mathcal{B}}$ run the protocol $(\mathcal{A}^{f,\mathcal{O}_{\mathsf{SVL}}}, \mathcal{B}^{f,\mathcal{O}_{\mathsf{SVL}}})$ by using the values $x_1, \ldots, x_q$ instead of accessing $\mathcal{O}_{\mathsf{SVL}}$. That is, $\widetilde{A}$ and $\widetilde{B}$ run the underlying protocol relative to the given oracle f and to the oracle $\widetilde{\mathcal{O}_{\mathsf{SVL}}}$ defined via the following successor function $\widetilde{\mathsf{S}}$ (which each party can compute on its own):

$$\widetilde{\mathsf{S}}(\alpha) = \begin{cases} x_{i+1} & \text{if } \alpha = x_i \text{ for some } i \in \{0, \ldots, q-1\} \\ \alpha & \text{otherwise} \end{cases} .$$

The formal proof is again rather subtle, and we refer the reader to Appendix A for more details and for the formal proof.

Average-case PPAD hardness does not imply unique-TFNP hardness. Theorem 1.3 is proved by presenting a distribution of oracles relative to which there exists a hard-on-average distribution of instances of a PPAD-complete problem (specifically, we consider the source-or-sink problem), but there are no hard TFNP instances having unique solutions.

A TFNP instance with a unique solution, denoted a unique-TFNP instance, is of the form $\{C_n\}_{n\in\mathbb{N}}$, where for every $n \in \mathbb{N}$ it holds that $C_n : \{0,1\}^n \to \{0,1\}$ and there is a unique $x^* \in \{0,1\}^n$ such that $C(x^*) = 1$. Note that any *valid* SVL instance yields a TFNP instance that has a unique solution. Therefore, relative to our distribution over oracles any valid SVL instance can be efficiently solved.

A source-or-sink instance is of the form $\{(\mathsf{S}_n, \mathsf{P}_n)\}_{n\in\mathbb{N}}$, where for every $n \in \mathbb{N}$ it holds that $\mathsf{S}_n : \{0,1\}^n \to \{0,1\}^n$ and $\mathsf{P}_n : \{0,1\}^n \to \{0,1\}^n$. Intuitively, the circuits S_n and P_n can be viewed as implementing the successor and predecessor functions of a directed graph over $\{0,1\}^n$, where the in-degree and out-degree of every node is at most one, and the in-degree of 0^n is 0 (i.e., it is a source). The goal is to find any node, other than 0^n, with either no incoming edge and no outgoing edge. We again refer the reader to Section 2.1 for the formal definitions.

We consider an oracle that is a source-or-sink instance $\mathcal{O}_{\mathsf{PPAD}}$ which is based on the same sparse structure used to define the oracle $\mathcal{O}_{\mathsf{SVL}}$. It corresponds to a graph with a single line $0^n \to x_1 \to \cdots \to x_{L(n)}$ of length $L(n) = 2^{n/2}$. The line is chosen uniformly among all lines in $\{0,1\}^n$ of length $L(n)$ starting at 0^n (and all nodes outside the line have self loops). The fact that the oracle $\mathcal{O}_{\mathsf{PPAD}}$ is a hard-on-average source-or-sink instance follows quite easily from the above-mentioned observation on its sparse and uniform structure: Any algorithm performing $q = q(n)$ oracle queries should not be able to query $\mathcal{O}_{\mathsf{PPAD}}$ with any element on the line beyond the first q elements $x_0, x_1, \ldots, x_{q-1}$. In particular, for our choice of parameters, any such algorithm should have only an exponentially-small probability of reaching $x_{L(n)}$.

Solving any oracle-aided unique-TFNP instance relative to $\mathcal{O}_{\mathsf{PPAD}}$, however, turns out to be a completely different challenge. One might be tempted to follow a same approach based on the oracle's sparse and uniform structure. Specifically, let C_n be a unique-TFNP instance, and consider the unique value $x^* \in \{0,1\}^n$ for which $C_n^{\mathcal{O}_{\mathsf{PPAD}}}(x^*) = 1$. Then, if C_n issues at most $q = q(n)$ oracle queries, the computation $C_n^{\mathcal{O}_{\mathsf{PPAD}}}(x^*)$ should essentially not be able to query $\mathcal{O}_{\mathsf{PPAD}}$ with any elements on the line $0^n \to x_1 \to \cdots \to x_{L(n)}$ except for the first q elements $0^n, x_1, \ldots, x_{q-1}$. Therefore, one can define a "fake" oracle $\widetilde{\mathcal{O}_{\mathsf{PPAD}}}$ whose successor and predecessor functions agree with $\mathcal{O}_{\mathsf{PPAD}}$ on $0^n, x_1, \ldots, x_q$ (and are defined as the identity functions for all other inputs), and then find the unique $\widetilde{x}$ such that $C_n^{\widetilde{\mathcal{O}_{\mathsf{PPAD}}}}(\widetilde{x}) = 1$. This approach, however, completely fails since the solution x^* itself may depend on $\mathcal{O}_{\mathsf{PPAD}}$ in an arbitrary manner, providing the computation $C_n^{\mathcal{O}_{\mathsf{PPAD}}}(x^*)$ with sufficient information for querying $\mathcal{O}_{\mathsf{PPAD}}$ with an input x_i that is located further along the line (i.e., $q \leq i \leq L(n)$).

As discussed in Section 1.1, our proof is obtained by significantly extending Rudich's classic proof for ruling out black-box constructions of one-way permutations based on one-way functions [Rud88]. Here, we show that his approach provides a rich framework that allows to bound not only the limitations of one-way functions as a building block, but even the limitations of *significantly more structured* primitives as building blocks. Specifically, our proof of Theorem 1.3 generalizes Rudich's technique for bounding the limitations of hard-on-average source-or-sink instances. We refer the reader to Section 4 for more details and for the formal proof.

Injective trapdoor functions do not imply bounded-TFNP hardness. Theorem 1.4 and Corollary 1.5 are proved by presenting a distribution of oracles relative to which there exists a collection of injective trapdoor functions, but there are no hard TFNP instances having a bounded number of solutions (specifically, our result will apply to a sub-exponential number of solutions).

A TFNP instance with bounded number $k(\cdot)$ of solutions, denoted a k-bounded TFNP instance, is of the form $\{C_n\}_{n\in\mathbb{N}}$, where for every $n \in \mathbb{N}$ it holds that $C : \{0,1\}^n \to \{0,1\}$, and there is at least one and at most $k(n)$ distinct inputs $x \in \{0,1\}^n$ such that $C(x) = 1$ (any one of these x's is a solution). In particular, as discussed above, any *valid* SVL instance yields a 1-bounded TFNP instance (i.e., a unique-TFNP instance), and therefore our result rules out black-box constructions of a hard-on-average distribution of SVL instances from injective trapdoor functions. Similarly, any source-or-sink instance which consists of at most $(k+1)/2$ disjoint lines yields a k-bounded TFNP instance, and therefore our result rules out black-box constructions of a hard-on-average distribution of source-or-sink instances with a bounded number of disjoint lines from injective trapdoor functions.

For emphasizing the main ideas underlying our proof, in Section 5 we first prove our result for constructions that are based on one-way functions, and then in Section 6 we generalize the proof to constructions that are based on injective trapdoor functions. Each of these two parts requires introducing new ideas and techniques, and such a level of modularity is useful in pointing them out.

When considering constructions that are based on one-way functions, our proof is obtained via an additional generalization of Rudich's proof technique [Rud88]. As discussed above, we first observe that Rudich's approach can be generalized from ruling out constructions of one-way permutations based on one-way functions to ruling out constructions of any hard-on-average distribution of unique-TFNP instances based on one-way functions. Then, by extending and refining Rudich's proof technique once again, we show that we can rule out not only constructions of unique-TFNP instances, but even constructions of bounded-TFNP instances. This require a substantial generalization of Rudich's attacker, and we refer reader to Section 5 for more details and for the formal proof.

Then, when considering constructions that are based on injective trapdoor functions, we show that our proof from Section 5 can be generalized from constructions of bounded-TFNP instances

based on one-way functions to constructions of bounded-TFNP instances based on injective trapdoor functions. Combined with our the proof of Theorem 1.3, this extends Rudich's approach from its somewhat restricted context of one-way functions (as building blocks) and one-way permutations (as target objects) to provide a richer framework that considers: (1) *significantly more structured* building blocks, and (2) *significantly less restricted* target objects. We refer reader to Section 6 for more details and for the formal proof.

1.4 Paper Organization

The remainder of this paper is organized as follows. In Section 2 we introduce our notation as well as the search problems and the cryptographic primitives that we consider in this paper. In Section 3 we show that average-case SVL hardness does not imply one-way functions in a black-box manner (proving Theorem 1.1). In Section 4 we show that average-case PPAD hardness does not imply unique-TFNP hardness in a black-box manner (proving Theorem 1.3). In Section 5 we show that one-way functions do not imply bounded-TFNP hardness in a black-box manner, and in Section 6 we generalize this result, showing that even injective trapdoor functions do not imply bounded-TFNP hardness in a black-box manner (proving Theorem 1.4 and Corollary 1.5). Finally, in Appendix A we extend our approach from Section 3 and show that average-case SVL hardness does not imply key agreement even when assuming the existence of one-way functions.

2 Preliminaries

In this section we present the notation and basic definitions that are used in this work. For a distribution X (e.g., the output distribution of a randomized algorithm when given a certain input) we denote by $x \leftarrow X$ the process of sampling a value x from the distribution X. Similarly, for a set $\mathcal{X}$ we denote by $x \leftarrow \mathcal{X}$ the process of sampling a value x from the uniform distribution over $\mathcal{X}$. For an integer $n \in \mathbb{N}$ we denote by $[n]$ the set $\{1, \ldots, n\}$. A *q-query algorithm* is an oracle-aided algorithm $\mathcal{A}$ such that for any oracle $\mathcal{O}$ and input $x \in \{0,1\}^*$, the computation $\mathcal{A}^{\mathcal{O}}(x)$ consists of at most $q(|x|)$ oracle calls to $\mathcal{O}$. An *oracle-aided circuit* C is a circuit equipped with additional *oracle gates*, where the input to an oracle gate is a query and the output is the answer of the oracle for that query. We denote by $C^{\mathcal{O}}(x)$ the result of computing the oracle-aided circuit C on input $x \in \{0,1\}^*$ with respect to the oracle $\mathcal{O}$.

2.1 Complexity Classes and Total Search Problems

An efficiently-verifiable search problem is described via a pair (I, R), where $I \subseteq \{0,1\}^*$ is an efficiently-recognizable set of instances, and R is an efficiently-computable binary relation. Such a search problem is *total* if for every instance $z \in I$ there exists a witness w of length polynomial in the length z such that $R(z, w) = 1$.

The class TFNP consists of all efficiently-verifiable search problem that are total, and its subclass PPAD consists of all such problems that are polynomial-time reducible to the source-or-sink problem [Pap94], defined as follows.

Definition 2.1 (The source-or-sink problem)**.** A source-or-sink instance consists of a pair of circuits $\mathsf{S}, \mathsf{P} : \{0,1\}^n \to \{0,1\}^n$ such that $\mathsf{P}(0^n) = 0^n \neq \mathsf{S}(0^n)$. The goal is to find an element $w \in \{0,1\}^n$ such that $\mathsf{P}(\mathsf{S}(w)) \neq w$ or $\mathsf{S}(\mathsf{P}(w)) \neq w \neq 0^n$.

Intuitively, the circuits S and P can be viewed as implementing the successor and predecessor functions of a directed graph over $\{0,1\}^n$, where for each pair of nodes x and y there exists an edge

from x to y if and only if $\mathsf{S}(x) = y$ and $\mathsf{P}(y) = x$ (note that the in-degree and out-degree of every node in this graph is at most one, and the in-degree of 0^n is 0). The goal is to find any node, other than 0^n, with either no incoming edge or no outgoing edge. Such a node must always exist by a parity argument.

The sink-of-verifiable-line (SVL) problem is a search problem introduced by Abbot et al. [AKV04] and further studied by Bitansky et al. [BPR15] and Garg et al. [GPS16]. It is defined as follows:

Definition 2.2 (The sink-of-verifiable-line (SVL) problem)**.** An SVL instance consists of a triplet $(\mathsf{S}, \mathsf{V}, T)$, where $T \in [2^n]$, and $\mathsf{S} : \{0,1\}^n \to \{0,1\}^n$ and $\mathsf{V} : \{0,1\}^n \times [2^n] \to \{0,1\}$ are two circuits with the guarantee that for every $x \in \{0,1\}^n$ and $i \in [2^n]$ it holds that $\mathsf{V}(x,i) = 1$ if and only if $x = \mathsf{S}^i(0^n)$. The goal is to find an element $w \in \{0,1\}^n$ such that $\mathsf{V}(w,T) = 1$.

Intuitively, the circuit S can be viewed as implementing the successor function of a directed graph over $\{0,1\}^n$ that consists of a single line starting at 0^n. The circuit V enables to efficiently test whether a given node x is of distance i from 0^n on the line, and the goal is to find the node of distance T from 0^n. Note that not any triplet $(\mathsf{S}, \mathsf{V}, T)$ is a *valid* SVL instance (moreover, there may not be an efficient algorithm for verifying whether a triplet $(\mathsf{S}, \mathsf{V}, T)$ is a valid instance).

Oracle-aided instances with private randomness. We consider source-or-sink and SVL instances that are described by oracle-aided circuits, and we would like to allow these circuits to share an oracle-dependent state that may be generated via private randomness (this clearly strengthens the class of problems that we consider, and in particular, capture those constructed by [BPR15, GPS16] using indistinguishability obfuscation). For this purpose, we equip the instances with an oracle-aided randomized index-generation algorithm, denoted Gen, that produces a public index σ which is then provided to all circuits of the instance (and to any algorithm that attempts to solve the instance).

Specifically, we consider source-or-sink instances of the form $\{(\mathsf{Gen}_n, \mathsf{S}_n, \mathsf{P}_n)\}_{n\in\mathbb{N}}$, where for every $n \in \mathbb{N}$ and for every index σ produced by Gen_n it holds that $\mathsf{S}_n(\sigma, \cdot) : \{0,1\}^n \to \{0,1\}^n$ and $\mathsf{P}_n(\sigma, \cdot) : \{0,1\}^n \to \{0,1\}^n$. Similarly, we consider SVL instances of the form $\{(\mathsf{Gen}_n, \mathsf{S}_n, \mathsf{V}_n, T(n))\}_{n\in\mathbb{N}}$, where for every $n \in \mathbb{N}$ and for every index σ produced by Gen_n it holds that $\mathsf{S}_n(\sigma, \cdot) : \{0,1\}^n \to \{0,1\}^n$, $\mathsf{V}_n(\sigma, \cdot, \cdot) : \{0,1\}^n \times [2^n] \to \{0,1\}$, and $T(n) \in [2^n]$. We say that an SVL instance is *valid* if for every $n \in \mathbb{N}$, σ produced by Gen_n, $x \in \{0,1\}^n$, and $i \in [2^n]$, it holds that $\mathsf{V}_n(\sigma, x, i) = 1$ if and only if $x = \mathsf{S}_n^i(\sigma, 0^n)$.

Bounded TFNP instances. As discussed in Section 1.1, we prove our results using the notion of *bounded*-TFNP instances, naturally generalizing source-or-sink instances (and valid SVL instances) by considering TFNP instances with a guaranteed upper bound on the number of solutions.

Definition 2.3. A k-bounded TFNP instance is of the form $\{\mathsf{Gen}_n, C_n\}_{n\in\mathbb{N}}$, where for every $n \in \mathbb{N}$ and for every index σ produced by Gen_n it holds that $C_n(\sigma, \cdot) : \{0,1\}^n \to \{0,1\}$, and there is at least one and at most $k(n)$ distinct inputs $x \in \{0,1\}^n$ such that $C_n(\sigma, x) = 1$ (any one of these x's is a solution).

Note that any *valid* SVL instance yields a 1-bounded TFNP instance (to which we refer as a *unique*-TFNP instance), and any source-or-sink instance which consists of at most $(k+1)/2$ disjoint lines yields a k-bounded TFNP instance.

Average-case PPAD hardness and bounded-TFNP hardness. The following two definitions formalize the standard notion of average-case hardness in the specific context of source-or-sink instances and k-bounded TFNP instances. These notions then serve as the basis of our definitions of black-box constructions.

Definition 2.4. Let $t = t(n)$ and $\epsilon = \epsilon(n)$ be functions of the security parameter $n \in \mathbb{N}$. A source-or-sink instance $\{(\mathsf{Gen}_n, \mathsf{S}_n, \mathsf{P}_n)\}_{n\in\mathbb{N}}$ is (t, ϵ)-hard if for any algorithm $\mathcal{A}$ that runs in time $t(n)$ it holds that

$$\Pr\left[\mathcal{A}\left(1^n, \sigma\right) = w \text{ s.t. } \mathsf{P}_n(\sigma, \mathsf{S}_n(\sigma, w)) \neq w \text{ or } \mathsf{S}_n(\sigma, \mathsf{P}_n(\sigma, w)) \neq w \neq 0^n\right] \leq \epsilon(n)$$

for all sufficiently large $n \in \mathbb{N}$, where the probability is taken over the choice of $\sigma \leftarrow \mathsf{Gen}_n()$ and over the internal randomness of $\mathcal{A}$.

Definition 2.5. Let $k = k(n)$, $t = t(n)$ and $\epsilon = \epsilon(n)$ be functions of the security parameter $n \in \mathbb{N}$. A k-bounded TFNP instance $\{\mathsf{Gen}_n, C_n\}_{n\in\mathbb{N}}$ is (t, ϵ)-hard if for any algorithm $\mathcal{A}$ that runs in time $t(n)$ it holds that

$$\Pr\left[\mathcal{A}\left(1^n, \sigma\right) = x \text{ s.t. } C_n(\sigma, x) = 1\right] \leq \epsilon(n)$$

for all sufficiently large $n \in \mathbb{N}$, where the probability is taken over the choice of $\sigma \leftarrow \mathsf{Gen}_n()$ and over the internal randomness of $\mathcal{A}$.

2.2 One-Way Functions and Injective Trapdoor Functions

We rely on the standard (parameterized) notions of a one-way function and injective trapdoor functions [Gol01].

Definition 2.6. An efficiently-computable function $f : \{0,1\}^* \to \{0,1\}^*$ is $(t(\cdot), \epsilon(\cdot))$-one-way if for any probabilistic algorithm $\mathcal{A}$ that runs in time $t(n)$ it holds that

$$\Pr\left[\mathcal{A}\left(f(x)\right) \in f^{-1}\left(f(x)\right)\right] \leq \epsilon(n)$$

for all sufficiently large $n \in \mathbb{N}$, where the probability is taken over the choice of $x \leftarrow \{0,1\}^n$ and over the internal randomness of $\mathcal{A}$.

A collection of injective trapdoor functions is a triplet $(\mathsf{KG}, \mathsf{F}, \mathsf{F}^{-1})$ of polynomial-time algorithms. The key-generation algorithm KG is a probabilistic algorithm that on input the security parameter 1^n outputs a pair $(\mathsf{pk}, \mathsf{td})$, where pk is a public key and td is a corresponding trapdoor. For any $n \in \mathbb{N}$ and for any pair $(\mathsf{pk}, \mathsf{td})$ that is produced by $\mathsf{KG}(1^n)$, the evaluation algorithm F computes an injective function $\mathsf{F}(\mathsf{pk}, \cdot) : \{0,1\}^n \to \{0,1\}^{\upsilon(n)}$, and the inversion algorithm $F^{-1}(\mathsf{td}, \cdot) : \{0,1\}^{\upsilon(n)} \to \{0,1\}^n \cup \{\perp\}$ computes its inverse whenever an inverse exists (i.e., it outputs $\perp$ on all values y that are not in the image of the function $\mathsf{F}(\mathsf{pk}, \cdot)$). The security requirement of injective trapdoor functions is formalized as follows:

Definition 2.7. A collection of injective trapdoor functions $(\mathsf{KG}, \mathsf{F}, \mathsf{F}^{-1})$ is $(t(\cdot), \epsilon(\cdot))$-secure if for any probabilistic algorithm $\mathcal{A}$ that runs in time $t(n)$ it holds that

$$\Pr\left[\mathcal{A}\left(\mathsf{pk}, \mathsf{F}(\mathsf{pk}, x)\right) = x\right] \leq \epsilon(n)$$

for all sufficiently large $n \in \mathbb{N}$, where the probability is taken over the choice of $(\mathsf{pk}, \mathsf{td}) \leftarrow \mathsf{KG}(1^n)$, $x \leftarrow \{0,1\}^n$, and over the internal randomness of $\mathcal{A}$.

2.3 Key-Agreement Protocols

We rely on the standard (parameterized) notion of a key-agreement protocol. For our purposes in this paper it suffices to consider key-agreement protocols in which the parties agree on a single bit, and we refer to such protocols as bit-agreement protocols.

A bit-agreement protocol consists of a pair $(\mathcal{A}, \mathcal{B})$ of probabilistic polynomial-time algorithms. We denote by $(\mathsf{k}_{\mathcal{A}}, \mathsf{k}_{\mathcal{B}}, \mathsf{Trans}) \leftarrow \langle \mathcal{A}(1^n; r_{\mathcal{A}}), \mathcal{B}(1^n; r_{\mathcal{B}}) \rangle$ the random process of executing the protocol, where $r_{\mathcal{A}}$ and $r_{\mathcal{B}}$ are the random tapes of $\mathcal{A}$ and $\mathcal{B}$, respectively, $\mathsf{k}_{\mathcal{A}}$ and $\mathsf{k}_{\mathcal{B}}$ are the output bits of $\mathcal{A}$ and $\mathcal{B}$, respectively, and Trans is the transcript of the protocol (i.e., the messages exchanged by the parties).

Definition 2.8. A pair $\Pi = (\mathcal{A}, \mathcal{B})$ of probabilistic polynomial-time algorithms is a $(t(\cdot), \epsilon(\cdot))$-secure bit-agreement protocol with correctness $\rho(\cdot)$ if the following two conditions hold:

- **Correctness.** For any $n \in \mathbb{N}$ it holds that

$$\Pr_{r_{\mathcal{A}}, r_{\mathcal{B}}} \left[\mathsf{k}_{\mathcal{A}} = \mathsf{k}_{\mathcal{B}} \,|\, (\mathsf{k}_{\mathcal{A}}, \mathsf{k}_{\mathcal{B}}, \mathsf{Trans}) \leftarrow \langle \mathcal{A}(1^n; r_{\mathcal{A}}), \mathcal{B}(1^n; r_{\mathcal{B}}) \rangle \right] \geq \frac{1}{2} + \rho(n).$$

- **Security.** For any probabilistic algorithm E that runs in time $t(n)$ it holds that

$$\mathsf{Adv}^{\mathsf{KA}}_{\Pi,E}(n) \stackrel{\mathsf{def}}{=} \left| \Pr\left[\mathsf{Exp}^{\mathsf{KA}}_{\Pi,E}(n) = 1 \right] - \frac{1}{2} \right| \leq \epsilon(n)$$

  for all sufficiently large $n \in \mathbb{N}$, where the random variable $\mathsf{Exp}^{\mathsf{KA}}_{\Pi,E}(n)$ is defined via the following experiment:

  1. $(\mathsf{k}_{\mathcal{A}}, \mathsf{k}_{\mathcal{B}}, \mathsf{Trans}) \leftarrow \langle \mathcal{A}(1^n), \mathcal{B}(1^n) \rangle$.
  2. $\mathsf{k}' \leftarrow E(1^n, \mathsf{Trans})$.
  3. If $\mathsf{k}' = \mathsf{k}_{\mathcal{A}}$ then output 1, and otherwise output 0.

3 Average-Case SVL Hardness Does Not Imply One-Way Functions

In this section we prove that there is no fully black-box construction of a one-way function from a hard-on-average distribution of SVL instances[4] (proving Theorem 1.1). Our result is obtained by presenting a distribution of oracles relative to which the following two properties hold:

1. There exists a hard-on-average distribution of SVL instances.
2. There are no one-way functions.

Recall that an SVL instance is of the form $\{(\mathsf{Gen}_n, \mathsf{S}_n, \mathsf{V}_n, L(n))\}_{n \in \mathbb{N}}$, where for every $n \in \mathbb{N}$ and for every index σ produced by Gen_n it holds that $\mathsf{S}_n(\sigma, \cdot) : \{0,1\}^n \to \{0,1\}^n$, $\mathsf{V}_n(\sigma, \cdot, \cdot) : \{0,1\}^n \times [2^n] \to \{0,1\}$, and $L(n) \in [2^n]$. We say that an SVL instance is *valid* if for every $n \in \mathbb{N}$, σ produced by Gen_n, $x \in \{0,1\}^n$, and $i \in [2^n]$, it holds that $\mathsf{V}_n(\sigma, x, i) = 1$ if and only if $x = \mathsf{S}^i_n(\sigma, 0^n)$. The following definition tailors the standard notion of a fully black-box construction (based, for example, on [Lub96, Gol00, RTV04]) to the specific primitives under consideration.

Definition 3.1. A fully black-box construction of a one-way function from a hard-on-average distribution of SVL instances consists of an oracle-aided polynomial-time algorithm F, an oracle-aided algorithm M that runs in time $T_M(\cdot)$, and functions $\epsilon_{M,1}(\cdot)$ and $\epsilon_{M,2}(\cdot)$, such that the following conditions hold:

- **Correctness**: There exists a polynomial $\ell(\cdot)$ such that for any valid SVL instance $\mathcal{O}_{\mathsf{SVL}}$ and for any $x \in \{0,1\}^*$ it holds that $F^{\mathcal{O}_{\mathsf{SVL}}}(x) \in \{0,1\}^{\ell(|x|)}$.

[4] Recall that any hard-on-average distribution of SVL instances can be used in a black-box manner to construct a hard-on-average distribution of instances of a PPAD-complete problem [AKV04, BPR15]. Thus, our result implies (in particular) that average-case PPAD hardness does not imply one-way functions in a black-box manner.

- **Black-box proof of security:** For any valid SVL instance $\mathcal{O}_{\mathsf{SVL}} = \{(\mathsf{Gen}_n, \mathsf{S}_n, \mathsf{V}_n, L(n))\}_{n\in\mathbb{N}}$, for any oracle-aided algorithm $\mathcal{A}$ that runs in time $T_{\mathcal{A}} = T_{\mathcal{A}}(n)$, and for any function $\epsilon_{\mathcal{A}}(\cdot)$, if

$$\Pr\left[\mathcal{A}^{\mathcal{O}_{\mathsf{SVL}}}\left(F^{\mathcal{O}_{\mathsf{SVL}}}(x)\right) \in \left(F^{\mathcal{O}_{\mathsf{SVL}}}\right)^{-1}\left(F^{\mathcal{O}_{\mathsf{SVL}}}(x)\right)\right] \geq \epsilon_{\mathcal{A}}(n)$$

  for infinitely many values of $n \in \mathbb{N}$, where the probability is taken over the choice of $x \leftarrow \{0,1\}^n$ and over the internal randomness of $\mathcal{A}$, then

$$\Pr\left[M^{\mathcal{A},\mathcal{O}_{\mathsf{SVL}}}\left(1^n, \sigma\right) \text{ solves } \left(\mathsf{S}_n(\sigma,\cdot), \mathsf{V}_n(\sigma,\cdot), L(n)\right)\right] \geq \epsilon_{M,1}\left(T_{\mathcal{A}}(n)/\epsilon_{\mathcal{A}}(n)\right) \cdot \epsilon_{M,2}(n)$$

  for infinitely many values of $n \in \mathbb{N}$, where the probability is taken over the choice of $\sigma \leftarrow \mathsf{Gen}_n()$ and over the internal randomness of M.

Following Asharov and Segev [AS15, AS16], we split the security loss in the above definition to an adversary-dependent security loss and an adversary-independent security loss, as this allows us to capture constructions where one of these losses is super-polynomial whereas the other is polynomial (e.g., [BPR15, BPW16]). In addition, we note that the correctness requirement in the above definition may seem somewhat trivial since the fact that the output length of $F^{\mathcal{O}_{\mathsf{SVL}}}(\cdot)$ is polynomial follows directly from the requirement that F runs in polynomial time. However, for avoiding rather trivial technical complications in the proofs of this section, for simplicity (and without loss of generality) we nevertheless ask explicitly that the output length is some fixed polynomial $\ell(n)$ for any input length n (clearly, $\ell(n)$ may depend on the running time of F, and shorter outputs can always be padded). Equipped with the above definition we prove the following theorem:

Theorem 3.2. *Let $(F, M, T_M, \epsilon_{M,1}, \epsilon_{M,2})$ be a fully black-box construction of a one-way function from a hard-on-average SVL instance. Then, at least one of the following properties holds:*

1. *$T_M(n) \geq 2^{\zeta n}$ for some constant $\zeta > 0$ (i.e., the reduction runs in exponential time).*
2. *$\epsilon_{M,1}(n^c) \cdot \epsilon_{M,2}(n) \leq 2^{-n/10}$ for some constant $c > 1$ (i.e., the security loss is exponential).*

In particular, Theorem 3.2 rules out standard "polynomial-time polynomial-loss" reductions. More generally, the theorem implies that if the running time $T_M(\cdot)$ of the reduction is sub-exponential and the adversary-dependent security loss $\epsilon_{M,1}(\cdot)$ is polynomial (as expected), then the adversary-independent security loss $\epsilon_{M,2}(\cdot)$ must be exponential (thus even ruling out constructions based on SVL instances with *sub-exponential* average-case hardness).

3.1 Proof Overview

In what follows we first describe the oracle, denoted $\mathcal{O}_{\mathsf{SVL}}$, on which we rely for proving Theorem 3.2. Then, we describe the structure of the proof, showing that relative to the oracle $\mathcal{O}_{\mathsf{SVL}}$ there exists a hard-on-average distribution of SVL instances, but there are no one-way functions. For the remainder of this section we remind the reader that a *q-query algorithm* is an oracle-aided algorithm $\mathcal{A}$ such that for any oracle $\mathcal{O}$ and input $x \in \{0,1\}^*$, the computation $\mathcal{A}^{\mathcal{O}}(x)$ consists of at most $q(|x|)$ oracle calls to $\mathcal{O}$.

The oracle $\mathcal{O}_{\mathsf{SVL}}$. The oracle $\mathcal{O}_{\mathsf{SVL}}$ is a valid SVL instance $\{(\mathsf{S}_n, \mathsf{V}_n, L(n))\}_{n\in\mathbb{N}}$ that is sampled via the following process for every $n \in \mathbb{N}$:

- Let $L(n) = 2^{n/2}$, $x_0 = 0^n$, and uniformly sample *distinct* elements $x_1, \ldots, x_{L(n)} \leftarrow \{0,1\}^n \setminus \{0^n\}$.

- The successor function $\mathsf{S}_n : \{0,1\}^n \to \{0,1\}^n$ is defined as

$$\mathsf{S}_n(x) = \begin{cases} x_{i+1} & \text{if } x = x_i \text{ for some } i \in \{0,\ldots,L(n)-1\} \\ x & \text{otherwise} \end{cases} .$$

- The verification function $\mathsf{V}_n : \{0,1\}^n \times [2^n] \to \{0,1\}$ is defined in a manner that is consistent with S_n (i.e., V_n is defined such that the instance is valid).

Part I: $\mathcal{O}_{\mathsf{SVL}}$ is a hard-on-average SVL instance. We show that the oracle $\mathcal{O}_{\mathsf{SVL}}$ itself is a hard-on-average SVL instance[5], which implies in particular that relative to the oracle $\mathcal{O}_{\mathsf{SVL}}$ there exists a hard-on-average distribution of SVL instances. We prove the following claim stating that, in fact, the oracle $\mathcal{O}_{\mathsf{SVL}}$ is an *exponentially* hard-on-average SVL instance (even without an index-generation algorithm):

Claim 3.3. *For every $q(n)$-query algorithm M, where $q(n) \leq L(n) - 1$, it holds that*

$$\Pr\left[M^{\mathcal{O}_{\mathsf{SVL}}}(1^n) \text{ solves } (\mathsf{S}_n, \mathsf{V}_n, L(n))\right] \leq \frac{(q(n)+1)\cdot L(n)}{2^n - q(n) - 1}$$

for all sufficiently large $n \in \mathbb{N}$, where the probability is taken over the choice of the oracle $\mathcal{O}_{\mathsf{SVL}} = \{(\mathsf{S}_n, \mathsf{V}_n, L(n))\}_{n\in\mathbb{N}}$ as described above.

The proof of the claim, which is provided in Section 3.2, is based on the following, rather intuitive, observation: Since the line $0^n \to x_1 \to \cdots \to x_{L(n)}$ is *sparse* and *uniformly sampled*, then any algorithm performing $q = q(n)$ oracle queries should not be able to query $\mathcal{O}_{\mathsf{SVL}}$ with any element on the line beyond the first q elements $0^n, x_1, \ldots, x_{q-1}$. In particular, for our choice of parameters, any such algorithm should have only an exponentially-small probability of reaching $x_{L(n)}$.

Part II: Inverting oracle-aided functions relative to $\mathcal{O}_{\mathsf{SVL}}$. We show that any oracle-aided function $F^{\mathcal{O}_{\mathsf{SVL}}}(\cdot)$ computable in time $t(n)$ can be inverted with high probability by an inverter that issues roughly $t(n)^4$ oracle queries. We prove the following claim:

Claim 3.4. *For every deterministic oracle-aided function F that is computable in time $t(n)$ there exists a $q(n)$-query algorithm $\mathcal{A}$, where $q(n) = O(t(n)^4)$, such that*

$$\Pr\left[\mathcal{A}^{\mathcal{O}_{\mathsf{SVL}}}\left(F^{\mathcal{O}_{\mathsf{SVL}}}(x)\right) \in \left(F^{\mathcal{O}_{\mathsf{SVL}}}\right)^{-1}\left(F^{\mathcal{O}_{\mathsf{SVL}}}(x)\right)\right] \geq \frac{1}{2}$$

for all sufficiently large $n \in \mathbb{N}$ and for every $x \in \{0,1\}^n$, where the probability is taken over the choice of the oracle $\mathcal{O}_{\mathsf{SVL}} = \{(\mathsf{S}_n, \mathsf{V}_n, L(n))\}_{n\in\mathbb{N}}$ as described above. Moreover, the algorithm $\mathcal{A}$ can be implemented in time polynomial in $q(n)$ given access to a PSPACE*-complete oracle.*

The proof of the claim, which is provided in Section 3.3, is based on the following approach. Consider the value $y = F^{\mathcal{O}_{\mathsf{SVL}}}(x)$ that is given as input to the inverter $\mathcal{A}$. Since F is computable in time $t = t(n)$, it can issue at most t oracle queries and therefore the observation used for proving Claim 3.3 implies that the computation $F^{\mathcal{O}_{\mathsf{SVL}}}(x)$ should not query $\mathcal{O}_{\mathsf{SVL}}$ with any elements on the line $0^n \to x_1 \to \cdots \to x_{L(n)}$ except for the first t elements $x_0, x_1, \ldots, x_{t-1}$. In this case, any S_n-query α in the computation $F^{\mathcal{O}_{\mathsf{SVL}}}(x)$ can be answered as follows: If $\alpha = x_i$ for some $i \in \{0,\ldots,t-1\}$ then the answer is x_{i+1}, and otherwise the answer is α. Similarly, any V_n-query (α, j) in the computation

[5] Formally speaking, as the SVL instance we consider oracle-aided circuits that simply call $\mathcal{O}_{\mathsf{SVL}}$ on their input and output the result.

$F^{\mathcal{O}_{\mathsf{SVL}}}(x)$ can be answered as follows: If $(\alpha, j) = (x_i, i)$ for some $i \in \{0, \ldots, t-1\}$ then the answer is 1, and otherwise the answer is 0.

This observation gives rise to the following inverter $\mathcal{A}$: First perform t queries to S_n for discovering $x_1, \ldots, x_t$, and then invert $y = F^{\mathcal{O}_{\mathsf{SVL}}}(x)$ relative to the oracle $\widetilde{\mathcal{O}_{\mathsf{SVL}}}$ defined via the following successor function $\widetilde{\mathsf{S}}_n$:

$$\widetilde{\mathsf{S}}_n(\alpha) = \begin{cases} x_{i+1} & \text{if } \alpha = x_i \text{ for some } i \in \{0, \ldots, t-1\} \\ \alpha & \text{otherwise} \end{cases}.$$

The formal proof is in fact more subtle, and requires a significant amount of caution when inverting $y = F^{\mathcal{O}_{\mathsf{SVL}}}(x)$ relative to the oracle $\widetilde{\mathcal{O}_{\mathsf{SVL}}}$. Specifically, the inverter $\mathcal{A}$ should find an input $\widetilde{x}$ such that the computations $F^{\widetilde{\mathcal{O}_{\mathsf{SVL}}}}(\widetilde{x})$ and $F^{\mathcal{O}_{\mathsf{SVL}}}(\widetilde{x})$ do not query the oracles $\widetilde{\mathcal{O}_{\mathsf{SVL}}}$ and $\mathcal{O}_{\mathsf{SVL}}$, respectively, with any of $x_t, \ldots, x_{L(n)}$. In this case, we show that indeed $F^{\mathcal{O}_{\mathsf{SVL}}}(\widetilde{x}) = y$ and the inverter is successful.

3.2 $\mathcal{O}_{\mathsf{SVL}}$ is a Hard-on-Average SVL Instance

The proof of Claim 3.3 relies on the fact that the line $0^n \to x_1 \to \cdots \to x_{L(n)}$ is *sparse* and *uniformly sampled*. This intuitively implies that any algorithm performing q oracle queries should not be able to query $\mathcal{O}_{\mathsf{SVL}}$ with any element on the line beyond the first q elements $0^n, x_1, \ldots, x_{q-1}$, except with an exponentially-small probability.

Given an oracle $\mathcal{O}_{\mathsf{SVL}} = \{(\mathsf{S}_n, \mathsf{V}_n, L(n))\}_{n\in\mathbb{N}}$, sampled as described in Section 3.1, and given a q-query algorithm M, for every $n \in \mathbb{N}$ and $i \in [q]$ we denote by α_i the random variable corresponding to M's ith oracle query if this is an S_n-query, and we denote by (α_i, k_i) the random variable corresponding to M's ith oracle query if this is a V_n-query. We denote by $\mathsf{HIT}_{M,n}^{\mathcal{O}_{\mathsf{SVL}}}$ the event in which there exist indices $j \in [q]$ and $i \in [L(n)]$ for which $\alpha_j = x_i$ but $x_{i-1} \notin \{\alpha_1, \ldots, \alpha_{j-1}\}$. That is, this is the event in which M queries $\mathcal{O}_{\mathsf{SVL}}$ with one of the x_i's *before* querying on x_{i-1}. In particular, note that if the event $\mathsf{HIT}_{M,n}^{\mathcal{O}_{\mathsf{SVL}}}$ does not occur, then M does not query $\mathcal{O}_{\mathsf{SVL}}$ with x_i for $i \in \{q, \ldots, L(n)\}$. The following claim bounds the probability of event $\mathsf{HIT}_{M,n}^{\mathcal{O}_{\mathsf{SVL}}}$.

Claim 3.5. *For every q-query algorithm M and for every $n \in \mathbb{N}$ it holds that*

$$\Pr\left[\mathsf{HIT}_{M,n}^{\mathcal{O}_{\mathsf{SVL}}}\right] \leq \frac{q \cdot L(n)}{2^n - q},$$

where the probability is taken over the choice of the oracle $\mathcal{O}_{\mathsf{SVL}} = \{(\mathsf{S}_n, \mathsf{V}_n, L(n))\}_{n\in\mathbb{N}}$. Moreover, q can be a bound on the number of calls to S_n and V_n.

Proof. Let M be a q-query algorithm, fix $n \in \mathbb{N}$, and fix $(\mathcal{O}_{\mathsf{SVL}})_{-n} = \{(\mathsf{S}_i, \mathsf{V}_i, T_i)\}_{i\in\mathbb{N}\setminus\{n\}}$ (i.e., we fix the entire oracle $\mathcal{O}_{\mathsf{SVL}}$ except for the nth SVL instance). For every $i \in [q]$ denote by M_i the following i-query algorithm: Invoke the computation $M^{\mathcal{O}_{\mathsf{SVL}}}$, and terminate once i oracle queries have been performed. Note that since we do not place any restriction on the running time of M and since the oracle distribution is known, we can assume without loss of generality that M is deterministic. Therefore, for every $i \in [q]$ and every fixing of the oracle $\mathcal{O}_{\mathsf{SVL}}$, the computation $M_i^{\mathcal{O}_{\mathsf{SVL}}}$ is the "prefix" of the computation $M^{\mathcal{O}_{\mathsf{SVL}}}$ which contains its first i oracle queries. This implies that

$$\Pr\left[\mathsf{HIT}_{M,n}^{\mathcal{O}_{\mathsf{SVL}}}\right] \leq \Pr\left[\mathsf{HIT}_{M_1,n}^{\mathcal{O}_{\mathsf{SVL}}}\right] + \sum_{i=1}^{q-1} \Pr\left[\mathsf{HIT}_{M_{i+1},n}^{\mathcal{O}_{\mathsf{SVL}}} \middle| \overline{\mathsf{HIT}_{M_i,n}^{\mathcal{O}_{\mathsf{SVL}}}}\right],$$

where the probability is taken over the choice of the nth SVL instance $(\mathsf{S}_n, \mathsf{V}_n, L(n))$ (i.e., over the choice of the elements $x_1, \ldots, x_{L(n)}$ that are used for defining the nth instance as described in Section 3.1).

For bounding the probability of the event $\mathsf{HIT}^{\mathcal{O}_{\mathsf{SVL}}}_{M_1,n}$, note that this event corresponds to the fact that M, without any information on $x_1,\ldots,x_{L(n)}$ (since no oracle queries have been issued so far), manages to produce an oracle query with $\alpha_1 \in \{x_1,\ldots,x_{L(n)}\}$. Since the value α_1 is fixed by the description of M, and we are now sampling distinct and uniformly distributed $x_1,\ldots,x_{L(n)} \leftarrow \{0,1\}^n \setminus \{0^n\}$, we have that

$$\Pr\left[\mathsf{HIT}^{\mathcal{O}_{\mathsf{SVL}}}_{M_1,n}\right] \leq \frac{\binom{2^n-2}{L(n)-1}}{\binom{2^n-1}{L(n)}} = \frac{L(n)}{2^n-1}.$$

For bounding the probability of the event $\mathsf{HIT}^{\mathcal{O}_{\mathsf{SVL}}}_{M_{i+1},n}$ given that $\overline{\mathsf{HIT}^{\mathcal{O}_{\mathsf{SVL}}}_{M_i,n}}$ occurred, we fix the queries $\alpha_1,\ldots,\alpha_i$ with the corresponding k_i's for the V_n queries, we fix their successors $\beta_1,\ldots,\beta_i$ where $\beta_j = \mathsf{S}_n(\alpha_j)$, and for each $j \in [i]$ and $k \in [L(n)]$ we fix whether $\alpha_j = x_k$ or not. This fixes the oracle answers to the above queries, hence fixes α_{i+1} by the assumption that M is deterministic. By the assumption $\overline{\mathsf{HIT}^{\mathcal{O}_{\mathsf{SVL}}}_{M_i,n}}$, there is some $0 \leq \ell \leq i$ for which $x_0,x_1,\ldots,x_{\ell-1} \in \{\alpha_1,\ldots,\alpha_i\}$ but $x_\ell,\ldots,x_{L(n)} \notin \{\alpha_1,\ldots,\alpha_i\}$. Hence $x_1,\ldots,x_\ell \in \{\beta_1,\ldots,\beta_i\}$ but $x_{\ell+1},\ldots,x_{L(n)} \notin \{\beta_1,\ldots,\beta_i\}$. No further information about $x_{\ell+1},\ldots,x_{L(n)}$ is known, therefore, we are now sampling distinct and uniformly distributed $x_{\ell+1},\ldots,x_{L(n)} \leftarrow \{0,1\}^n \setminus \{0^n,\beta_1,\ldots,\beta_i\}$, hence

$$\Pr\left[\mathsf{HIT}^{\mathcal{O}_{\mathsf{SVL}}}_{M_{i+1},n}\middle|\overline{\mathsf{HIT}^{\mathcal{O}_{\mathsf{SVL}}}_{M_i,n}}\right] \leq \frac{L(n)}{2^n-i-1}.$$

We conclude that

$$\begin{aligned}\Pr\left[\mathsf{HIT}^{\mathcal{O}_{\mathsf{SVL}}}_{M,n}\right] &\leq \Pr\left[\mathsf{HIT}^{\mathcal{O}_{\mathsf{SVL}}}_{M_1,n}\right] + \sum_{i=1}^{q-1}\Pr\left[\mathsf{HIT}^{\mathcal{O}_{\mathsf{SVL}}}_{M_{i+1},n}\middle|\overline{\mathsf{HIT}^{\mathcal{O}_{\mathsf{SVL}}}_{M_i,n}}\right] \\ &\leq \sum_{i=0}^{q-1}\frac{L(n)}{2^n-i-1} \\ &\leq \frac{q\cdot L(n)}{2^n-q}.\end{aligned}$$

■

Equipped with Claim 3.5 we can now easily derive the proof of Claim 3.3.

Proof of Claim 3.3. We modify M such that it queries the oracle S_n with its output before it terminates. Now, M is a $(q(n)+1)$-query algorithm, and by the assumption $q(n)+1 \leq L(n)$. If $M(1^n)$ solves $(\mathsf{S}_n,\mathsf{V}_n,L(n))$ then $\mathsf{HIT}^{\mathcal{O}_{\mathsf{SVL}}}_{M(1^n),n}$ occurs, and by Claim 3.5 we deduce

$$\Pr\left[M^{\mathcal{O}_{\mathsf{SVL}}}(1^n)\ \text{solves}\ (\mathsf{S}_n,\mathsf{V}_n,L(n))\right] \leq \Pr\left[\mathsf{HIT}^{\mathcal{O}_{\mathsf{SVL}}}_{M(1^n),n}\right] \leq \frac{(q(n)+1)\cdot L(n)}{2^n-q(n)-1}.$$

■

3.3 Inverting Oracle-Aided Functions Relative to $\mathcal{O}_{\mathsf{SVL}}$

Proof of Claim 3.4. Let F be a deterministic oracle-aided function computable in time $t(n)$, and let $p(n) = 1/2$ (although the proof goes through for any value of $0 < p(n) < 1$). We describe an oracle-aided algorithm $\mathcal{A}$ that manages to invert $F^{\mathcal{O}_{\mathsf{SVL}}}(x)$ for every x with high probability over the choice of the oracle $\mathcal{O}_{\mathsf{SVL}}$. Let $\mathcal{A}$ be the following oracle-aided algorithm that on input 1^n and $y = F^{\mathcal{O}_{\mathsf{SVL}}}(x)$, where $x \in \{0,1\}^n$, proceeds as follows:

$\mathcal{A}1$. Set $a(n) = 2 \cdot \log(3t(n)^2/p(n) + 1)$.

$\mathcal{A}2$. For every $1 \leq i < a(n)$, the algorithm $\mathcal{A}$ queries S_i on all possible inputs $\alpha \in \{0,1\}^i$.

$\mathcal{A}3$. For every $a(n) \leq i \leq t(n)$, the algorithm $\mathcal{A}$ repeatedly queries S_i for $t(n)$ times starting with the query 0^i (i.e., $\mathcal{A}$ discovers the line of length $t(n)$ starting from 0^i).

$\mathcal{A}4$. The algorithm $\mathcal{A}$ constructs the "fake" oracle $\widetilde{\mathcal{O}_{\mathsf{SVL}}}$ that is consistent with the "true" oracle $\mathcal{O}_{\mathsf{SVL}}$ on all queries performed in steps $\mathcal{A}2$ and $\mathcal{A}3$ above, and is defined as the identity function on all other queries.

$\mathcal{A}5$. The algorithm $\mathcal{A}$ finds and outputs an input $\widetilde{x} \in \{0,1\}^n$ such that $F^{\widetilde{\mathcal{O}_{\mathsf{SVL}}}}(\widetilde{x}) = y$ and such that the computation of $F^{\widetilde{\mathcal{O}_{\mathsf{SVL}}}}(\widetilde{x})$ does not query $\widetilde{\mathcal{O}_{\mathsf{SVL}}}$ with any input of the form $\mathsf{S}_i^{t(n)}(0^i)$ where $a(n) \leq i \leq t(n)$. If no such input $\widetilde{x}$ exists, then the algorithm $\mathcal{A}$ outputs $\perp$.

First, note that steps $\mathcal{A}4$ and $\mathcal{A}5$ do not require any queries to the oracle $\mathcal{O}_{\mathsf{SVL}}$. Second, note that the number of oracle queries made by $\mathcal{A}$ in steps $\mathcal{A}2$ and $\mathcal{A}3$ is at most $q(n) \leq t(n)^2 + 2 \cdot 2^{a(n)} = O(t(n)^4/p(n)^2)$.

Moreover, given oracle access to a PSPACE-complete oracle, the algorithm $\mathcal{A}$ can be implemented to run in time polynomial in $q(n)$. To see this, we observe that the only non-trivial step is $\mathcal{A}5$. The "fake" oracle $\widetilde{\mathcal{O}_{\mathsf{SVL}}}$ can be described in space polynomial in $q(n)$ (because it is the identity function on all but at most $q(n)$ queries), and then step $\mathcal{A}5$ can be efficiently computed using an oracle that decides the following PSPACE language:

$$\left\{ (\widetilde{\mathcal{O}_{\mathsf{SVL}}}, i, b) \;\middle|\; \begin{array}{c} \widetilde{x}_i = b \text{ for the lexicographically first } \widetilde{x} \text{ such that } F^{\widetilde{\mathcal{O}_{\mathsf{SVL}}}}(\widetilde{x}) = y \text{ and} \\ \text{such that the computation of } F^{\widetilde{\mathcal{O}_{\mathsf{SVL}}}}(\widetilde{x}) \text{ does not query } \widetilde{\mathcal{O}_{\mathsf{SVL}}} \\ \text{with any input of the form } \mathsf{S}_i^{t(n)}(0^i) \text{ where } a(n) \leq i \leq t(n) \end{array} \right\} .$$

We now prove that for any $n \in \mathbb{N}$ and $x \in \{0,1\}^n$, the algorithm $\mathcal{A}$ inverts $y = F^{\mathcal{O}_{\mathsf{SVL}}}(x)$ with probability at least $1 - p(n)$ over the choice of the oracle $\mathcal{O}_{\mathsf{SVL}}$. Fix $n \in \mathbb{N}$ and $x \in \{0,1\}^n$, and consider the oracle-aided algorithm M_x defined as follows:

$M1$. Compute $y = F^{\mathcal{O}_{\mathsf{SVL}}}(x)$.

$M2$. Compute $\widetilde{x} = \mathcal{A}^{\mathcal{O}_{\mathsf{SVL}}}(1^n, y)$.

$M3$. If $\widetilde{x} = \perp$ then output 0 and terminate.

$M4$. Compute $\widetilde{y} = F^{\mathcal{O}_{\mathsf{SVL}}}(\widetilde{x})$.

$M5$. If $\widetilde{y} = y$ then output 1, and otherwise output 0.

The probability over the choice of $\mathcal{O}_{\mathsf{SVL}}$ that M_x outputs 1 is exactly the probability that $\mathcal{A}$ manages to invert $y = F^{\mathcal{O}_{\mathsf{SVL}}}(x)$. Now suppose for all $a(n) \leq i \leq t(n)$ the event $\mathsf{HIT}_{M_x,i}^{\mathcal{O}_{\mathsf{SVL}}}$ does not occur. We aim to show that in this case $\widetilde{y} = y$. To start with, we claim that in this case the computation of M_x until step $M3$ does not query $\mathcal{O}_{\mathsf{SVL}}$ with an input of the form $\mathsf{S}_i^{t(n)}(0^i)$ where $a(n) \leq i \leq t(n)$:

- M_x does not query $\mathsf{S}_i^{t(n)}(0^i)$ in step $M1$ because $F^{\mathcal{O}_{\mathsf{SVL}}}(x)$ performs at most $t(n)$ queries, and querying $\mathsf{S}_i^{t(n)}(0^i)$ when $\mathsf{HIT}_{M_x,i}^{\mathcal{O}_{\mathsf{SVL}}}$ does not occur requires at least $t(n)+1$ queries.
- M_x does not query $\mathsf{S}_i^{t(n)}(0^i)$ in step $M2$ by the definition of the algorithm $\mathcal{A}$, since $\mathcal{A}$ only queries the oracle S_i with input of the form $\mathsf{S}_i^j(0^i)$ where $j \in [t(n)-1]$.

Note that since the computation of M_x until step $M3$ does not query $\mathcal{O}_{\mathsf{SVL}}$ with $\mathsf{S}_i^{t(n)}(0^i)$, and since the event $\mathsf{HIT}_{M,i}^{\mathcal{O}_{\mathsf{SVL}}}$ does not occur, then the computation of M_x until step $M3$ does not query $\mathcal{O}_{\mathsf{SVL}}$

with any input of the form $\mathsf{S}_i^k(0^i)$ where $k \in \{t(n), \ldots, L(i)\}$. At this point, since x itself satisfies $F^{\widetilde{\mathcal{O}_{\mathsf{SVL}}}}(x) = F^{\mathcal{O}_{\mathsf{SVL}}}(x) = y$, and since the computation of $F^{\widetilde{\mathcal{O}_{\mathsf{SVL}}}}(x)$ does not query $\mathsf{S}_i^{t(n)}(0^i)$, we know for sure that the algorithm $\mathcal{A}$ in step $M2$ will not return $\perp$. It remains to show that any $\widetilde{x}$ that $\mathcal{A}$ might return will satisfy $F^{\mathcal{O}_{\mathsf{SVL}}}(\widetilde{x}) = F^{\widetilde{\mathcal{O}_{\mathsf{SVL}}}}(\widetilde{x})$, hence $\widetilde{y} = y$ as claimed.

Assume by contradiction that $F^{\mathcal{O}_{\mathsf{SVL}}}(\widetilde{x}) \neq F^{\widetilde{\mathcal{O}_{\mathsf{SVL}}}}(\widetilde{x})$, and consider the first oracle query for which the computations of $F^{\mathcal{O}_{\mathsf{SVL}}}(\widetilde{x})$ and $F^{\widetilde{\mathcal{O}_{\mathsf{SVL}}}}(\widetilde{x})$ diverge. By the definition of $\widetilde{\mathcal{O}_{\mathsf{SVL}}}$, it must be a query to S_i or V_i where $a(n) \leq i \leq t(n)$, with input of the form $\mathsf{S}_i^j(0^i)$ where $j \in \{t(n), \ldots, L(i) - 1\}$ (in case of a query to the oracle V_i, $\mathsf{S}_i^j(0^i)$ is only the first argument of the input). The case $j = t(n)$ is impossible because $\mathcal{A}$ chooses $\widetilde{x}$ for which the computation of $F^{\widetilde{\mathcal{O}_{\mathsf{SVL}}}}(\widetilde{x})$ does not query $\mathsf{S}_i^{t(n)}(0^i)$. The case $j > t(n)$ is also impossible since until this point M_x did not query $\mathsf{S}_i^{t(n)}(0^i)$, and since the event $\mathsf{HIT}_{M,i}^{\mathcal{O}_{\mathsf{SVL}}}$ does not occur.

We conclude that if $y \neq \widetilde{y}$ then $\mathsf{HIT}_{M,i}^{\mathcal{O}_{\mathsf{SVL}}}$ occurs for some $a(n) \leq i \leq t(n)$. By the fact that M_x issues at most $3t(n)$ queries to S_i and V_i for every $a(n) \leq i \leq t(n)$, Claim 3.5 implies that

$$\begin{aligned}
\Pr_{\mathcal{O}_{\mathsf{SVL}}}\left[\mathcal{A}^{\mathcal{O}_{\mathsf{SVL}}}\left(F^{\mathcal{O}_{\mathsf{SVL}}}(x)\right) \notin \left(F^{\mathcal{O}_{\mathsf{SVL}}}\right)^{-1}\left(F^{\mathcal{O}_{\mathsf{SVL}}}(x)\right)\right] &\leq \sum_{i=\lceil a(n)\rceil}^{t(n)} \Pr_{\mathcal{O}_{\mathsf{SVL}}}\left[\mathsf{HIT}_{M_x,i}^{\mathcal{O}_{\mathsf{SVL}}}\right] \\
&\leq \sum_{i=\lceil a(n)\rceil}^{t(n)} \frac{3t(n)\cdot L(i)}{2^i - 3t(n)} \\
&\leq \sum_{i=\lceil a(n)\rceil}^{t(n)} \frac{3t(n)\cdot L(i)}{2^i - L(i)} \\
&= \sum_{i=\lceil a(n)\rceil}^{t(n)} \frac{3t(n)}{2^{i/2} - 1} \\
&\leq \frac{3t(n)^2}{2^{a(n)/2} - 1} \\
&\leq p(n).
\end{aligned}$$

■

3.4 Proof of Theorem 3.2

Proof of Theorem 3.2. Let $(F, M, T_M, \epsilon_{M,1}, \epsilon_{M,2})$ be a fully black-box construction of a one-way function from a hard-on-average distribution of SVL instances (recall Definition 3.1). Claim 3.4 guarantees an oracle-aided algorithm $\mathcal{A}$ that runs in polynomial time $T_{\mathcal{A}}(n)$ such that

$$\Pr\left[\mathcal{A}^{\mathsf{PSPACE},\mathcal{O}_{\mathsf{SVL}}}\left(F^{\mathcal{O}_{\mathsf{SVL}}}(x)\right) \in \left(F^{\mathcal{O}_{\mathsf{SVL}}}\right)^{-1}\left(F^{\mathcal{O}_{\mathsf{SVL}}}(x)\right)\right] \geq \epsilon_{\mathcal{A}}(n)$$

for all sufficiently large $n \in \mathbb{N}$ and for every $x \in \{0,1\}^n$, where $\epsilon_{\mathcal{A}}(n) = 1/2$, and the probability is taken over the choice of the oracle $\mathcal{O}_{\mathsf{SVL}}$. Definition 3.1 then guarantees that

$$\Pr\left[M^{\mathcal{A},\mathsf{PSPACE},\mathcal{O}_{\mathsf{SVL}}}(1^n) \text{ solves } (\mathsf{S}_n, \mathsf{V}_n, L(n))\right] \geq \epsilon_{M,1}\left(T_{\mathcal{A}}(n)/\epsilon_{\mathcal{A}}(n)\right)\cdot \epsilon_{M,2}(n)$$

for infinitely many values of $n \in \mathbb{N}$, where M runs in time $T_M(n)$, and the probability is again taken over the choice of the oracle $\mathcal{O}_{\mathsf{SVL}}$.

The algorithm M may invoke $\mathcal{A}$ on various security parameters (i.e., in general M is not restricted to invoking $\mathcal{A}$ only on security parameter n), and we denote by $\ell(n)$ the maximal security parameter on which M invokes $\mathcal{A}$ (when M itself is invoked on security parameter n). Thus, viewing $M^{\mathcal{A}}$ as a single oracle-aided algorithm that has access to a PSPACE-complete oracle and to the oracle $\mathcal{O}_{\mathsf{SVL}}$, its running time $T_{M^{\mathcal{A}}}(n)$ satisfies $T_{M^{\mathcal{A}}}(n) \leq T_M(n) \cdot T_{\mathcal{A}}(\ell(n))$ (this follows since M may invoke $\mathcal{A}$ at most $T_M(n)$ times, and the running time of $\mathcal{A}$ on each such invocation is at most $T_{\mathcal{A}}(\ell(n))$). In particular, viewing $M' \stackrel{\text{def}}{=} M^{\mathcal{A}^{\mathsf{PSPACE}}}$ as a single oracle-aided algorithm that has oracle access to the oracle $\mathcal{O}_{\mathsf{SVL}}$, implies that M' is a $q(n)$-query algorithm where $q(n) = T_{M^{\mathcal{A}}}(n)$. Claim 3.3 and our choice of $L(n) = 2^{n/2}$ then imply that

$$\epsilon_{M,1}\left(T_{\mathcal{A}}(n)/\epsilon_{\mathcal{A}}(n)\right) \cdot \epsilon_{M,2}(n) \leq \frac{(q(n)+1) \cdot 2^{n/2}}{2^n - q(n) - 1}.$$

There are now two possible cases to consider:

Case 1: $2^{n/4} \leq q(n)$. In this case, noting that $\ell(n) \leq T_M(n)$, we obtain that

$$2^{n/4} \leq q(n) = T_{M^{\mathcal{A}}}(n) \leq T_M(n) \cdot T_{\mathcal{A}}(\ell(n)) \leq T_M(n) \cdot T_{\mathcal{A}}(T_M(n)).$$

The running time $T_{\mathcal{A}}(n)$ of the adversary $\mathcal{A}$ (when given access to a PSPACE-complete oracle) is some fixed polynomial in n, and therefore $T_M(n) \geq 2^{\zeta n}$ for some constant $\zeta > 0$.

Case 2: $2^{n/4} > q(n)$. In this case we have that

$$\epsilon_{M,1}\left(T_{\mathcal{A}}(n)/\epsilon_{\mathcal{A}}(n)\right) \cdot \epsilon_{M,2}(n) \leq \frac{(q(n)+1) \cdot 2^{n/2}}{2^n - q(n) - 1} \leq \frac{1}{2^{n/10}},$$

and since $T_{\mathcal{A}}(n)$ is some fixed polynomial in n (and $\epsilon_{\mathcal{A}}(n)$ is a constant) we obtain that $\epsilon_{M,1}(n^c) \cdot \epsilon_{M,2}(n) \leq 2^{-n/10}$ for some constant $c > 1$. ■

4 Average-Case PPAD Hardness Does Not Imply Unique-TFNP Hardness

In this section we prove that there is no fully black-box construction of a hard-on-average distribution of TFNP instances having a unique solution from a hard-on-average distribution of instances of a PPAD-complete problem (proving, in particular, Theorem 1.3). Our result is obtained by presenting a distribution of oracles relative to which the following two properties hold:

1. There exists a hard-on-average distribution of instances of a PPAD-complete problem (specifically, we consider the source-or-sink problem).
2. There are no hard-on-average distributions over TFNP instances having a unique solution.

Recall that a TFNP instance with a unique solution, denoted a unique-TFNP instance (see Definitions 2.3 and 2.5), is of the form $\{\mathsf{Gen}_n, C_n\}_{n\in\mathbb{N}}$, where for every $n \in \mathbb{N}$ and for every index σ produced by Gen_n it holds that $C_n(\sigma, \cdot) : \{0,1\}^n \to \{0,1\}$ and there is a unique $x^* \in \{0,1\}^n$ such that $C_n(\sigma, x) = 1$. In particular, for any *valid* SVL instance $(\mathsf{Gen}, \mathsf{S}, \mathsf{V}, T)$ it holds that $(\mathsf{Gen}, \mathsf{V}(\cdot,\cdot,T))$ is a TFNP instance that has a unique solution since for every σ produced by Gen there is exactly one value x^* for which $\mathsf{V}(\sigma, x^*, T) = 1$. Therefore, our result shows, in particular, that there is no fully black-box construction of a hard-on-average distribution of SVL instances from a hard-on-average distribution of instances of a PPAD-complete problem[6].

[6] Recall that constructions in the opposite direction do exist: Any hard-on-average distribution of SVL instances can be used in a black-box manner to construct a hard-on-average distribution of instances of a PPAD-complete problem [AKV04, BPR15].

Recall that a source-or-sink instance is of the form $\{(\mathsf{Gen}_n, \mathsf{S}_n, \mathsf{P}_n)\}_{n\in\mathbb{N}}$, where for every $n \in \mathbb{N}$ and for every index σ produced by Gen_n it holds that $\mathsf{S}_n(\sigma,\cdot) : \{0,1\}^n \to \{0,1\}^n$ and $\mathsf{P}_n(\sigma,\cdot) : \{0,1\}^n \to \{0,1\}^n$. The following definition tailors the standard notion of a fully black-box construction to the specific primitives under consideration.

Definition 4.1. A fully black-box construction of a hard-on-average distribution of unique-TFNP instances from a hard-on-average distribution of source-or-sink instances consists of a sequence of polynomial-size oracle-aided circuits $C = \{\mathsf{Gen}_n, C_n\}_{n\in\mathbb{N}}$, an oracle-aided algorithm M that runs in time $T_M(\cdot)$, and functions $\epsilon_{M,1}(\cdot)$ and $\epsilon_{M,2}(\cdot)$, such that the following conditions hold:

- **Correctness**: For any source-or-sink instance $\mathcal{O}_{\mathsf{PPAD}}$, for any $n \in \mathbb{N}$, and for any index σ produced by $\mathsf{Gen}_n^{\mathcal{O}_{\mathsf{PPAD}}}$, there exists a unique $x^* \in \{0,1\}^n$ such that $C_n^{\mathcal{O}_{\mathsf{PPAD}}}(\sigma, x^*) = 1$.
- **Black-box proof of security:** For any source-or-sink instance $\mathcal{O}_{\mathsf{PPAD}} = \{(\mathsf{Gen}'_n, \mathsf{S}_n, \mathsf{P}_n)\}_{n\in\mathbb{N}}$, for any oracle-aided algorithm $\mathcal{A}$ that runs in time $T_{\mathcal{A}} = T_{\mathcal{A}}(n)$, and for any function $\epsilon_{\mathcal{A}}(\cdot)$, if

$$\Pr\left[\mathcal{A}^{\mathcal{O}_{\mathsf{PPAD}}}(1^n, \sigma) = x^* \text{ s.t. } C_n^{\mathcal{O}_{\mathsf{PPAD}}}(\sigma, x^*) = 1\right] \geq \epsilon_{\mathcal{A}}(n)$$

  for infinitely many values of $n \in \mathbb{N}$, where the probability is taken over the choice of $\sigma \leftarrow \mathsf{Gen}_n()$ and over the internal randomness of $\mathcal{A}$, then

$$\Pr\left[M^{\mathcal{A},\mathcal{O}_{\mathsf{PPAD}}}\left(1^n, \sigma'\right) \text{ solves } \left(\mathsf{S}_n(\sigma',\cdot), \mathsf{P}_n(\sigma',\cdot)\right)\right] \geq \epsilon_{M,1}\left(T_{\mathcal{A}}(n)/\epsilon_{\mathcal{A}}(n)\right) \cdot \epsilon_{M,2}(n)$$

  for infinitely many values of $n \in \mathbb{N}$, where the probability is taken over the choice of $\sigma' \leftarrow \mathsf{Gen}'_n()$ and over the internal randomness of M.

We note that, as in Definition 3.1, we split the security loss in the above definition to an adversary-dependent security loss and an adversary-independent security loss, as this allows us to capture constructions where one of these losses is super-polynomial whereas the other is polynomial. Equipped with the above definition we prove the following theorem:

Theorem 4.2. *Let $(C, M, T_M, \epsilon_{M,1}, \epsilon_{M,2})$ be a fully black-box construction of a hard-on-average distribution of unique-TFNP instances from a hard-on-average distribution of source-or-sink instances. Then, at least one of the following properties holds:*

1. *$T_M(n) \geq 2^{\zeta n}$ for some constant $\zeta > 0$ (i.e., the reduction runs in exponential time).*
2. *$\epsilon_{M,1}(n^c) \cdot \epsilon_{M,2}(n) \leq 2^{-n/10}$ for some constant $c > 1$ (i.e., the security loss is exponential).*

In particular, Theorem 4.2 rules out standard "polynomial-time polynomial-loss" reductions. More generally, the theorem implies that if the running time $T_M(\cdot)$ of the reduction is sub-exponential and the adversary-dependent security loss $\epsilon_{M,1}(\cdot)$ is polynomial (as expected), then the adversary-independent security loss $\epsilon_{M,2}(\cdot)$ must be exponential (thus even ruling out constructions based on SVL instances with *sub-exponential* average-case hardness).

4.1 Proof Overview

In what follows we first describe the oracle, denoted $\mathcal{O}_{\mathsf{PPAD}}$, on which we rely for proving Theorem 4.2. Then, we describe the structure of the proof, showing that relative to the oracle $\mathcal{O}_{\mathsf{PPAD}}$ there exists a hard-on-average distribution of source-or-sink instances, but there are no hard-on-average unique-TFNP instances. For the remainder of this section we remind the reader that a *q-query*

algorithm is an oracle-aided algorithm $\mathcal{A}$ such that for any oracle $\mathcal{O}$ and input $x \in \{0,1\}^*$, the computation $\mathcal{A}^{\mathcal{O}}(x)$ consists of at most $q(|x|)$ oracle calls to $\mathcal{O}$.

The oracle $\mathcal{O}_{\mathsf{PPAD}}$. The oracle $\mathcal{O}_{\mathsf{PPAD}}$ is a source-or-sink instance $\{(\mathsf{S}_n, \mathsf{P}_n)\}_{n\in\mathbb{N}}$ that is based on the same sparse structure used to define the oracle $\mathcal{O}_{\mathsf{SVL}}$ in Section 3. The oracle $\mathcal{O}_{\mathsf{PPAD}}$ is sampled via the following process for every $n \in \mathbb{N}$:

- Let $L(n) = 2^{n/2}$, $x_0 = 0^n$, and uniformly sample distinct elements $x_1, \ldots, x_{L(n)} \leftarrow \{0,1\}^n \setminus \{0^n\}$.
- The successor function $\mathsf{S}_n : \{0,1\}^n \to \{0,1\}^n$ is defined as

$$\mathsf{S}_n(x) = \begin{cases} x_{i+1} & \text{if } x = x_i \text{ for some } i \in \{0, \ldots, L(n)-1\} \\ x & \text{otherwise} \end{cases} .$$

- The predecessor function $\mathsf{P}_n : \{0,1\}^n \to \{0,1\}^n$ is defined in a manner that is consistent with the successor function S_n:

$$\mathsf{P}_n(x) = \begin{cases} x_{i-1} & \text{if } x = x_i \text{ for some } i \in \{1, \ldots, L(n)\} \\ x & \text{otherwise} \end{cases} .$$

Note that the oracle $\mathcal{O}_{\mathsf{PPAD}}$ corresponds to a source-or-sink instance that consists of the single line $0^n \to x_1 \to \cdots \to x_{L(n)}$, and therefore the only solution to this instance is the element $x_{L(n)}$.

Part I: $\mathcal{O}_{\mathsf{PPAD}}$ is a hard-on-average source-or-sink instance. We show that the oracle $\mathcal{O}_{\mathsf{PPAD}}$ itself is a hard-on-average source-or-sink instance, which implies in particular that relative to the oracle $\mathcal{O}_{\mathsf{PPAD}}$ there exists a hard-on-average distribution of instances to the source-or-sink problem. We prove the following claim stating that, in fact, the oracle $\mathcal{O}_{\mathsf{PPAD}}$ is an *exponentially* hard-on-average source-or-sink instance (even without an index-generation algorithm):

Claim 4.3. *For every $q(n)$-query algorithm M, where $q(n) \leq L(n) - 1$, it holds that*

$$\Pr\left[M^{\mathcal{O}_{\mathsf{PPAD}}}(1^n) \text{ solves } (\mathsf{S}_n, \mathsf{P}_n)\right] \leq \frac{(q(n)+1) \cdot L(n)}{2^n - q(n) - 1}$$

for all sufficiently large $n \in \mathbb{N}$, where the probability is taken over the choice of the oracle $\mathcal{O}_{\mathsf{PPAD}} = \{(\mathsf{S}_n, \mathsf{P}_n)\}_{n\in\mathbb{N}}$ as described above.

The proof of the claim, which is provided in Section 4.2, is based on an observation similar to the one used for proving Claim 3.3: Since the line $0^n \to x_1 \to \cdots \to x_{L(n)}$ is *sparse* and *uniformly sampled*, then any algorithm performing $q = q(n)$ oracle queries should not be able to query $\mathcal{O}_{\mathsf{PPAD}}$ with any element on the line beyond the first q elements $x_0, x_1, \ldots, x_{q-1}$. In particular, for our choice of parameters, any such algorithm should have only an exponentially-small probability of reaching $x_{L(n)}$.

Part II: Solving oracle-aided unique-TFNP instances relative to $\mathcal{O}_{\mathsf{PPAD}}$. We show that any oracle-aided unique-TFNP instance $\{\mathsf{Gen}_n, C_n\}_{n\in\mathbb{N}}$, where Gen_n and C_n are circuits that contain at most $q(n)$ oracle gates, can always be solved by an algorithm that issues roughly $q(n)^2$ oracle queries. We prove the following claim:

Claim 4.4. *Let $C = \{\mathsf{Gen}_n, C_n\}_{n\in\mathbb{N}}$ be an oracle-aided unique-TFNP instance, where Gen_n and C_n are circuits that contain at most $q(n)$ oracle gates each for every $n \in \mathbb{N}$. If C satisfies the correctness requirement stated in Definition 4.1, then there exists an $O(q(n)^2)$-query algorithm $\mathcal{A}$ such that*

$$\Pr\left[\mathcal{A}^{\mathcal{O}_{\mathsf{PPAD}}}(1^n, \sigma) = x^* \text{ s.t. } C_n^{\mathcal{O}_{\mathsf{PPAD}}}(\sigma, x^*) = 1\right] = 1$$

for every $n \in \mathbb{N}$, where the probability is taken over the choice of the oracle $\mathcal{O}_{\mathsf{PPAD}} = \{(\mathsf{S}_n, \mathsf{P}_n)\}_{n\in\mathbb{N}}$ as described above and over the choice of $\sigma \leftarrow \mathsf{Gen}_n^{\mathcal{O}_{\mathsf{PPAD}}}()$. Moreover, the algorithm $\mathcal{A}$ can be implemented in time $q(n)^2 \cdot \mathsf{poly}(n)$ given access to a PSPACE*-complete oracle.*

For proving Claim 4.4, one might be tempted to follow the same approach used for proving Claim 3.4, based on the sparse and uniform structure of the oracle. However, as discussed in Section 1.3, this approach seems to completely fail.

Our proof of Claim 4.4, which is provided in Section 4.3, is obtained by building upon Rudich's classic proof for ruling out black-box constructions of one-way permutations based on one-way functions [Rud88]. We show, by extending and refining Rudich's proof technique, that his approach provides a rich framework that allows to bound not only the limitations of one-way functions as a building block, but even the limitations of *significantly more structured* primitives as building blocks. Specifically, our proof of Claim 4.4 extends Rudich's technique for bounding the limitations of hard-on-average source-or-sink instances.

4.2 $\mathcal{O}_{\mathsf{PPAD}}$ is a Hard-on-Average Source-or-Sink Instance

Given an oracle $\mathcal{O}_{\mathsf{PPAD}} = \{(\mathsf{S}_n, \mathsf{P}_n)\}_{n\in\mathbb{N}}$, sampled as described in Section 4.1, and given a q-query algorithm M, for every $n \in \mathbb{N}$ and $i \in [q]$ we denote by $\alpha_i \in \{0,1\}$ the random variable corresponding to M's ith oracle query to S_n or P_n (note that we ignore oracle queries to S_i or P_i where $i \neq n$). We denote by $\mathsf{HIT}_{M,n}^{\mathcal{O}_{\mathsf{PPAD}}}$ the event in which there exist indices $j \in [q]$ and $i \in [L(n)]$ for which $\alpha_j = x_i$ but $x_{i-1} \notin \{\alpha_1, \ldots, \alpha_{j-1}\}$. That is, this is the event in which M queries $\mathcal{O}_{\mathsf{PPAD}}$ with one of the x_i's *before* querying on x_{i-1}. In particular, note that if the event $\mathsf{HIT}_{M,n}^{\mathcal{O}_{\mathsf{PPAD}}}$ does not occur, then M does not query $\mathcal{O}_{\mathsf{PPAD}}$ with x_i for $i \in \{q, \ldots, L(n)\}$. The following claim bounds the probability of event $\mathsf{HIT}_{M,n}^{\mathcal{O}_{\mathsf{PPAD}}}$ (its proof is essentially identical to that of Claim 3.5).

Claim 4.5. *For every q-query algorithm M it holds that*

$$\Pr\left[\mathsf{HIT}_{M,n}^{\mathcal{O}_{\mathsf{PPAD}}}\right] \leq \frac{q \cdot L(n)}{2^n - q}$$

for all sufficiently large $n \in \mathbb{N}$, where the probability is taken over the choice of the oracle $\mathcal{O}_{\mathsf{PPAD}} = \{(\mathsf{S}_n, \mathsf{P}_n)\}_{n\in\mathbb{N}}$.

Proof. Let M be a q-query algorithm, fix $n \in \mathbb{N}$, and fix $(\mathcal{O}_{\mathsf{PPAD}})_{-n} = \{(\mathsf{S}_i, \mathsf{P}_i)\}_{i\in\mathbb{N}\setminus\{n\}}$ (i.e., we fix the entire oracle $\mathcal{O}_{\mathsf{PPAD}}$ except for the nth source-or-sink instance). For every $i \in [q]$ denote by M_i the following i-query algorithm: Invoke the computation $M^{\mathcal{O}_{\mathsf{PPAD}}}$, and terminate once i oracle queries have been performed. Note that since we do not place any restriction on the running time of M and since the oracle distribution is known, we can assume without loss of generality that M is deterministic. Therefore, for every $i \in [q]$ and every fixing of the oracle $\mathcal{O}_{\mathsf{PPAD}}$, the computation $M_i^{\mathcal{O}_{\mathsf{PPAD}}}$ is the "prefix" of the computation $M^{\mathcal{O}_{\mathsf{PPAD}}}$ which contains its first i oracle queries. This implies that

$$\Pr\left[\mathsf{HIT}_{M,n}^{\mathcal{O}_{\mathsf{PPAD}}}\right] \leq \Pr\left[\mathsf{HIT}_{M_1,n}^{\mathcal{O}_{\mathsf{PPAD}}}\right] + \sum_{i=1}^{q-1} \Pr\left[\mathsf{HIT}_{M_{i+1},n}^{\mathcal{O}_{\mathsf{PPAD}}} \middle| \overline{\mathsf{HIT}_{M_i,n}^{\mathcal{O}_{\mathsf{PPAD}}}}\right],$$

where the probability is taken over the choice of the nth source-or-sink instance $(\mathsf{S}_n, \mathsf{P}_n)$ (i.e., over the choice of the elements $x_1, \ldots, x_{L(n)}$ that are used for defining the nth instance as described in Section 4.1).

For bounding the probability of the event $\mathsf{HIT}_{M_1,n}^{\mathcal{O}_{\mathsf{PPAD}}}$, note that this event corresponds to the fact that M, without any information on $x_1, \ldots, x_{L(n)}$ (since no oracle queries have been issued so

far), manages to produce an oracle query with $\alpha_1 \in \{x_1, \ldots, x_{L(n)}\}$. Since the value α_1 is fixed by the description of M, and we are now sampling distinct and uniformly distributed $x_1, \ldots, x_{L(n)} \leftarrow \{0,1\}^n \setminus \{0^n\}$, we have that

$$\Pr\left[\mathsf{HIT}_{M_1,n}^{\mathcal{O}_{\mathsf{PPAD}}}\right] \leq \frac{\binom{2^n-2}{L(n)-1}}{\binom{2^n-1}{L(n)}} = \frac{L(n)}{2^n-1}.$$

For bounding the probability of the event $\mathsf{HIT}_{M_{i+1},n}^{\mathcal{O}_{\mathsf{PPAD}}}$ given that $\overline{\mathsf{HIT}_{M_i,n}^{\mathcal{O}_{\mathsf{PPAD}}}}$ occurred, we fix the queries $\alpha_1, \ldots, \alpha_i$, we fix their successors $\beta_1, \ldots, \beta_i$ where $\beta_j = \mathsf{S}_n(\alpha_j)$, fix their predecessors $\gamma_1, \ldots, \gamma_i$ where $\gamma_j = \mathsf{P}_n(\alpha_j)$, and for each $j \in [i]$ and $k \in [L(n)]$ we fix whether $\alpha_j = x_k$ or not. This fixes the oracle answers to the above queries, hence fixes α_{i+1} by the assumption that M is deterministic. By the assumption $\overline{\mathsf{HIT}_{M_i,n}^{\mathcal{O}_{\mathsf{PPAD}}}}$, there is some $0 \leq \ell \leq i$ for which $x_0, x_1, \ldots, x_{\ell-1} \in \{\alpha_1, \ldots, \alpha_i\}$ but $x_\ell, \ldots, x_{L(n)} \notin \{\alpha_1, \ldots, \alpha_i\}$. Hence $x_1, \ldots, x_\ell \in \{\beta_1, \ldots, \beta_i\}$ but $x_{\ell+1}, \ldots, x_{L(n)} \notin \{\beta_1, \ldots, \beta_i\}$. No further information about $x_{\ell+1}, \ldots, x_{L(n)}$ is known, therefore, we are now sampling distinct and uniformly distributed $x_{\ell+1}, \ldots, x_{L(n)} \leftarrow \{0,1\}^n \setminus \{0^n, \beta_1, \ldots, \beta_i\}$, hence

$$\Pr\left[\mathsf{HIT}_{M_{i+1},n}^{\mathcal{O}_{\mathsf{PPAD}}} \middle| \overline{\mathsf{HIT}_{M_i,n}^{\mathcal{O}_{\mathsf{PPAD}}}}\right] \leq \frac{L(n)}{2^n-i-1}.$$

We conclude that

$$\begin{aligned}
\Pr\left[\mathsf{HIT}_{M,n}^{\mathcal{O}_{\mathsf{PPAD}}}\right] &\leq \Pr\left[\mathsf{HIT}_{M_1,n}^{\mathcal{O}_{\mathsf{PPAD}}}\right] + \sum_{i=1}^{q-1} \Pr\left[\mathsf{HIT}_{M_{i+1},n}^{\mathcal{O}_{\mathsf{PPAD}}} \middle| \overline{\mathsf{HIT}_{M_i,n}^{\mathcal{O}_{\mathsf{PPAD}}}}\right] \\
&\leq \sum_{i=0}^{q-1} \frac{L(n)}{2^n-i-1} \\
&\leq \frac{q \cdot L(n)}{2^n-q}.
\end{aligned}$$

■

Equipped with Claim 4.5 we can now easily derive the proof of Claim 4.3.

Proof of Claim 4.3. We modify M such that it queries the oracle S_n with its output before it terminates. Now, M is a $(q(n)+1)$-query algorithm, and by the assumption $q(n)+1 \leq L(n)$, if $M(1^n)$ solves $(\mathsf{S}_n, \mathsf{P}_n)$ then $\mathsf{HIT}_{M(1^n),n}^{\mathcal{O}_{\mathsf{PPAD}}}$ occurs. By Claim 4.5 we deduce

$$\Pr\left[M^{\mathcal{O}_{\mathsf{PPAD}}}(1^n) \text{ solves } (\mathsf{S}_n, \mathsf{P}_n)\right] \leq \Pr\left[\mathsf{HIT}_{M(1^n),n}^{\mathcal{O}_{\mathsf{PPAD}}}\right] \leq \frac{(q(n)+1) \cdot L(n)}{2^n - q(n) - 1}.$$

■

4.3 Solving Oracle-Aided Unique-TFNP Instances Relative to $\mathcal{O}_{\mathsf{PPAD}}$

Proof of Claim 4.4. Fix the oracle $\mathcal{O}_{\mathsf{PPAD}} = \{(\mathsf{S}_n, \mathsf{P}_n)\}_{n\in\mathbb{N}}$ and let $C = \{\mathsf{Gen}_n, C_n\}_{n\in\mathbb{N}}$ be an oracle-aided unique-TFNP instance that satisfies the correctness requirement stated in Definition 4.1, where each of Gen_n and C_n contains at most $q(n)$ oracle gates for any $n \in \mathbb{N}$. Consider the following oracle-aided algorithm $\mathcal{A}$ that on input an index σ produced by $\mathsf{Gen}_n^{\mathcal{O}_{\mathsf{PPAD}}}$ would like to find an input $x \in \{0,1\}^n$ such that $C_n^{\mathcal{O}_{\mathsf{PPAD}}}(\sigma, x) = 1$. The algorithm $\mathcal{A}$ initializes two empty sets, Q_{S} and

Q_{P}, which at any point in time will be consistent with the functions $\mathsf{S} = \{\mathsf{S}_n\}_{n\in\mathbb{N}}$ and $\mathsf{P} = \{\mathsf{P}_n\}_{n\in\mathbb{N}}$, respectively. That is, the set Q_{S} will contain pairs of the form (α, β) where $\mathsf{S}_{|\alpha|}(\alpha) = \beta$, and the set Q_{P} will contain pairs of the form (β, α) where $\mathsf{P}_{|\beta|}(\beta) = \alpha$. The algorithm $\mathcal{A}$ performs the following steps for $q(n) + 1$ iterations:

Step 1. The algorithm $\mathcal{A}$ finds an oracle $\widetilde{\mathcal{O}_{\mathsf{PPAD}}} = \left\{\left(\widetilde{\mathsf{S}}_n, \widetilde{\mathsf{P}}_n\right)\right\}_{n\in\mathbb{N}}$ and values $\widetilde{r} \in \{0,1\}^*$ and $\widetilde{x} \in \{0,1\}^n$ subject to the following three requirements:

- $\widetilde{\mathcal{O}_{\mathsf{PPAD}}}$ is consistent with the sets Q_{S} and Q_{P}. That is, for every $(\alpha, \beta) \in Q_{\mathsf{S}}$ it holds that $\widetilde{\mathsf{S}}_{|\alpha|}(\alpha) = \beta$, and for every $(\beta, \alpha) \in Q_{\mathsf{P}}$ it holds that $\widetilde{\mathsf{P}}_{|\beta|}(\beta) = \alpha$.
- $\mathsf{Gen}_n^{\widetilde{\mathcal{O}_{\mathsf{PPAD}}}}(\widetilde{r}) = \sigma$.
- $C_n^{\widetilde{\mathcal{O}_{\mathsf{PPAD}}}}(\sigma, \widetilde{x}) = 1$.

Step 2. The algorithm $\mathcal{A}$ computes $\widetilde{y} = C_n^{\mathcal{O}_{\mathsf{PPAD}}}(\sigma, \widetilde{x})$, and if $\widetilde{y} = 1$ then it outputs $\widetilde{x}$ and terminates.

Step 3. The algorithm $\mathcal{A}$ queries the oracle $\mathcal{O}_{\mathsf{PPAD}}$ with all inputs to $\widetilde{\mathcal{O}_{\mathsf{PPAD}}}$-gates in the computations $\mathsf{Gen}_n^{\widetilde{\mathcal{O}_{\mathsf{PPAD}}}}(\widetilde{r})$ and $C_n^{\widetilde{\mathcal{O}_{\mathsf{PPAD}}}}(\sigma, \widetilde{x})$, and adds these queries to the sets Q_{S} and Q_{P}.

That is, for every input α to an $\widetilde{\mathsf{S}}$-gate in the computation $\mathsf{Gen}_n^{\widetilde{\mathcal{O}_{\mathsf{PPAD}}}}(\widetilde{r})$ or in the computation $C_n^{\widetilde{\mathcal{O}_{\mathsf{PPAD}}}}(\sigma, \widetilde{x})$, the algorithm $\mathcal{A}$ computes $\beta = \mathsf{S}_{|\alpha|}(\alpha)$, and adds the pair (α, β) to the set Q_{S}. Similarly, for every input β to a $\widetilde{\mathsf{P}}$-gate in the computation $\mathsf{Gen}_n^{\widetilde{\mathcal{O}_{\mathsf{PPAD}}}}(\widetilde{r})$ or in the computation $C_n^{\widetilde{\mathcal{O}_{\mathsf{PPAD}}}}(\sigma, \widetilde{x})$, the algorithm $\mathcal{A}$ computes $\alpha = \mathsf{P}_{|\beta|}(\beta)$, and adds the pair (β, α) to the set Q_{P}.

If the algorithm $\mathcal{A}$ did not return an output during the above iterations, then it outputs $\perp$. In terms of the number of oracle queries made by $\mathcal{A}$, observe that step 1 does not require any oracle queries, while steps 2 and 3 require at most $3q(n)$ queries. Therefore, the total number of queries made by $\mathcal{A}$ is $(q(n) + 1) \cdot 3q(n) = O(q(n)^2)$, as required.

Moreover, given oracle access a PSPACE-complete oracle, the algorithm $\mathcal{A}$ can be implemented to run in time $q(n)^2 \cdot \mathsf{poly}(n)$. To see this, we observe that there exists an oracle $\widetilde{\mathcal{O}_{\mathsf{PPAD}}}$ that satisfies the requirements in step 1 and can be described in space polynomial in $q(n)$. This is because the only queries of $\widetilde{\mathcal{O}_{\mathsf{PPAD}}}$ that matter are those in the sets Q_{S} and Q_{P}, and the queries performed in the computations $\mathsf{Gen}_n^{\widetilde{\mathcal{O}_{\mathsf{PPAD}}}}(\widetilde{r})$ and $C_n^{\widetilde{\mathcal{O}_{\mathsf{PPAD}}}}(\sigma, \widetilde{x})$. Therefore, the algorithm can be efficiently computed using an oracle that decides the following PSPACE language:

$$\left\{ (\sigma, Q_{\mathsf{S}}, Q_{\mathsf{P}}, i, b) \;\middle|\; \begin{array}{c} \text{The } i\text{th bit of } (\widetilde{x}, Q') \text{ is } b, \text{ where } (\widetilde{\mathcal{O}_{\mathsf{PPAD}}}, \widetilde{r}, \widetilde{x}) \text{ is lexicographically first} \\ \text{tuple satisfying the requirements in step 1 with respect to } (\sigma, Q_{\mathsf{S}}, Q_{\mathsf{P}}), \\ \text{and } Q' \text{ is the set of queries described in step 3} \end{array} \right\},$$

which allows discovering the input $\widetilde{x}$ required in step 2, as well as the queries required in step 3.

In the remainder of this proof, we show that $\mathcal{A}$ is always successful in one of its $q(n)+1$ iterations. This follows from the following claim:

Claim 4.6. *Let $x^* \in \{0,1\}^n$ be the unique input such that $C_n^{\mathcal{O}_{\mathsf{PPAD}}}(\sigma, x^*) = 1$. Then, in each iteration, at least one of the following events occur:*

- *In step 2 of the iteration $\mathcal{A}$ outputs x^*.*
- *During step 3 of the iteration $\mathcal{A}$ adds to Q_{S} or Q_{P} a new $\mathcal{O}_{\mathsf{PPAD}}$-query that is performed in the computation $C_n^{\mathcal{O}_{\mathsf{PPAD}}}(\sigma, x^*)$.*

We now show that Claim 4.6 indeed guarantees that $\mathcal{A}$ is always successful when repeating steps 1–3 above for $q(n)+1$ iterations. Let $x^* \in \{0,1\}^n$ be the unique input such that $C_n^{\mathcal{O}_{\mathsf{PPAD}}}(\sigma, x^*) = 1$, and assume that in the first $q(n)$ iterations $\mathcal{A}$ does not output x^* in step 2. Claim 4.6 implies that in each of these $q(n)$ iterations $\mathcal{A}$ adds to Q_{S} or Q_{P} a new $\mathcal{O}_{\mathsf{PPAD}}$-query that is performed in the computation $C_n^{\mathcal{O}_{\mathsf{PPAD}}}(\sigma, x^*)$. Since this computation contains at most $q(n)$ oracle queries to $\mathcal{O}_{\mathsf{PPAD}}$, at the end of the first $q(n)$ iterations we are guaranteed that all of these queries are included in the sets Q_{S} and Q_{P}. Therefore, in the final iteration, for any $\widetilde{\mathcal{O}_{\mathsf{PPAD}}}$ that will be chosen in step 1 it holds that $C_n^{\widetilde{\mathcal{O}_{\mathsf{PPAD}}}}(\sigma, x^*) = 1$ since $\widetilde{\mathcal{O}_{\mathsf{PPAD}}}$ is chosen to be consistent with Q_{S} and Q_{P}, and by uniqueness this is the only solution to $C_n^{\widetilde{\mathcal{O}_{\mathsf{PPAD}}}}(\sigma, \cdot) = 1$. Thus, in step 2 of this iteration $\mathcal{A}$ is guaranteed to output x^*. We now conclude the proof of Claim 4.4 by proving Claim 4.6.

Proof of Claim 4.6. Let $x^* \in \{0,1\}^n$ be the unique input such that $C_n^{\mathcal{O}_{\mathsf{PPAD}}}(\sigma, x^*) = 1$, and assume towards a contradiction that in some iteration $j \in [q(n)+1]$ the following two events occur:

- In step 2 of the iteration $\mathcal{A}$ does not output x^*. In particular, this implies that for the input $\widetilde{x}$ that $\mathcal{A}$ finds in this iteration it holds that $\widetilde{x} \neq x^*$.
- During step 3 of the iteration $\mathcal{A}$ does not add to Q_{S} or Q_{P} a new $\mathcal{O}_{\mathsf{PPAD}}$-query that is performed in the computation $C_n^{\mathcal{O}_{\mathsf{PPAD}}}(\sigma, x^*)$. In particular, all inputs to $\widetilde{\mathsf{S}}$-gates and $\widetilde{\mathsf{P}}$-gates in the computation $\mathsf{Gen}_n^{\widetilde{\mathcal{O}_{\mathsf{PPAD}}}}(\widetilde{r})$ and all inputs to $\widetilde{\mathsf{S}}$-gates and $\widetilde{\mathsf{P}}$-gates in the computation $C_n^{\widetilde{\mathcal{O}_{\mathsf{PPAD}}}}(\sigma, \widetilde{x})$ are either already in the sets Q_{S} and Q_{P}, respectively, at the beginning of the jth iteration, or are not used as inputs to S-gates or P-gates in the computation $C_n^{\mathcal{O}_{\mathsf{PPAD}}}(\sigma, x^*)$.

We now show that, in fact, there exists an oracle $\mathcal{O}'_{\mathsf{PPAD}}$, which is a source-or-sink instance, such that σ is a valid output of $\mathsf{Gen}_n^{\mathcal{O}'_{\mathsf{PPAD}}}$ but $C_n^{\mathcal{O}'_{\mathsf{PPAD}}}(\sigma, x^*) = C_n^{\mathcal{O}'_{\mathsf{PPAD}}}(\sigma, \widetilde{x}) = 1$. This contradicts the correctness requirement stated in Definition 4.1, asking that $C_n(\alpha, \cdot)$ has a *unique* solution for any valid index σ and relative to *any* source-or-sink oracle. The oracle $\mathcal{O}'_{\mathsf{PPAD}} = \{(\mathsf{S}'_n, \mathsf{P}'_n)\}_{n\in\mathbb{N}}$ is defined as follows (according to the following 4 types of possible inputs):

- **Type 1 inputs:** For every pair $(\alpha, \beta) \in Q_{\mathsf{S}}$ we set $\mathsf{S}'_{|\alpha|}(\alpha) = \beta$, and for every pair $(\beta, \alpha) \in Q_{\mathsf{P}}$ we set $\mathsf{P}'_{|\beta|}(\beta) = \alpha$.

  Note that since Q_{S} and Q_{P} are consistent with $\mathcal{O}_{\mathsf{PPAD}}$, and $\widetilde{\mathcal{O}_{\mathsf{PPAD}}}$ is consistent with Q_{S} and Q_{P}, then for all type 1 inputs α and β it holds that $\mathsf{S}'(\alpha) = \mathsf{S}(\alpha) = \widetilde{\mathsf{S}}(\alpha)$ and $\mathsf{P}'(\beta) = \mathsf{P}(\beta) = \widetilde{\mathsf{P}}(\beta)$.
- **Type 2 inputs:** For every input α that is used as input to an S-gate in the computation $C_n^{\mathcal{O}_{\mathsf{PPAD}}}(\sigma, x^*)$ and $(\alpha, \cdot) \notin Q_{\mathsf{S}}$, we set $\mathsf{S}'(\alpha) = \mathsf{S}(\alpha)$. Similarly, for every input β that is used as input to a P-gate in the computation $C_n^{\mathcal{O}_{\mathsf{PPAD}}}(\sigma, x^*)$ and $(\beta, \cdot) \notin Q_{\mathsf{P}}$, we set $\mathsf{P}'(\beta) = \mathsf{P}(\beta)$.
- **Type 3 inputs:** For every input α that is used as input to a $\widetilde{\mathsf{S}}$-gate in the computation $\mathsf{Gen}_n^{\widetilde{\mathcal{O}_{\mathsf{PPAD}}}}(\widetilde{r})$ or in the computation $C_n^{\widetilde{\mathcal{O}_{\mathsf{PPAD}}}}(\sigma, \widetilde{x})$ and $(\alpha, \cdot) \notin Q_{\mathsf{S}}$, we set $\mathsf{S}'(\alpha) = \widetilde{\mathsf{S}}(\alpha)$. Similarly, for every input β that is used as input to a $\widetilde{\mathsf{P}}$-gate in the computation $\mathsf{Gen}_n^{\widetilde{\mathcal{O}_{\mathsf{PPAD}}}}(\widetilde{r})$ or in the computation $C_n^{\widetilde{\mathcal{O}_{\mathsf{PPAD}}}}(\sigma, \widetilde{x})$ and $(\beta, \cdot) \notin Q_{\mathsf{P}}$, we set $\mathsf{P}'(\beta) = \widetilde{\mathsf{P}}(\beta)$.
- **Type 4 inputs:** For any other inputs α and β we set $\mathsf{S}'(\alpha)$ and $\mathsf{P}'(\beta)$ to arbitrary values (e.g., we set them to $0^{|\alpha|}$ and $0^{|\beta|}$, respectively).

First, note that the oracle $\mathcal{O}'_{\mathsf{PPAD}}$ is indeed a source-or-sink instance since its successor and predecessor functions are well defined (i.e., the above 4 types of inputs are indeed a *partition* of the input

space). Relative to $\mathcal{O}'_{\mathsf{PPAD}}$, however, it holds that $\mathsf{Gen}^{\mathcal{O}'_{\mathsf{PPAD}}}(\widetilde{r}) = \widetilde{\mathsf{Gen}_n^{\mathcal{O}_{\mathsf{PPAD}}}(\widetilde{r})} = \sigma$ (i.e., σ is a valid index relative to $\mathcal{O}'_{\mathsf{PPAD}}$), but $C_n^{\mathcal{O}'_{\mathsf{PPAD}}}(\sigma, x^*) = C_n^{\mathcal{O}_{\mathsf{PPAD}}}(\sigma, x^*) = 1$ and $C_n^{\mathcal{O}'_{\mathsf{PPAD}}}(\sigma, \widetilde{x}) = \widetilde{C_n^{\mathcal{O}_{\mathsf{PPAD}}}(\sigma, \widetilde{x})} = 1$. Recall that $x^* \neq \widetilde{x}$ and this contradicts the fact that $C_n^{\mathcal{O}'_{\mathsf{PPAD}}}(\sigma, \cdot)$ has a unique solution. ■

This settles the proof of Claim 4.4. ■

4.4 Proof of Theorem 4.2

Proof of Theorem 4.2. Let $(C, M, T_M, \epsilon_{M,1}, \epsilon_{M,2})$ be a fully black-box construction of a hard-on-average distribution of unique-TFNP instances from a hard-on-average distribution of source-or-sink instances (recall Definition 4.1), where $C = \{\mathsf{Gen}_n, C_n\}_{n\in\mathbb{N}}$. Claim 4.4 guarantees an oracle-aided algorithm $\mathcal{A}$ that runs in polynomial time $T_{\mathcal{A}}(n)$ such that

$$\Pr\left[\mathcal{A}^{\mathcal{O}_{\mathsf{PPAD}}}(1^n, \sigma) = x^* \text{ s.t. } C_n^{\mathcal{O}_{\mathsf{PPAD}}}(\sigma, x^*) = 1\right] = \epsilon_{\mathcal{A}}(n)$$

for all $n \in \mathbb{N}$, where $\epsilon_{\mathcal{A}}(n) = 1$, and the probability is taken over the choice of the oracle $\mathcal{O}_{\mathsf{PPAD}}$ and over the choice of $\sigma \leftarrow \mathsf{Gen}_n^{\mathcal{O}_{\mathsf{PPAD}}}()$. Definition 4.1 then guarantees that

$$\Pr\left[M^{\mathcal{A},\mathsf{PSPACE},\mathcal{O}_{\mathsf{PPAD}}}(1^n) \text{ solves } (\mathsf{S}_n, \mathsf{P}_n)\right] \geq \epsilon_{M,1}\left(T_{\mathcal{A}}(n)/\epsilon_{\mathcal{A}}(n)\right) \cdot \epsilon_{M,2}(n)$$

for infinitely many values of $n \in \mathbb{N}$, where M runs in time $T_M(n)$, and the probability is taken over the choice of the oracle $\mathcal{O}_{\mathsf{PPAD}} = \{(\mathsf{S}_n, \mathsf{P}_n)\}_{n\in\mathbb{N}}$.

The algorithm M may invoke $\mathcal{A}$ on various security parameters (i.e., in general M is not restricted to invoking $\mathcal{A}$ only on security parameter n), and we denote by $\ell(n)$ the maximal security parameter on which M invokes $\mathcal{A}$ (when M itself is invoked on security parameter n). Thus, viewing $M^{\mathcal{A}}$ as a single oracle-aided algorithm that has access to a PSPACE-complete oracle and to the oracle $\mathcal{O}_{\mathsf{PPAD}}$, its running time $T_{M^{\mathcal{A}}}(n)$ satisfies $T_{M^{\mathcal{A}}}(n) \leq T_M(n) \cdot T_{\mathcal{A}}(\ell(n))$ (this follows since M may invoke $\mathcal{A}$ at most $T_M(n)$ times, and the running time of $\mathcal{A}$ on each such invocation is at most $T_{\mathcal{A}}(\ell(n))$). In particular, viewing $M' \stackrel{\mathsf{def}}{=} M^{\mathcal{A}^{\mathsf{PSPACE}}}$ as a single oracle-aided algorithm that has oracle access to the oracle $\mathcal{O}_{\mathsf{PPAD}}$, implies that M' is a $q(n)$-query algorithm where $q(n) = T_{M^{\mathcal{A}}}(n)$. Claim 4.3 then implies that

$$\epsilon_{M,1}\left(T_{\mathcal{A}}(n)/\epsilon_{\mathcal{A}}(n)\right) \cdot \epsilon_{M,2}(n) \leq \frac{(q(n)+1) \cdot L(n)}{2^n - q(n) - 1}.$$

There are now two possible cases to consider:

Case 1: $2^{n/4} \leq q(n)$. In this case, noting that $\ell(n) \leq T_M(n)$, we obtain that

$$2^{n/4} \leq q(n) = T_{M^{\mathcal{A}}}(n) \leq T_M(n) \cdot T_{\mathcal{A}}(\ell(n)) \leq T_M(n) \cdot T_{\mathcal{A}}(T_M(n)).$$

The running time $T_{\mathcal{A}}(n)$ of the adversary $\mathcal{A}$ (when given access to a PSPACE-complete oracle) is some fixed polynomial in n, and therefore $T_M(n) \geq 2^{\zeta n}$ for some constant $\zeta > 0$.

Case 2: $2^{n/4} > q(n)$. In this case we have that

$$\epsilon_{M,1}\left(T_{\mathcal{A}}(n)/\epsilon_{\mathcal{A}}(n)\right) \cdot \epsilon_{M,2}(n) \leq \frac{(q(n)+1) \cdot 2^{n/2}}{2^n - q(n) - 1} \leq \frac{1}{2^{n/10}},$$

and since $T_{\mathcal{A}}(n)$ is some fixed polynomial in n (and $\epsilon_{\mathcal{A}}(n)$ is a constant) we obtain that $\epsilon_{M,1}(n^c) \cdot \epsilon_{M,2}(n) \leq 2^{-n/10}$ for some constant $c > 1$.

■

5 One-Way Functions Do Not Imply Bounded-TFNP Hardness

In this section we prove that there is no fully black-box construction of a hard-on-average distribution of TFNP instances having a bounded number of solutions from a one-way function. Our result is obtained by presenting a distribution of oracles relative to which the following two properties hold:

1. There exists a one-way function.
2. There are no hard-on-average distributions of TFNP instances having a bounded number of solutions. Specifically, our result will apply to any sub-exponential number of solutions.

Recall that a TFNP instance with bounded number $k(\cdot)$ of solutions, denoted a k-bounded TFNP instance (see Definitions 2.3 and 2.5), is of the form $\{\mathsf{Gen}_n, C_n\}_{n\in\mathbb{N}}$, where for every $n \in \mathbb{N}$ and for every index σ produced by Gen_n it holds that $C_n(\sigma,\cdot) : \{0,1\}^n \to \{0,1\}$, and there is at least one and at most $k(n)$ distinct inputs $x \in \{0,1\}^n$ such that $C_n(\sigma,x) = 1$ (any one of these x's is a solution). In particular, as discussed in Section 4, any *valid* SVL instance yields a 1-bounded TFNP instance (i.e., a unique-TFNP instance as defined in Section 4), and therefore our result rules out fully black-box constructions of a hard-on-average distribution of SVL instances from a one-way function. Similarly, any source-or-sink instance which consists of at most $(k+1)/2$ disjoint lines yields a k-bounded TFNP instance, and therefore our result rules out fully black-box constructions of a hard-on-average distribution of source-or-sink instances with a bounded number of disjoint lines from a one-way function.

In this section we model a one-way function as a sequence $f = \{f_n\}_{n\in\mathbb{N}}$, where for every $n \in \mathbb{N}$ it holds that $f_n : \{0,1\}^n \to \{0,1\}^n$. The following definition tailors the standard notion of a fully black-box construction to the specific primitives under consideration.

Definition 5.1. A fully black-box construction of a hard-on-average distribution of k-bounded TFNP instances from a one-way function consists of a sequence of polynomial-size oracle-aided circuits $C = \{\mathsf{Gen}_n, C_n\}_{n\in\mathbb{N}}$, an oracle-aided algorithm M that runs in time $T_M(\cdot)$, and functions $\epsilon_{M,1}(\cdot)$ and $\epsilon_{M,2}(\cdot)$, such that the following conditions hold:

- **Correctness**: For any function $f = \{f_n\}_{n\in\mathbb{N}}$, for any $n \in \mathbb{N}$, and for any index σ produced by Gen_n^f, there exists at least one and at most $k(n)$ distinct inputs $x \in \{0,1\}^n$ such that $C_n^f(\sigma,x) = 1$.
- **Black-box proof of security:** For any function $f = \{f_n\}_{n\in\mathbb{N}}$, for any oracle-aided algorithm $\mathcal{A}$ that runs in time $T_{\mathcal{A}} = T_{\mathcal{A}}(n)$, and for any function $\epsilon_{\mathcal{A}}(\cdot)$, if

$$\Pr\left[\mathcal{A}^f\left(1^n,\sigma\right) = x \text{ s.t. } C_n^f(\sigma,x) = 1\right] \geq \epsilon_{\mathcal{A}}(n)$$

  for infinitely many values of $n \in \mathbb{N}$, where the probability is taken over the choice of $\sigma \leftarrow \mathsf{Gen}_n^f()$ and over the internal randomness of $\mathcal{A}$, then

$$\Pr\left[M^{\mathcal{A},f}\left(f_n(x)\right) \in f_n^{-1}\left(f_n(x)\right)\right] \geq \epsilon_{M,1}\left(T_{\mathcal{A}}(n)/\epsilon_{\mathcal{A}}(n)\right)\cdot\epsilon_{M,2}(n)$$

  for infinitely many values of $n \in \mathbb{N}$, where the probability is taken over the choice of $x \leftarrow \{0,1\}^n$ and over the internal randomness of M.

We note that, as in Definitions 3.1 and 4.1, we split the security loss in the above definition to an adversary-dependent security loss and an adversary-independent security loss, as this allows us to capture constructions where one of these losses is super-polynomial whereas the other is polynomial. Equipped with the above definition we prove the following theorem:

Theorem 5.2. *Let $(C, M, T_M, \epsilon_{M,1}, \epsilon_{M,2})$ be a fully black-box construction of a hard-on-average distribution of k-bounded TFNP instances from a one-way function. Then, at least one of the following properties holds:*

1. *$T_M(n) \geq 2^{\zeta n}$ for some constant $\zeta > 0$ (i.e., the reduction runs in exponential time).*
2. *$k(T_M(n)) \geq 2^{n/8}$ (i.e., the number of solutions, as a function of the reduction's running time, is exponential).*
3. *$\epsilon_{M,1}(k(n) \cdot n^c) \cdot \epsilon_{M,2}(n) \leq 2^{-n/2}$ for some constant $c > 1$ (i.e., the security loss is exponential).*

In particular, Theorem 5.2 rules out standard "polynomial-time polynomial-loss" reductions resulting in at most $2^{n^{o(1)}}$ solutions. That is, if $T_M(n)$, $\epsilon_{M,1}(n)$ and $\epsilon_{M,2}(n)$ are all polynomials in n, then the number $k(n)$ of solutions must be at least sub-exponential in n (i.e., $k(n) \geq 2^{n^{\Theta(1)}}$). In addition, if the number $k(n)$ of solutions is constant, the running time $T_M(\cdot)$ of the reduction is sub-exponential, and the adversary-dependent security loss $\epsilon_{M,1}(\cdot)$ is polynomial (all as in [BPR15]), then the adversary-independent security loss $\epsilon_{M,2}(\cdot)$ must be exponential (thus even ruling out constructions based on one-way functions with *sub-exponential* hardness).

5.1 Proof Overview

In what follows we first describe the oracle, denoted f, on which we rely for proving Theorem 5.2. Then, we describe the structure of the proof, showing that relative to the oracle f there exists a one-way function, but there are no hard-on-average bounded-TFNP instances. For the remainder of this section we remind the reader that a *q-query algorithm* is an oracle-aided algorithm $\mathcal{A}$ such that for any oracle $\mathcal{O}$ and input $x \in \{0,1\}^*$, the computation $\mathcal{A}^{\mathcal{O}}(x)$ consists of at most $q(|x|)$ oracle calls to $\mathcal{O}$.

The oracle f. The oracle f is a sequence $\{f_n\}_{n \in \mathbb{N}}$ where for every $n \in \mathbb{N}$ the function $f_n : \{0,1\}^n \to \{0,1\}^n$ is sampled uniformly from the set of all functions mapping n-bit inputs to n-bit outputs.

Part I: f is a one-way function. We prove the following standard claim stating that the oracle f is an exponentially-hard one-way function.

Claim 5.3. *For every $q(n)$-query algorithm M it holds that*

$$\Pr\left[M^f\left(f_n(x)\right) \in f_n^{-1}\left(f_n(x)\right)\right] \leq \frac{2(q(n)+1)}{2^n - q(n)}$$

for all sufficiently large $n \in \mathbb{N}$, where the probability is taken over the choice of $x \leftarrow \{0,1\}^n$, and over the choice of the oracle $f = \{f_n\}_{n \in \mathbb{N}}$ as described above.

Part II: Solving oracle-aided bounded-TFNP instances relative to f. We show that any oracle-aided k-bounded TFNP instance $C = \{C_n\}_{n \in \mathbb{N}}$, where each C_n is a circuit that contains at most $q(n)$ oracle gates, can always be solved by an algorithm that issues roughly $k(n) \cdot q(n)^2$ oracle queries. We prove the following claim:

Claim 5.4. *Let $C = \{\mathsf{Gen}_n, C_n\}_{n \in \mathbb{N}}$ be an oracle-aided $k(n)$-bounded TFNP instance, where Gen_n and C_n are circuits that contain at most $q(n)$ oracle gates each for every $n \in \mathbb{N}$. If C satisfies the correctness requirement stated in Definition 5.1, then there exists an $O(k(n) \cdot q(n)^2)$-query algorithm $\mathcal{A}$ such that*

$$\Pr\left[\mathcal{A}^f\left(1^n, \sigma\right) = x \text{ s.t. } C_n^f(\sigma, x) = 1\right] = 1$$

for all $n \in \mathbb{N}$, where the probability is taken over the choice of the oracle $f = \{f_n\}_{n\in\mathbb{N}}$ as described above and over the choice of $\sigma \leftarrow \mathsf{Gen}_n^f()$. Moreover, the algorithm $\mathcal{A}$ can be implemented in time $k(n) \cdot q(n)^2 \cdot \mathsf{poly}(n)$ given access to a PSPACE*-complete oracle.*

Our proof of Claim 5.4, which is provided in Section 5.3, is obtained by further generalizing our extension of Rudich's classic proof technique [Rud88]. As discussed in Section 4.1, by extending and refining Rudich's proof technique once again, we show that his approach allows to rule out even constructions of bounded-TFNP instances.

5.2 f is a One-Way Function

Proof of Claim 5.3. Let M be a $q(n)$-query algorithm, fix $n \in \mathbb{N}$, and fix $(f)_{-n} = \{f_i\}_{i\in\mathbb{N}\setminus\{n\}}$ (i.e., we fix the entire oracle f except for the nth function f_n). Without loss of generality, by viewing M as a $(q(n)+1)$-query algorithm, we may assume that M always queries f_n with its output. For any $y \in \{0,1\}^n$ and for every $i \in [q(n)+1]$ denote by $\alpha_i(y)$ the random variable corresponding to the ith query made by M to f_n when M is given y as input (note that since we do not place any restriction on the running time of M we can assume without loss of generality that M is deterministic). Therefore,

$$\begin{aligned}\Pr\left[M^f(y) \in f_n^{-1}(y)\right] \leq{}& \Pr\left[\alpha_1(y) \in f_n^{-1}(y)\right] \\ &+ \sum_{i=1}^{q(n)} \Pr\left[\alpha_{i+1}(y) \in f_n^{-1}(y) \,\middle|\, \alpha_1(y),\ldots,\alpha_i(y) \notin f_n^{-1}(y)\right],\end{aligned}$$

where $y = f_n(x)$, and the probability is taken over the choice of the nth function $f_n : \{0,1\}^n \to \{0,1\}^n$ and over the choice of $x \leftarrow \{0,1\}^n$.

For bounding the probability of the event $\alpha_1(y) \in f_n^{-1}(y)$, note that this event corresponds to the fact that M, when given input $y = f_n(x)$ and without any information on the uniformly chosen $x \in \{0,1\}^n$, manages to produce an input $\alpha_1(y)$ that f_n maps to y. If $\alpha_1(y) = x$ then clearly $\alpha_1(y) \in f_n^{-1}(y)$ (but this happen with probability 2^{-n} since x is still uniform from M's point of view), and if $\alpha_1(y) \neq x$ then the value $f_n(\alpha_1(y))$ is completely independent of $f_n(x)$ and therefore uniformly distributed over $\{0,1\}^n$. Therefore,

$$\begin{aligned}\Pr\left[\alpha_1(y) \in f_n^{-1}(y)\right] &\leq \Pr\left[\alpha_1(y) = x\right] + \Pr\left[\alpha_1(y) \in f_n^{-1}(y) \,\middle|\, \alpha_1(y) \neq x\right] \\ &= \frac{1}{2^n} + \frac{1}{2^n} \\ &= \frac{2}{2^n}.\end{aligned}$$

For bounding the probability of the event $\alpha_{i+1}(y) \in f_n^{-1}(y)$ conditioned on $\alpha_1(y),\ldots,\alpha_i(y) \notin f_n^{-1}(y)$ we follow a similar argument. Without loss of generality, we assume that $\alpha_1(y),\ldots,\alpha_{i+1}(y)$ are all distinct, and then it holds that:

- Given that $\alpha_1(y),\ldots,\alpha_i(y) \notin f_n^{-1}(y)$ then from M's point of view the value x is uniformly distributed over the set $\{0,1\}^n \setminus \{\alpha_1(y),\ldots,\alpha_i(y)\}$. Therefore

$$\Pr\left[\alpha_{i+1}(y) = x \,\middle|\, \alpha_1(y),\ldots,\alpha_i(y) \notin f_n^{-1}(y)\right] = \frac{1}{2^n - i}.$$

- Given that $\alpha_1(y),\ldots,\alpha_i(y) \notin f_n^{-1}(y)$ and $\alpha_{i+1}(y) \neq x$, we have that $\alpha_{i+1}(y) \notin \{x, \alpha_1(y),\ldots,\alpha_i(y)\}$ based on our assumption that the queries are all distinct. This implies that the random

variable $f_n(\alpha_{i+1}(y))$ is completely independent of $f_n(x), f_n(\alpha_1(y)), \ldots, f_n(\alpha_i(y))$ and therefore uniformly distributed over $\{0,1\}^n$. That is,

$$\Pr\left[\alpha_{i+1}(y) \in f_n^{-1}(y) \,\middle|\, \alpha_1(y), \ldots, \alpha_i(y) \notin f_n^{-1}(y) \wedge \alpha_{i+1}(y) \neq x\right] = \frac{1}{2^n}.$$

Therefore,

$$\begin{aligned}
&\Pr\left[\alpha_{i+1}(y) \in f_n^{-1}(y) \,\middle|\, \alpha_1(y), \ldots, \alpha_i(y) \notin f_n^{-1}(y)\right] \\
&\qquad \leq \Pr\left[\alpha_{i+1}(y) = x \,\middle|\, \alpha_1(y), \ldots, \alpha_i(y) \notin f_n^{-1}(y)\right] \\
&\qquad\quad + \Pr\left[\alpha_{i+1}(y) \in f_n^{-1}(y) \,\middle|\, \alpha_1(y), \ldots, \alpha_i(y) \notin f_n^{-1}(y) \wedge \alpha_{i+1}(y) \neq x\right] \\
&\qquad = \frac{1}{2^n - i} + \frac{1}{2^n} \\
&\qquad \leq \frac{2}{2^n - i}.
\end{aligned}$$

We conclude that

$$\begin{aligned}
\Pr\left[M^f(y) \in f_n^{-1}(y)\right] &\leq \sum_{i=0}^{q(n)} \frac{2}{2^n - i} \\
&\leq \frac{2(q(n)+1)}{2^n - q(n)}.
\end{aligned}$$

■

5.3 Solving Oracle-Aided Bounded-TFNP Instances Relative to f

Proof of Claim 5.4. Fix the oracle $f = \{f_n\}_{n\in\mathbb{N}}$ and let $C = \{\mathsf{Gen}_n, C_n\}_{n\in\mathbb{N}}$ be an oracle-aided k-bounded TFNP instance that satisfies the correctness requirement stated in Definition 5.1, where each of Gen_n and C_n contains at most $q(n)$ oracle gates for any $n \in \mathbb{N}$. Consider the following oracle-aided algorithm $\mathcal{A}$ that on input an index σ produced by Gen_n^f would like to find an input $x \in \{0,1\}^n$ such that $C_n^f(\sigma, x) = 1$. The algorithm $\mathcal{A}$ initializes an empty set Q, which at any point in time will contain pairs of the form (α, β) where $\beta = f(\alpha)$ (i.e., the set Q is always consistent with f). The algorithm $\mathcal{A}$ performs the following steps for $q(n)+1$ iterations:

Step 1. The algorithm $\mathcal{A}$ finds a function g subject to the following three requirements:

- g is consistent with Q.
- There exists a value $r \in \{0,1\}^*$ such that $\mathsf{Gen}_n^g(r) = \sigma$.
- Among all functions g that satisfy the first two requirements, choose the one that maximizes the number k_g of solutions to the instance $C_n^g(\sigma, \cdot)$ (i.e., k_g is the number of distinct inputs $x \in \{0,1\}^n$ such that $C_n^g(\alpha, x) = 1$).

Step 2. The algorithm $\mathcal{A}$ finds a value $r \in \{0,1\}^*$ such that $\mathsf{Gen}_n^g(r) = \sigma$, and finds the distinct inputs $x_1, \ldots, x_{k_g} \in \{0,1\}^n$ for which $C_n^g(\sigma, x_i) = 1$ for every $i \in [k_g]$.

Step 3. For every $i \in [k_g]$, the algorithm $\mathcal{A}$ computes $C_n^f(\sigma, x_i)$. If there exists an $i \in [k_g]$ for which $C_n^f(\sigma, x_i) = 1$, then $\mathcal{A}$ outputs the first such x_i and terminates.

Step 4. The algorithm $\mathcal{A}$ queries f with all inputs to g-gates in the computations $\mathsf{Gen}_n^g(r), C_n^g(\sigma, x_1), \ldots, C_n^g(\sigma, x_{k_g})$, and adds these queries to the set Q.

If the algorithm $\mathcal{A}$ did not return an output during the above iterations, then it outputs $\perp$. In terms of the number of oracle queries made by $\mathcal{A}$, observe that steps 1 and 2 do not require any oracle queries, while each of steps 3 and 4 require at most $q + k \cdot q$ queries[7]. Therefore, the total number of queries made by $\mathcal{A}$ is at most $2(k+1) \cdot q(q+1) = O(k \cdot q^2)$, as required.

Moreover, given oracle access a PSPACE-complete oracle, the algorithm $\mathcal{A}$ can be implemented to run in time $k(n) \cdot q(n)^2 \cdot \mathsf{poly}(n)$. To see this, we observe that there exists an oracle g that satisfies the requirements in step 1 and can be described in space polynomial in $q(n)$ and $k(n)$. This is because the only queries of g that matter are those in the set Q and the queries performed in the computations $\mathsf{Gen}_n^g(r)$, $C_n^g(\sigma, x_1), \ldots, C_n^g(\sigma, x_{k_g})$. Therefore, the algorithm can be efficiently computed using an oracle that decides the following PSPACE language:

$$\left\{ (\sigma, Q, i, b) \;\middle|\; \begin{array}{c} \text{The } i\text{th bit of } (x_1, \ldots, x_{k_g}, Q') \text{ is } b, \text{ where } (g, r) \text{ is the lexicographically first} \\ \text{tuple satisfying the requirements in step 1 with respect to } (\sigma, Q), \\ x_1, \ldots, x_{k_g} \text{ are the inputs decribed in step 2,} \\ \text{and } Q' \text{ is the set of queries described in step 4} \end{array} \right\},$$

which allows discovering the inputs $x_1, \ldots, x_{k_g}$ required in step 3, as well as the queries required in step 4.

In the remainder of this proof, we show that $\mathcal{A}$ is always successful in one of its $q+1$ iterations. This follows from the following claim:

Claim 5.5. *Fix any $x^* \in \{0,1\}^n$ such that $C_n^f(\sigma, x^*) = 1$. Then, in each iteration, at least one of the following events occur:*

- *During step 3 of the iteration $\mathcal{A}$ finds an input $x \in \{0,1\}^n$ such that $C_n^f(\sigma, x) = 1$.*
- *During step 4 of the iteration $\mathcal{A}$ adds to Q a new f-query that is performed in the computation $C_n^f(\sigma, x^*)$.*

We now show that Claim 5.5 indeed guarantees that $\mathcal{A}$ is always successful when repeating steps 1–4 above for $q+1$ iterations. Fix any $x^* \in \{0,1\}^n$ such that $C_n^f(\sigma, x^*) = 1$, and assume that in the first q iterations $\mathcal{A}$ does not find a solution in step 3. Claim 5.5 implies that in each of these q iterations $\mathcal{A}$ adds to the set Q a new f-query that is performed in the computation $C_n^f(\sigma, x^*)$. Since this computation contains at most q oracle queries to f, at the end of the first q iterations we are guaranteed that all of these queries are included in the set Q. Therefore, in the final iteration, for any g that will be chosen in step 1 it holds that $C_n^g(\sigma, x^*) = 1$ since g is chosen to be consistent with Q. Thus, in this iteration $x^* \in \{x_1, \ldots, x_{k_g}\}$, and therefore there exists at least one index $i \in [k_g]$ for which $C_n^f(\sigma, x_i) = 1$, which implies that $\mathcal{A}$ outputs a solution. We now conclude the proof of Claim 5.4 by proving Claim 5.5.

Proof of Claim 5.5. Fix any $x^* \in \{0,1\}^n$ such that $C_n^f(\sigma, x^*) = 1$, and assume towards a contradiction that in some iteration $j \in [q+1]$ the following two events occur:

- During step 3 of the iteration $\mathcal{A}$ does not find an input $x \in \{0,1\}^n$ such that $C_n^f(\sigma, x) = 1$. In particular, this implies that $x^* \notin \{x_1, \ldots, x_{k_g}\}$.
- During step 4 of the iteration $\mathcal{A}$ does not add to Q a new f-query that is performed in the computation $C_n^f(\sigma, x^*)$. That is, all inputs to g-queries in the computations $\mathsf{Gen}_n^g(r), C_n^g(\sigma, x_1), \ldots, C_n^g(\sigma, x_{k_g})$ are either already in the set Q at the beginning of the jth iteration, or are not used as inputs to f-queries in the computation $C_n^f(\sigma, x^*)$.

[7] Since Q is always consistent with f, and since C is a k-bounded TFNP instance, then in each iteration it holds that $k_f \leq k_g \leq k$.

We now show that, in fact, at the beginning of the jth iteration there was a function g' such that: (1) g' is consistent with Q, (2) σ is a valid index produced by $\mathsf{Gen}_n^{g'}$, and (3) there are at least $k_g + 1$ inputs $x \in \{0,1\}^n$ for which $C_n^{g'}(\sigma, x) = 1$. This contradicts the fact that, in step 1 of the jth iteration, $\mathcal{A}$ chose g that maximizes the number k_g of solutions to the instance $C_n^g(\cdot)$ among all functions that are consistent with Q and for which σ is a valid index. The function g' is defined as follows (according to the following 4 types of possible inputs):

- **Type 1 inputs:** For every input α that appears in the set Q we set $g'(\alpha) = f(\alpha)$.

  Note that since Q is consistent with f, and g is consistent with Q, then for all type 1 inputs α it holds that $g'(\alpha) = f(\alpha) = g(\alpha)$.

- **Type 2 inputs:** For every input α that is used as input to an f-query in the computation $C_n^f(\sigma, x^*)$ and is not in the set Q, we set $g'(\alpha) = f(\alpha)$.

- **Type 3 inputs:** For every input α that is used as input to a g-query in the computations $\mathsf{Gen}_n^g(r), C_n^g(\sigma, x_1), \ldots, C_n^g(\sigma, x_{k_g})$ and is not in the set Q, we set $g'(\alpha) = g(\alpha)$.

- **Type 4 inputs:** For any other input α we set $g'(\alpha)$ to an arbitrary value.

For the function g' it holds that $\mathsf{Gen}_n^{g'}(r) = \mathsf{Gen}_n^g(r) = \sigma$, $C_n^{g'}(\sigma, x^*) = C_n^f(\sigma, x^*) = 1$, and $C_n^{g'}(\sigma, x_i) = C_n^g(\sigma, x_i) = 1$ for every $i \in [k_g]$. Thus, the values $x^*, x_1, \ldots, x_{k_g}$ are $k_g + 1$ distinct solutions to the instance $C_n^{g'}(\sigma, \cdot)$. ■

This settles the proof of Claim 5.4. ■

5.4 Proof of Theorem 5.2

Proof of Theorem 5.2. Let $(C, M, T_M, \epsilon_{M,1}, \epsilon_{M,2})$ be a fully black-box construction of a hard-on-average distribution of k-bounded TFNP instances from a one-way function (recall Definition 5.1), where $C = \{\mathsf{Gen}_n, C_n\}_{n\in\mathbb{N}}$. Claim 5.4 guarantees an oracle-aided algorithm $\mathcal{A}$ that runs in time $T_{\mathcal{A}}(n) = k(n) \cdot \mathsf{poly}(n)$ such that

$$\Pr\left[\mathcal{A}^{\mathsf{PSPACE},f}(1^n, \sigma) = x \text{ s.t. } C_n^f(\sigma, x) = 1\right] = \epsilon_{\mathcal{A}}(n)$$

for all $n \in \mathbb{N}$, where $\epsilon_{\mathcal{A}}(n) = 1$, and the probability is taken over the choice of the oracle $f = \{f_n\}_{n\in\mathbb{N}}$ and over the choice of $\sigma \leftarrow \mathsf{Gen}_n^f()$. Definition 5.1 then guarantees that

$$\Pr\left[M^{\mathcal{A},\mathsf{PSPACE},f}(f_n(x)) \in f_n^{-1}(f_n(x))\right] \geq \epsilon_{M,1}\left(T_{\mathcal{A}}(n)/\epsilon_{\mathcal{A}}(n)\right) \cdot \epsilon_{M,2}(n)$$

for infinitely many values of $n \in \mathbb{N}$, where M runs in time $T_M(n)$, and the probability is taken over the choice of the oracle $f = \{f_n\}_{n\in\mathbb{N}}$ and over the choice of $x \leftarrow \{0,1\}^n$.

The algorithm M may invoke $\mathcal{A}$ on various security parameters (i.e., in general M is not restricted to invoking $\mathcal{A}$ only on security parameter n), and we denote by $\ell(n)$ the maximal security parameter on which M invokes $\mathcal{A}$ (when M itself is invoked on security parameter n). Thus, viewing $M^{\mathcal{A}}$ as a single oracle-aided algorithm that has access to a PSPACE-complete oracle and to the oracle f, its running time $T_{M^{\mathcal{A}}}(n)$ satisfies $T_{M^{\mathcal{A}}}(n) \leq T_M(n) \cdot T_{\mathcal{A}}(\ell(n))$ (this follows since M may invoke $\mathcal{A}$ at most $T_M(n)$ times, and the running time of $\mathcal{A}$ on each such invocation is at most $T_{\mathcal{A}}(\ell(n))$). In particular, viewing $M' \stackrel{\mathsf{def}}{=} M^{\mathcal{A}^{\mathsf{PSPACE}}}$ as a single oracle-aided algorithm that has oracle access to the

oracle f, implies that M' is a $q(n)$-query algorithm where $q(n) = T_{M^{\mathcal{A}}}(n)$. Claim 5.3 then implies that

$$\epsilon_{M,1}\left(T_{\mathcal{A}}(n)/\epsilon_{\mathcal{A}}(n)\right) \cdot \epsilon_{M,2}(n) \leq \frac{2(q(n)+1)}{2^n - q(n)}.$$

There are now two possible cases to consider:

Case 1: $2^{n/4} \leq q(n)$. In this case, noting that $\ell(n) \leq T_M(n)$, we obtain that

$$2^{n/4} \leq q(n) = T_{M^{\mathcal{A}}}(n) \leq T_M(n) \cdot T_{\mathcal{A}}(\ell(n)) \leq T_M(n) \cdot T_{\mathcal{A}}(T_M(n)).$$

Since $T_{\mathcal{A}}(n) = k(n) \cdot \mathsf{poly}(n)$ for some fixed polynomial $\mathsf{poly}(n)$, then it holds that

$$2^{n/4} \leq k\left(T_M(n)\right) \cdot \mathsf{poly}(T_M(n))$$

which implies that either $k(T_M(n)) \geq 2^{n/8}$ or $T_M(n) \geq 2^{\zeta n}$ for some constant $\zeta > 0$.

Case 2: $2^{n/4} > q(n)$. In this case we have that

$$\epsilon_{M,1}\left(T_{\mathcal{A}}(n)/\epsilon_{\mathcal{A}}(n)\right) \cdot \epsilon_{M,2}(n) \leq \frac{2(q(n)+1)}{2^n - q(n)} \leq \frac{1}{2^{n/2}},$$

and since $T_{\mathcal{A}}(n) = k(n) \cdot \mathsf{poly}(n)$ for some fixed polynomial $\mathsf{poly}(n)$ (and $\epsilon_{\mathcal{A}}(n) = 1$) we obtain that $\epsilon_{M,1}(k(n) \cdot n^c) \cdot \epsilon_{M,2}(n) \leq 2^{-n/2}$ for some constant $c > 1$. ■

6 Public-Key Cryptography Does Not Imply Bounded-TFNP Hardness

In this section we generalize the result proved in Section 5 from considering a one-way function to considering a collection of injective trapdoor functions as the underlying building block. This proves, in particular, Theorem 1.4 and Corollary 1.5. Specifically, we prove that there is no fully black-box construction of a hard-on-average distribution of TFNP instances having a bounded number of solutions from a collection of injective trapdoor functions. Our result is obtained by presenting a distribution of oracles relative to which the following two properties hold:

1. There exists a collection of injective trapdoor functions.
2. There are no hard-on-average distributions of TFNP instances having a bounded number of solutions. Specifically, our result will apply to any sub-exponential number of solutions, exactly as in Section 5.

From the technical perspective, instead of considering an oracle $f = \{f_n\}_{n\in\mathbb{N}}$ where for every $n \in \mathbb{N}$ the function $f_n : \{0,1\}^n \to \{0,1\}^n$ is sampled uniformly, we consider a more structured oracle, $\mathcal{O}_{\mathsf{TDF}}$, corresponding to a collection of injective trapdoor functions. Proving that the oracle $\mathcal{O}_{\mathsf{TDF}}$ is indeed hard to invert is quite standard (based, for example, on the approach of Haitner et al. [HHR+15]). However, showing that relative to the oracle $\mathcal{O}_{\mathsf{TDF}}$ we can solve bounded-TFNP instances is significantly more challenging than the corresponding proof relative to the oracle f.

We say that $\tau = \left\{\left(\mathsf{KG}_n, \mathsf{F}_n, \mathsf{F}_n^{-1}\right)\right\}_{n\in\mathbb{N}}$ is a collection of injective trapdoor functions if for every $n \in \mathbb{N}$ and for every pair $(\mathsf{td}, \mathsf{pk})$ produced by $\mathsf{KG}_n()$, the function $\mathsf{F}_n(\mathsf{pk}, \cdot) : \{0,1\}^n \to \{0,1\}^m$ is injective (for some $m \geq n$) and the function $\mathsf{F}_n^{-1}(\mathsf{td}, \cdot)$ computes it inverse whenever an inverse exists (i.e., it outputs $\perp$ on all values y that are not in the image of the function $\mathsf{F}_n(\mathsf{pk}, \cdot)$) – see Section 2.2 for more details. The following definition tailors the standard notion of a fully black-box construction to the specific primitives under consideration.

Definition 6.1. A fully black-box construction of a hard-on-average distribution of k-bounded TFNP instances from a collection of injective trapdoor functions consists of a sequence of polynomial-size oracle-aided circuits $C = \{\mathsf{Gen}_n, C_n\}_{n\in\mathbb{N}}$, an oracle-aided algorithm M that runs in time $T_M(\cdot)$, and functions $\epsilon_{M,1}(\cdot)$ and $\epsilon_{M,2}(\cdot)$, such that the following conditions hold:

- **Correctness**: For any collection τ of injective trapdoor functions, for any $n \in \mathbb{N}$, and for any index σ produced by Gen_n^τ, there exists at least one and at most $k(n)$ distinct inputs $x \in \{0,1\}^n$ such that $C_n^\tau(\sigma, x) = 1$.
- **Black-box proof of security:** For any collection $\tau = \left\{\left(\mathsf{KG}_n, \mathsf{F}_n, \mathsf{F}_n^{-1}\right)\right\}_{n\in\mathbb{N}}$ of injective trapdoor functions, for any oracle-aided algorithm $\mathcal{A}$ that runs in time $T_\mathcal{A} = T_\mathcal{A}(n)$, and for any function $\epsilon_\mathcal{A}(\cdot)$, if
$$\Pr\left[\mathcal{A}^\tau\left(1^n, \sigma\right) = x \text{ s.t. } C_n^\tau(\sigma, x) = 1\right] \geq \epsilon_\mathcal{A}(n)$$
for infinitely many values of $n \in \mathbb{N}$, where the probability is taken over the choice of $\sigma \leftarrow \mathsf{Gen}_n^\tau()$ and $x \leftarrow \{0,1\}^n$, and over the internal randomness of $\mathcal{A}$, then
$$\Pr\left[M^{\mathcal{A},\tau}\left(\mathsf{pk}, \mathsf{F}_n(\mathsf{pk}, x)\right) = x\right] \geq \epsilon_{M,1}\left(T_\mathcal{A}(n)/\epsilon_\mathcal{A}(n)\right) \cdot \epsilon_{M,2}(n)$$
for infinitely many values of $n \in \mathbb{N}$, where the probability is taken over the choice of $(\mathsf{td}, \mathsf{pk}) \leftarrow \mathsf{KG}_n()$, $x \leftarrow \{0,1\}^n$, and over the internal randomness of M.

We note that, as in Definitions 3.1, 4.1 and 5.1, we split the security loss in the above definition to an adversary-dependent security loss and an adversary-independent security loss, as this allows us to capture constructions where one of these losses is super-polynomial whereas the other is polynomial. Equipped with the above definition we prove the following theorem (generalizing Theorem 5.2):

Theorem 6.2. *Let $(C, M, T_M, \epsilon_{M,1}, \epsilon_{M,2})$ be a fully black-box construction of a hard-on-average distribution of k-bounded TFNP instances from a collection of injective trapdoor functions. Then, at least one of the following properties holds:*

1. *$T_M(n) \geq 2^{\zeta n}$ for some constant $\zeta > 0$ (i.e., the reduction runs in exponential time).*
2. *$k(T_M(n)) \geq 2^{n/8}$ (i.e., the number of solutions, as a function of the reduction's running time, is exponential).*
3. *$\epsilon_{M,1}(k(n) \cdot n^c) \cdot \epsilon_{M,2}(n) \leq 2^{-n/2}$ for some constant $c > 1$ (i.e., the security loss is exponential).*

In particular, and similarly to Theorem 5.2, Theorem 6.2 rules out standard "polynomial-time polynomial-loss" reductions resulting in at most $2^{n^{o(1)}}$ solutions. That is, if $T_M(n)$, $\epsilon_{M,1}(n)$ and $\epsilon_{M,2}(n)$ are all polynomials in n, then the number $k(n)$ of solutions must be at least sub-exponential in n (i.e., $k(n) \geq 2^{n^{\Theta(1)}}$). In addition, if the number $k(n)$ of solutions is constant, the running time $T_M(\cdot)$ of the reduction is sub-exponential, and the adversary-dependent security loss $\epsilon_{M,1}(\cdot)$ is polynomial (all as in [BPR15]), then the adversary-independent security loss $\epsilon_{M,2}(\cdot)$ must be exponential (thus even ruling out constructions based on one-way functions with *sub-exponential* hardness). Given our claims in the remainder of this section, the proof of Theorem 6.2 is derived in a nearly identical to proof of 5.2, and is therefore omitted.

6.1 Proof Overview

In what follows we first describe the oracle, denoted $\mathcal{O}_{\mathsf{TDF}}$, on which we rely for proving Theorem 6.2. Then, we describe the structure of the proof, and explain the main challenges in generalizing our proof from Section 5.

The oracle $\mathcal{O}_{\mathsf{TDF}}$. The oracle $\mathcal{O}_{\mathsf{TDF}}$ is a sequence of the form $\left\{\left(\mathsf{G}_n, \mathsf{F}_n, \mathsf{F}_n^{-1}\right)\right\}_{n\in\mathbb{N}}$ that is sampled via the following process for every $n \in \mathbb{N}$:

- The function $\mathsf{G}_n : \{0,1\}^n \to \{0,1\}^{2n}$ is sampled uniformly from the set of all functions mapping n-bit inputs to n-bit outputs.
- For every $\mathsf{pk} \in \{0,1\}^n$ the function $\mathsf{F}_n(\mathsf{pk}, \cdot) : \{0,1\}^n \to \{0,1\}^{2n}$ is sampled uniformly from the set of all *injective* functions mapping n-bit inputs to $2n$-bit outputs.
- For every $\mathsf{td} \in \{0,1\}^n$ and $y \in \{0,1\}^{2n}$ we set

$$\mathsf{F}_n^{-1}(\mathsf{td}, y) = \begin{cases} x & \text{if } \mathsf{F}_n(\mathsf{G}_n(\mathsf{td}), x) = y \\ \perp & \text{if no such } x \text{ exists} \end{cases} .$$

Part I: $\mathcal{O}_{\mathsf{TDF}}$ is a hard-to-invert collection of injective trapdoor functions. We show that the oracle $\mathcal{O}_{\mathsf{TDF}}$ naturally defines a hard-on-average collection of injective trapdoor functions. Specifically, the key-generation algorithm on input 1^n samples $\mathsf{td} \leftarrow \{0,1\}^n$ uniformly at random, and computes $\mathsf{pk} = \mathsf{G}_n(\mathsf{td})$ (where F_n and F_n^{-1} are used as the evaluation and inversion algorithms). We prove the following claim stating that collection of injective trapdoor functions is exponentially secure.

Claim 6.3. *For every $q(n)$-query algorithm M it holds that*

$$\Pr\left[M^{\mathcal{O}_{\mathsf{TDF}}}\left(\mathsf{G}_n(\mathsf{td}), \mathsf{F}_n(\mathsf{G}_n(\mathsf{td}), x)\right) = x\right] \leq \frac{4(q(n)+1)}{2^n - q(n)}$$

for all sufficiently large $n \in \mathbb{N}$, where the probability is taken over the choice of $\mathsf{td} \leftarrow \{0,1\}^n$, $x \leftarrow \{0,1\}^n$, and the oracle $\mathcal{O}_{\mathsf{TDF}} = \{(\mathsf{G}_n, \mathsf{F}_n, \mathsf{F}_n^{-1})\}_{n\in\mathbb{N}}$.

The proof of Claim 6.3, which is provided in Section 6.2, is based on the observation that the inversion oracle F_n^{-1} is not very useful. Specifically, the function G_n itself is uniformly chosen and thus hard to invert, and therefore any algorithm M that is given as input $(\mathsf{pk}, \mathsf{F}_n(\mathsf{pk}, x))$ should not be able to find the trapdoor td corresponding to $\mathsf{pk} = \mathsf{G}_n(\mathsf{td})$. Combining this with the fact that the function $\mathsf{F}_n(\mathsf{pk}, \cdot)$ is uniformly chosen and *length doubling*, such an algorithm M should not be able to find any y in its image, unless y was obtained as the result of a previous query (and, in this case, its inverse is already known). Therefore, the task of computing x given $(\mathsf{pk}, \mathsf{F}_n(\mathsf{pk}, x))$ essentially reduces to that of inverting a uniformly-sampled injective function.

Part II: Solving oracle-aided bounded-TFNP instances relative to $\mathcal{O}_{\mathsf{TDF}}$. We show that any oracle-aided k-bounded TFNP instance $C = \{\mathsf{Gen}_n, C_n\}_{n\in\mathbb{N}}$, where Gen_n and C_n contain at most $q(n)$ oracle gates, and the input to each such gate is of length at most $q(n)$ bits, can always be solved with constant probability by an algorithm that issues roughly $k(n)^3 \cdot q(n)^9$ oracle queries. We prove the following claim:

Claim 6.4. *Let $C = \{\mathsf{Gen}_n, C_n\}_{n\in\mathbb{N}}$ be an oracle-aided k-bounded TFNP instance, where for every $n \in \mathbb{N}$ it holds that Gen_n and C_n are circuits that contain at most $q(n)$ oracle gates, and the input*

to each such gate is of length at most $q(n)$ bits. If C satisfies the correctness requirement stated in Definition 6.1, then there exists a $O(q(n)^9 \cdot k(n)^3)$-query algorithm $\mathcal{A}$ such that

$$\Pr\left[\mathcal{A}^{\mathcal{O}_{\mathsf{TDF}}}(1^n, \sigma) = x \text{ s.t. } C_n^{\mathcal{O}_{\mathsf{TDF}}}(\sigma, x) = 1\right] \geq \frac{1}{2}$$

for all $n \in \mathbb{N}$, where the probability is taken over the choice of the oracle $\mathcal{O}_{\mathsf{TDF}} = \{(\mathsf{G}_n, \mathsf{F}_n, \mathsf{F}_n^{-1})\}_{n\in\mathbb{N}}$ as described above and over the choice of $\sigma \leftarrow \mathsf{Gen}_n^{\mathcal{O}_{\mathsf{TDF}}}()$. Moreover, the algorithm $\mathcal{A}$ can be implemented in time $q(n)^9 \cdot k(n)^3 \cdot \mathsf{poly}(n)$ given access to a PSPACE*-complete oracle.*

The proof of Claim 6.4, which is provided in Section 6.3, generalizes the proof of Claim 5.4 (which holds relative to the oracle f defined in Section 5). Recall that for the proof of Claim 5.4 we introduced an adversary that runs for $q+1$ iterations, with the goal of discovering a new oracle query from the computation $C_n^f(\sigma, x^*)$ in each iteration where x^* is any fixed solution of the instance $C_n^f(\sigma, \cdot)$. This approach is based on the observation if no progress is made then there exists an oracle g' for which the instance $C_n^{g'}(\sigma, \cdot)$ has too many solutions. The oracle oracle g' can be constructed by "pasting together" partial information on the actual oracle f with full information on an additional oracle g that is partially-consistent with f.

When dealing with the oracle $\mathcal{O}_{\mathsf{TDF}}$, which is clearly more structured than just a single random function f, this argument becomes much more subtle. One may hope to follow a similar iteration-based approach and argue that if no progress is made then there exists an oracle $\mathcal{O}'_{\mathsf{TDF}}$ for which the instance $C_n^{\mathcal{O}'_{\mathsf{TDF}}}(\sigma, \cdot)$ has too many solutions. However, "pasting together" partial information on the actual oracle $\mathcal{O}_{\mathsf{TDF}}$ with full information on an additional injective trapdoor function oracle that is partially-consistent with $\mathcal{O}_{\mathsf{TDF}}$ may completely fail, as the resulting oracle may not turn out injective at all.

Our main observation is that although pasting together the two oracles may not always work (as in Section 5), it does work with high probability over the choice of the oracle $\mathcal{O}_{\mathsf{TDF}}$. By closely examining the way the two oracles are combined, we show that if the resulting oracle is not a valid collection of injective trapdoor functions, then one of the following "bad" events must have occurred:

- The adversary was able to "guess" an element pk for which there exists td such that $\mathsf{pk} = \mathsf{G}_n(\mathsf{td})$ without previously querying G_n with td.
- The adversary was able to "guess" a public key pk and an element y for which there exists an input x such that $y = \mathsf{F}_n(\mathsf{pk}, x)$ without previously querying F_n with (pk, x).

We show that the probability of each of these two events is small, as we choose both G_n and all functions $\mathsf{F}_n(\mathsf{pk}, \cdot)$ to be length increasing and uniformly distributed.

6.2 $\mathcal{O}_{\mathsf{TDF}}$ is a Collection of Injective Trapdoor Functions

The proof of Claim 6.3, as discussed above, is based on the observation that the inversion oracle F_n^{-1} is not very useful. Specifically, we show that with high probability the behavior of F_n^{-1} is predictable, which means that it can be simulated without actually calling the oracle. In more details, for an oracle-aided algorithm M we denote by $\mathsf{HitInv}_{M,n}^{\mathcal{O}_{\mathsf{TDF}}}$ the event in which the computation $M^{\mathcal{O}_{\mathsf{TDF}}}$ manages to call F_n^{-1} with an input (td, y) which results with $x \neq \perp$ without previously calling F_n with $(\mathsf{G}_n(\mathsf{td}), x)$. We prove the following claim:

Claim 6.5. *For every q-query algorithm M and for every $n \in \mathbb{N}$ it holds that*

$$\Pr\left[\mathsf{HitInv}_{M,n}^{\mathcal{O}_{\mathsf{TDF}}}\right] \leq \frac{q}{2^n - q},$$

where the probability is taken over the choice of the oracle $\mathcal{O}_{\mathsf{TDF}} = \left\{\left(\mathsf{G}_i, \mathsf{F}_i, \mathsf{F}_i^{-1}\right)\right\}_{i\in\mathbb{N}}$ *as described above. Moreover,* q *can be a bound on the number of calls to* F_n *and* F_n^{-1} *only.*

This intuitively means that the access to the oracle F_n^{-1} does not strengthen the power of M by much, because with high probability it can be simulated by answering $\perp$ for every query to F_n^{-1} that cannot be determined by previous queries to F_n.

Proof. Let M be a q-query algorithm, fix $(\mathcal{O}_{\mathsf{TDF}})_{-n} = \{(\mathsf{G}_i, \mathsf{F}_i, \mathsf{F}_i^{-1})\}_{i\in\mathbb{N}\setminus\{n\}}$ (i.e., we fix the entire oracle $\mathcal{O}_{\mathsf{TDF}}$ except for the nth instance), and fix $\mathsf{G}_n : \{0,1\}^n \to \{0,1\}^{2n}$. Thus, we only consider queries to the oracles F_n and F_n^{-1}. For every $i \in [q]$ denote by M_i the following i-query algorithm: Invoke the computation $M^{\mathcal{O}_{\mathsf{TDF}}}$, and terminate once i oracle queries have been performed. Note that since we do not place any restriction on the running time of M and since the oracle distribution is known, we can assume without loss of generality that M is deterministic. Therefore, for every $i \in [q]$ and every fixing of the oracle $\mathcal{O}_{\mathsf{TDF}}$, the computation $M_i^{\mathcal{O}_{\mathsf{TDF}}}$ is the "prefix" of the computation $M^{\mathcal{O}_{\mathsf{TDF}}}$ which contains its first i oracle queries. This implies that

$$\Pr\left[\mathsf{HitInv}_{M,n}^{\mathcal{O}_{\mathsf{TDF}}}\right] \leq \Pr\left[\mathsf{HitInv}_{M_1,n}^{\mathcal{O}_{\mathsf{TDF}}}\right] + \sum_{i=1}^{q-1} \Pr\left[\mathsf{HitInv}_{M_{i+1},n}^{\mathcal{O}_{\mathsf{TDF}}} \middle| \overline{\mathsf{HitInv}_{M_i,n}^{\mathcal{O}_{\mathsf{TDF}}}}\right],$$

where the probability is taken over the choice of F_n.

For bounding the probability of the event $\mathsf{HitInv}_{M_1,n}^{\mathcal{O}_{\mathsf{TDF}}}$, note that this event corresponds to the fact that M, without any information on $\mathsf{F}_n : \{0,1\}^{2n} \times \{0,1\}^n \to \{0,1\}^{2n}$ (since no oracle queries have been issued so far), manages to produce an oracle query to F_n^{-1} of the form $q_1 = (\mathsf{td}_1, y_1)$ where there exists x such that $\mathsf{F}_n(\mathsf{G}_n(\mathsf{td}_1), x) = y_1$.

Since the value q_1 is fixed by the description of M, and we are now sampling $\mathsf{F}_n(\mathsf{G}_n(\mathsf{td}_1), \cdot) : \{0,1\}^n \to \{0,1\}^{2n}$ uniformly from the set of all injective functions mapping n-bit inputs to $2n$-bit outputs, we have that

$$\Pr\left[\mathsf{HitInv}_{M_1,n}^{\mathcal{O}_{\mathsf{TDF}}}\right] \leq \frac{\binom{4^n-1}{2^n-1}}{\binom{4^n}{2^n}} = \frac{2^n}{4^n}.$$

For bounding the probability of the event $\mathsf{HitInv}_{M_{i+1},n}^{\mathcal{O}_{\mathsf{TDF}}}$ given that $\overline{\mathsf{HitInv}_{M_i,n}^{\mathcal{O}_{\mathsf{TDF}}}}$ occurred, we fix the queries $q_1, \ldots, q_i$ and their answers. Each query q_j is to F_n and of the form (pk, x) or to F_n^{-1} and of the form (td, y). Suppose the query q_{i+1} is to F_n^{-1} and of the form $(\mathsf{td}_{i+1}, y_{i+1})$. Let $\mathsf{pk}_{i+1} = \mathsf{G}_n(\mathsf{td}_{i+1})$, let $x_1, \ldots, x_a$ be the second arguments of all previous queries to F_n of the form (pk_{i+1}, x), let $y_1, \ldots, y_a$ be the answers to those queries, and let $y_{a+1}, \ldots, y_b$ be the second arguments of all previous queries to F_n^{-1} of the form (td_{i+1}, y) such that $\mathsf{F}_n^{-1}(\mathsf{td}_{i+1}, y_j) = \perp$ (so $b \leq i$). By the assumption that $\overline{\mathsf{HitInv}_{M_i,n}^{\mathcal{O}_{\mathsf{TDF}}}}$ we know that for every other query to F_n^{-1} of the form (td_{i+1}, y) holds $\mathsf{F}_n^{-1}(\mathsf{td}_{i+1}, y) \in \{x_1, \ldots, x_a\}$, thus we are now sampling a function $\{0,1\}^n \setminus \{x_1, \ldots, x_a\} \to \{0,1\}^{2n} \setminus \{y_1, \ldots, y_b\}$ uniformly from the set of all injective functions on those domain and range. So we have that

$$\Pr\left[\mathsf{HitInv}_{M_i,n}^{\mathcal{O}_{\mathsf{TDF}}}\right] \leq \frac{2^n - a}{4^n - b} \leq \frac{2^n}{4^n - i}.$$

We conclude that

$$\begin{aligned}\Pr\left[\mathsf{HitInv}_{M,n}^{\mathcal{O}_{\mathsf{TDF}}}\right] &\leq \Pr\left[\mathsf{HitInv}_{M_1,n}^{\mathcal{O}_{\mathsf{TDF}}}\right] + \sum_{i=1}^{q-1}\Pr\left[\mathsf{HitInv}_{M_{i+1},n}^{\mathcal{O}_{\mathsf{TDF}}}\middle|\overline{\mathsf{HitInv}_{M_i,n}^{\mathcal{O}_{\mathsf{TDF}}}}\right]\\ &\leq \sum_{i=0}^{q-1}\frac{2^n}{4^n-i}\\ &\leq \frac{q\cdot 2^n}{4^n-q}\\ &\leq \frac{q}{2^n-q}.\end{aligned}$$

■

Given an oracle $\mathcal{O}_{\mathsf{TDF}}$, sampled as described above, we let $\widehat{\mathcal{O}_{\mathsf{TDF}n}} = \{(\mathsf{G}_i, \mathsf{F}_i, \mathsf{F}_i^{-1})\}_{i\in\mathbb{N}\setminus\{n\}} \cup \{(\mathsf{F}_n, \mathsf{G}_n)\}$ denote the oracle that is obtained by omitting F_n^{-1}. For proving Claim 6.3 we rely on the following two claims, stating that the functions G_n and $\mathsf{F}_n(\mathsf{pk},\cdot)$ are hard to invert relative to $\widehat{\mathcal{O}_{\mathsf{TDF}n}}$.

Claim 6.6. *For every $q(n)$-query algorithm M it holds that*

$$\Pr\left[M^{\widehat{\mathcal{O}_{\mathsf{TDF}n}}}(\mathsf{G}_n(\mathsf{td})) \in \mathsf{G}_n^{-1}(\mathsf{G}_n(\mathsf{td}))\right] \leq \frac{2(q(n)+1)}{2^n - q(n)}$$

for all sufficiently large $n \in \mathbb{N}$, where the probability is taken over the choice of $\mathsf{td} \leftarrow \{0,1\}^n$ and the choice of the oracle $\widehat{\mathcal{O}_{\mathsf{TDF}n}}$ as described above. Moreover, $q(n)$ can be a bound on the number of queries to G_n only.

Claim 6.7. *For every $q(n)$-query algorithm M and every $\mathsf{pk} \in \{0,1\}^{2n}$ it holds that*

$$\Pr\left[M^{\widehat{\mathcal{O}_{\mathsf{TDF}n}}}(\mathsf{pk}, \mathsf{F}_n(\mathsf{pk}, x)) = x\right] \leq \frac{q(n)+1}{2^n - q(n)}$$

for all sufficiently large $n \in \mathbb{N}$, where the probability is taken over the choice of the oracle $\widehat{\mathcal{O}_{\mathsf{TDF}n}}$ as described above. Moreover, $q(n)$ can be a bound on the number of queries to $\mathsf{F}_n(\mathsf{pk},\cdot)$ only.

The proofs of Claims 6.6 and 6.7 and are nearly identical to the proof of Claim 5.3 (where the factor 2 is not present in the bound of Claim 6.7 because x is the only preimage of $\mathsf{F}_n(\mathsf{pk},\cdot)$), and are therefore omitted. We now deduce the proof of Claim 6.3:

Proof of Claim 6.3. Suppose M is a $q(n)$-query algorithm, and consider the following algorithm N with oracle access to $\widehat{\mathcal{O}_{\mathsf{TDF}n}}$ and input $\mathsf{pk} = \mathsf{G}_n(\mathsf{td})$ where $\mathsf{td} \leftarrow \{0,1\}^n$:

1. The algorithm N samples $x \leftarrow \{0,1\}^n$.
2. The algorithm N obtains $y = \mathsf{F}_n(\mathsf{pk}, x)$.
3. The algorithm N runs $\widetilde{x} \leftarrow M^{\mathcal{O}_{\mathsf{TDF}}}(\mathsf{pk}, y)$, where queries are answered according to $\widehat{\mathcal{O}_{\mathsf{TDF}n}}$, except for queries to F_n^{-1} of the form (td', y') which are answered in the following manner:
   (a) If $\mathsf{G}_n(\mathsf{td}') = \mathsf{pk}$ then the algorithm N outputs td' and terminates.
   (b) If a previous query to F_n of the form $(\mathsf{G}_n(\mathsf{td}'), x')$ resulted with y', then algorithm N answers the query with x'.

(c) Otherwise, then algorithm N answers the query with answer $\perp$.

4. If $\widetilde{x} = x$ then output 1, and otherwise output 0.

If N terminates on step 3.(a) then it manages to invert G_n, therefore by Claim 6.6 it holds that

$$\Pr\left[N \text{ teminates on step 3.(a)}\right] \leq \frac{2(q(n)+1)}{2^n - q(n)}.$$

We may see step 3 of the algorithm N as an algorithm by itself with input $(\mathsf{pk}, F_n(\mathsf{pk}, x))$, oracle access to $\widehat{\mathcal{O}_{\mathsf{TDF}_n}}$ and output $\tilde{x}$, so by Claim 6.7 it holds that

$$\Pr\left[N \text{ outputs } 1\right] = \Pr\left[\text{Step 3 of } N \text{ outputs } x\right] \leq \frac{q(n)+1}{2^n - q(n)}.$$

Finally, if N gives M a wrong oracle answer for a query to F_n^{-1} (i.e. not consistent with $(\mathsf{G}_n, \mathsf{F}_n)$) then $\mathsf{HitInv}^{\mathcal{O}_{\mathsf{TDF}}}_{M(\mathsf{pk},y),n}$ occurs. The computation of $M(\mathsf{pk}, y)$, including the computation of pk and y, consists of at most $q(n)+1$ queries to F_n and F_n^{-1}, therefore by Claim 6.5 it holds that

$$\Pr[N \text{ gives } M \text{ a wrong oracle answer}] \leq \frac{q(n)+1}{2^n - q(n)}.$$

Now, if $M^{\mathcal{O}_{\mathsf{TDF}}}(\mathsf{pk}, \mathsf{F}_n(\mathsf{pk}, x)) = x$ then either the algorithm N outputs 1 or the simulation done by N goes wrong (i.e., N terminates or gives a wrong oracle answer to M). Therefore, it holds that

$$\begin{aligned}
&\Pr\left[M^{\mathcal{O}_{\mathsf{TDF}}}\left(\mathsf{G}_n(\mathsf{td}), \mathsf{F}_n(\mathsf{G}_n(\mathsf{pk}), x)\right) = x\right] \\
&\quad \leq \Pr[N \text{ outputs } 1] \\
&\qquad + \Pr[N \text{ gives } M \text{ a wrong answer}] \\
&\qquad + \Pr[N \text{ teminates on step 3.(a)}] \\
&\quad \leq \frac{4(q(n)+1)}{2^n - q(n)}.
\end{aligned}$$

■

6.3 Solving Oracle-Aided Bounded-TFNP Instances Relative to $\mathcal{O}_{\mathsf{TDF}}$

As discussed above, our generalization of the attack presented in Section 5 relies on the fact that it should be infeasible to "guess" elements in the images of the functions G_n and $\mathsf{F}_n(\mathsf{pk}, \cdot)$. Let M be an oracle-aided algorithm, and during the runtime of M we allow it to make "guesses" of the form $\mathsf{pk} \in \{0,1\}^{2i}$ or of the form (pk, y) where $\mathsf{pk} \in \{0,1\}^{2i}$ and $y \in \{0,1\}^{2i}$ for some i. When counting the number of oracle calls we also include the number of guesses. We denote by $\mathsf{HitFRange}^{\mathcal{O}_{\mathsf{TDF}}}_{M,n}$ the event in which M guesses (pk, y) for which there exists $x \in \{0,1\}^n$ with $\mathsf{F}_n(\mathsf{pk}, x) = y$ without querying F_n with (pk, x) before. Similarly, we denote by $\mathsf{HitGRange}^{\mathcal{O}_{\mathsf{TDF}}}_{M,n}$ the event in which M guesses $\mathsf{pk} \in \{0,1\}^{2n}$ for which there exists $\mathsf{td} \in \{0,1\}^n$ with $\mathsf{pk} = \mathsf{G}_n(\mathsf{td})$ without querying G_n on td before. In Section 6.4 we prove the following claims:

Claim 6.8. *Denote by $\widehat{\mathcal{O}_{\mathsf{TDF}_n}}$ the oracle obtained by omitting F_n^{-1} from the oracle $\mathcal{O}_{\mathsf{TDF}}$. For every q-query algorithm M it holds that*

$$\Pr\left[\mathsf{HitFRange}^{\widehat{\mathcal{O}_{\mathsf{TDF}_n}}}_{M,n}\right] \leq \frac{q}{2^n - q}$$

where the probability is taken over the choice of the oracle $\mathcal{O}_{\mathsf{TDF}} = \{(\mathsf{G}_i, \mathsf{F}_i, \mathsf{F}_i^{-1})\}_{i\in\mathbb{N}}$ as described above. Moreover, q can be a bound on the number of guesses and calls to F_n.

Claim 6.9. *For every q-query algorithm M it holds that*

$$\Pr\left[\mathsf{HitInv}_{M,n}^{\mathcal{O}_{\mathsf{TDF}}} \vee \mathsf{HitFRange}_{M,n}^{\mathcal{O}_{\mathsf{TDF}}}\right] \leq \frac{2q}{2^n - q}$$

where the probability is taken over the choice of the oracle $\mathcal{O}_{\mathsf{TDF}} = \{(\mathsf{G}_i, \mathsf{F}_i, \mathsf{F}_i^{-1})\}_{i\in\mathbb{N}}$ as described above. Moreover, q can be a bound on the number of guesses and calls to F_n and F_n^{-1}.

Claim 6.10. *For every q-query algorithm M it holds that*

$$\Pr\left[\mathsf{HitGRange}_{M,n}^{\mathcal{O}_{\mathsf{TDF}}}\right] \leq \frac{q}{2^n - q}$$

where the probability is taken over the choice of the oracle $\mathcal{O}_{\mathsf{TDF}} = \{(\mathsf{G}_i, \mathsf{F}_i, \mathsf{F}_i^{-1})\}_{i\in\mathbb{N}}$ as described above. Moreover, q can be a bound on the number of guesses and calls to G_n.

Equipped with Claims 6.8–6.10 we now prove Claim 6.4.

Proof of Claim 6.4. Let $p(n) = 1/2$ (although the proof goes through for any value of $p(n)$). To simplify the notation, we denote $\mathsf{G}_n(\mathsf{td})$, $\mathsf{F}_n(\mathsf{pk}, x)$ and $\mathsf{F}_n^{-1}(\mathsf{td}, y)$ by $\mathsf{G}(\mathsf{td})$, $\mathsf{F}(\mathsf{pk}, x)$ and $\mathsf{F}^{-1}(\mathsf{td}, y)$. There is no ambiguity since n can be determined by the size of the input. Let $C = \{(\mathsf{Gen}_n, C_n)\}_{n\in\mathbb{N}}$ be an oracle-aided k-bounded TFNP instance that satisfies the correctness requirement stated in Definition 6.1, where Gen_n and C_n contain at most $q(n)$ oracle gates each, and the input to each such gate is of length at most $q(n)$ bits. We modify the circuit such that each query to F^{-1} with input (td, y) is preceded by a query to G with input td. This may double $q(n)$, but to ease the notation we will assume that $q(n)$ is a bound on the number of gates in the modified circuits. Consider the following oracle-aided algorithm $\mathcal{A}$ that on input 1^n and σ tries to find an input $x \in \{0,1\}^n$ such that $C_n^{\mathcal{O}_{\mathsf{TDF}}}(\sigma, x) = 1$:

1. The algorithm $\mathcal{A}$ sets $a(n) = \log\left(12\left(q(n)+1\right)^3 \cdot \left(k(n)+1\right)/p(n) + 4\left(q(n)+1\right)^2 \cdot \left(k(n)+1\right)\right)$.
2. The algorithm $\mathcal{A}$ initialize empty lists Q_{G} and Q_{F}.
   The set Q_{G} will contain pairs of the form $(\mathsf{td}, \mathsf{pk})$ where $\mathsf{G}_i(\mathsf{td}) = \mathsf{pk}$ and the set Q_{F} will contain triplets of the form (pk, x, y) where $\mathsf{F}_i(\mathsf{pk}, x) = y$ and triplets of the form $(\mathsf{pk}, \perp, y)$ where $y \notin \mathsf{F}_i(\mathsf{pk}, \{0,1\}^i)$.
3. The algorithm $\mathcal{A}$ initialize an empty list Check.
   The list Check will contain inputs $x \in \{0,1\}^n$ to the oracle-aided circuit C_n.
4. For every $1 \leq i < a(n)$, the algorithm $\mathcal{A}$ queries G_i and F_i on all possible inputs, and adds these queries to the sets Q_{G} and Q_{F} respectively.
5. The algorithm $\mathcal{A}$ performs the following steps for $q(n)+1$ iterations:
   (a) The algorithm $\mathcal{A}$ finds an oracle $\widetilde{\mathcal{O}_{\mathsf{TDF}}} = \{(\widetilde{\mathsf{G}}_n, \widetilde{\mathsf{F}}_n, \widetilde{\mathsf{F}}_n^{-1})\}_{n\in\mathbb{N}}$ that is valid, consistent with Q_{G} and Q_{F}, has $\widetilde{r}$ such that $\sigma = \mathsf{Gen}_n^{\widetilde{\mathcal{O}_{\mathsf{TDF}}}}(\widetilde{r})$, and maximizes the number of solutions $\widetilde{k}$ to the instance $C_n^{\widetilde{\mathcal{O}_{\mathsf{TDF}}}}(\sigma, \cdot)$ under those constraints (i.e., $\widetilde{k}$ is the number of distinct inputs $x \in \{0,1\}^n$ such that $C_n^{\widetilde{\mathcal{O}_{\mathsf{TDF}}}}(\sigma, x) = 1$).
   (b) The algorithm $\mathcal{A}$ finds the distinct inputs $x_1, \ldots, x_{\widetilde{k}} \in \{0,1\}^n$ for which $C_n^{\widetilde{\mathcal{O}_{\mathsf{TDF}}}}(\sigma, x_i) = 1$ for every $i \in [\widetilde{k}]$, and $\widetilde{r}$ for which $\sigma = \mathsf{Gen}_n(\widetilde{r})$.
   (c) The algorithm $\mathcal{A}$ adds $x_1, \ldots, x_{\widetilde{k}}$ to the list Check.
   (d) For every query td to a $\widetilde{\mathsf{G}}$-gate in the computations $\mathsf{Gen}_n^{\widetilde{\mathcal{O}_{\mathsf{TDF}}}}(\widetilde{r}), C_n^{\widetilde{\mathcal{O}_{\mathsf{TDF}}}}(\sigma, x_1), \ldots, C_n^{\widetilde{\mathcal{O}_{\mathsf{TDF}}}}(\sigma, x_{\widetilde{k}})$, the algorithm $\mathcal{A}$ guesses the value $\widetilde{\mathsf{G}}(\mathsf{td})$.[8]

[8] For an explanation regarding the guessing mechanism we refer the reader to the beginning of this section.

(e) For every query (pk, x) to a $\widetilde{\mathsf{F}}$-gate in the computations $\mathsf{Gen}_n^{\widetilde{\mathcal{O}_{\mathsf{TDF}}}}(\widetilde{r}), C_n^{\widetilde{\mathcal{O}_{\mathsf{TDF}}}}(\sigma, x_1), \ldots, C_n^{\widetilde{\mathcal{O}_{\mathsf{TDF}}}}(\sigma, x_{\widetilde{k}})$, the algorithm $\mathcal{A}$ guesses the pair $(\mathsf{pk}, \widetilde{\mathsf{F}}(\mathsf{pk}, x))$.

(f) For every query (td, y) to a $\widetilde{\mathsf{F}}^{-1}$-gate in the computations $\mathsf{Gen}_n^{\widetilde{\mathcal{O}_{\mathsf{TDF}}}}(\widetilde{r}), C_n^{\widetilde{\mathcal{O}_{\mathsf{TDF}}}}(\sigma, x_1), \ldots, C_n^{\widetilde{\mathcal{O}_{\mathsf{TDF}}}}(\sigma, x_{\widetilde{k}})$, the algorithm $\mathcal{A}$ guesses the pair $(\widetilde{\mathsf{G}}(\mathsf{td}), y)$.

(g) The algorithm $\mathcal{A}$ queries G with all inputs to $\widetilde{\mathsf{G}}$-gates in the computations $\mathsf{Gen}_n^{\widetilde{\mathcal{O}_{\mathsf{TDF}}}}(\widetilde{r})$, $C_n^{\widetilde{\mathcal{O}_{\mathsf{TDF}}}}(\sigma, x_1), \ldots, C_n^{\widetilde{\mathcal{O}_{\mathsf{TDF}}}}(\sigma, x_{\widetilde{k}})$, and adds these queries to the set Q_{G}.

(h) For every query (pk, x) to a $\widetilde{\mathsf{F}}$-gate in the computations $\mathsf{Gen}_n^{\widetilde{\mathcal{O}_{\mathsf{TDF}}}}(\widetilde{r}), C_n^{\widetilde{\mathcal{O}_{\mathsf{TDF}}}}(\sigma, x_1), \ldots, C_n^{\widetilde{\mathcal{O}_{\mathsf{TDF}}}}(\sigma, x_{\widetilde{k}})$ resulting with $y = \widetilde{\mathsf{F}}(\mathsf{pk}, x)$:
   i. The algorithm $\mathcal{A}$ queries F with (pk, x) and adds $(\mathsf{pk}, x, \mathsf{F}(\mathsf{pk}, x))$ to Q_{F}.
   ii. If $(\mathsf{td}, \mathsf{pk}) \in Q_{\mathsf{G}}$ for some td then the algorithm $\mathcal{A}$ queries F^{-1} with (td, y) and adds $(\mathsf{pk}, \mathsf{F}^{-1}(\mathsf{td}, y), y)$ to Q_{F}.

(i) For every query (td, y) to a $\widetilde{\mathsf{F}}^{-1}$-gate in the computations $\mathsf{Gen}_n^{\widetilde{\mathcal{O}_{\mathsf{TDF}}}}(\widetilde{r}), C_n^{\widetilde{\mathcal{O}_{\mathsf{TDF}}}}(\sigma, x_1), \ldots, C_n^{\widetilde{\mathcal{O}_{\mathsf{TDF}}}}(\sigma, x_{\widetilde{k}})$ resulting with $x = \widetilde{\mathsf{F}}^{-1}(\mathsf{td}, y)$ which might be $\perp$:
   i. The algorithm $\mathcal{A}$ queries F^{-1} with (td, y) and adds $(\mathsf{G}(\mathsf{td}), \mathsf{F}^{-1}(\mathsf{td}, x), y)$ to Q_{F}.
   ii. If $x \neq \perp$ then the algorithm $\mathcal{A}$ queries F with $(\mathsf{G}(\mathsf{td}), x)$ and adds $(\mathsf{G}(\mathsf{td}), x, \mathsf{F}(\mathsf{G}(\mathsf{td}), x))$ to Q_{F}.

6. For every $x \in \mathsf{Check}$, the algorithm $\mathcal{A}$ computes $C_n^{\mathcal{O}_{\mathsf{TDF}}}(\sigma, x)$, and if $C_n^{\mathcal{O}_{\mathsf{TDF}}}(\sigma, x) = 1$ then it outputs x and terminates.
7. If no such x was found then the algorithm $\mathcal{A}$ outputs $\perp$.

In terms of the number of oracle queries made by $\mathcal{A}$, observe that in step 4 the algorithm require at most $2 \cdot 2^{a(n)} + 3 \cdot 2^{3a(n)} = O(q(n)^9 \cdot k(n)^3 / p(n)^3)$ queries, in each iteration the algorithm $\mathcal{A}$ performs at most $2 \cdot q \cdot (k+1)$ oracle queries, and in step 6 the algorithm performs at most $(q+1) \cdot q \cdot (k+1)$ oracle queries. Therefore the algorithm $\mathcal{A}$ in total performs at most $O(q(n)^9 \cdot k(n)^3 / p(n)^3)$ queries. For later analysis we note that including the oracle queries in the computation $\sigma \leftarrow \mathsf{Gen}_n^{\mathcal{O}_{\mathsf{TDF}}}()$, including the guesses of $\mathcal{A}$, and excluding the queries in step 4 which are only to the oracles $\{(\mathsf{G}_i, \mathsf{F}_i, \mathsf{F}_i^{-1})\}_{1 \leq i < a(n)}$, the algorithm performs at most $4 \cdot (q+1)^2 \cdot (k+1)$ queries. Moreover, given oracle access a PSPACE-complete oracle, the algorithm $\mathcal{A}$ can be implemented to run in time $q(n)^9 \cdot k(n)^3 \cdot \mathsf{poly}(n)$ (in a manner that is similar to that described in the proof of Claim 5.4).

Fix $\mathcal{O}_{\mathsf{TDF}} = (\mathsf{G}, \mathsf{F}, \mathsf{G}^{-1})$, fix $\sigma \leftarrow \mathsf{Gen}_n^{\mathcal{O}_{\mathsf{TDF}}}$ and fix some $x^* \in \{0,1\}^n$ such that $C_n^{\mathcal{O}_{\mathsf{TDF}}}(\sigma, x^*) = 1$. Fix an iteration $j \in [q(n)+1]$ of the algorithm $\mathcal{A}$. Let Q_{F} and Q_{G} denote these variables in the beginning of that iteration, let $\widetilde{\mathcal{O}_{\mathsf{TDF}}} = (\widetilde{\mathsf{G}}, \widetilde{\mathsf{F}}, \widetilde{\mathsf{F}}^{-1})$ be the oracle chosen in that iteration, let $x_1, \ldots, x_{\widetilde{k}}$ be the solutions to $C_n^{\widetilde{\mathcal{O}_{\mathsf{TDF}}}}(\sigma, \cdot)$ and assume that none of them is x^*, and let $\widetilde{r}$ such that $\sigma = \mathsf{Gen}_n^{\widetilde{\mathcal{O}_{\mathsf{TDF}}}}(\widetilde{r})$. Let $Q_{\widetilde{\mathsf{G}}}$ be the $\widetilde{\mathsf{G}}$-gate queries done in the computations $\mathsf{Gen}_n^{\widetilde{\mathcal{O}_{\mathsf{TDF}}}}(\widetilde{r}), C_n^{\widetilde{\mathcal{O}_{\mathsf{TDF}}}}(\sigma, x_1), \ldots, C_n^{\widetilde{\mathcal{O}_{\mathsf{TDF}}}}(\sigma, x_{\widetilde{k}})$, and let $Q_{\widetilde{\mathsf{F}}}$ be the $\widetilde{\mathsf{F}}$-gate and $\widetilde{\mathsf{F}}^{-1}$-gate queries done in these computations, i.e. for each query to $\widetilde{\mathsf{F}}$ of the form (pk, x) we store $(\mathsf{pk}, x, \widetilde{\mathsf{F}}(x, \mathsf{pk}))$ and for each query to $\widetilde{\mathsf{F}}^{-1}$ of the form (td, y) we store $(\widetilde{\mathsf{G}}(\mathsf{td}), \widetilde{F}^{-1}(\mathsf{td}, y), y)$ (where the middle value might be $\perp$). Note that in the case of query to $\widetilde{\mathsf{F}}^{-1}$ it holds that $(\mathsf{td}, \widetilde{\mathsf{G}}(\mathsf{td})) \in Q_{\widetilde{\mathsf{G}}}$ due to our assumption that each query to $\widetilde{\mathsf{F}}^{-1}$ is preceded by a matching query to $\widetilde{\mathsf{G}}$. Let Q_{F}^* and Q_{G}^* be the queries done in the computation $C_n^{\mathcal{O}_{\mathsf{TDF}}}(\sigma, x^*)$.

If there exists a valid oracle $\mathcal{O}'_{\mathsf{TDF}}$ which is consistent with Q_{F}, Q_{G}, $Q_{\widetilde{\mathsf{F}}}$, $Q_{\widetilde{\mathsf{G}}}$, Q_{F}^* and Q_{G}^* then we get that $\mathcal{O}'_{\mathsf{TDF}}$ has at least $\widetilde{k}+1$ solutions - $x_1, \ldots, x_{\widetilde{k}}$ and x^*. Along with the fact that $\mathcal{O}'_{\mathsf{TDF}}$ is consistent with Q_{F} and Q_{G} and that $\sigma = \mathsf{Gen}_n^{\mathcal{O}'_{\mathsf{TDF}}}(\widetilde{r})$, we get a contradiction to the maximality of $\widetilde{\mathcal{O}_{\mathsf{TDF}}}$. Therefore, at least one of the following cases holds:

Case 1 There exists td and $\mathsf{pk} \neq \mathsf{pk}'$ for which $(\mathsf{td}, \mathsf{pk}) \in Q_{\widetilde{\mathsf{G}}}$ but $(\mathsf{td}, \mathsf{pk}') \in Q^*_{\mathsf{G}}$. This means that the pair $(\mathsf{td}, \mathsf{pk}')$ is currently not contained in Q_{G} but the algorithm $\mathcal{A}$ will add it in step 5.(g).

Case 2 There exists pk, $x \neq \perp$ and $y \neq y'$ for which $(\mathsf{pk}, x, y) \in Q_{\widetilde{\mathsf{F}}}$ but $(\mathsf{pk}, x, y') \in Q^*_{\mathsf{F}}$. This means that the triplet (pk, x, y') is currently not contained in Q_{F} but the algorithm $\mathcal{A}$ will add it in step 5.(h).i or 5.(i).ii.

Case 3 There exists pk, $x \neq x'$ and y for which $(\mathsf{pk}, x, y) \in Q_{\widetilde{\mathsf{F}}}$ but $(\mathsf{pk}, x', y) \in Q^*_{\mathsf{F}}$. This case splits into two cases:

Case 3.a If $x' \neq \perp$ then that means that $\mathcal{A}$ managed to guess (pk, y) in step 5.(e) or 5.(f) without querying F on (pk, x') before.

Case 3.b If $x' = \perp$ then $(\mathsf{pk}, \perp, y)$ is in Q^*_{F} due to a query to F^{-1} of the form (td, y) where $\mathsf{G}(\mathsf{td}) = \mathsf{pk}$. If $(\mathsf{td}, \mathsf{pk}) \in Q_{\mathsf{G}}$ then the algorithm $\mathcal{A}$ will add $(\mathsf{pk}, \perp, y)$ to Q_{F} in step 5.(h).ii or 5.(i).i. If $(\mathsf{td}, \mathsf{pk}) \notin Q_{\mathsf{G}}$ that means that $\mathcal{A}$ managed to guess pk in step 5.(d) without querying G on td before.

Case 4 There exists $\mathsf{pk} \in \{0,1\}^{2i}$ for which there are more than $2^{2i} - 2^i$ pairs of the form $(\mathsf{pk}, \perp, y)$ in $Q_{\mathsf{F}} \cup Q_{\widetilde{\mathsf{F}}} \cup Q^*_{\mathsf{F}}$. Let $Y = \{y | \nexists x \mathsf{F}(\mathsf{pk}, x) = y\}$ and $\widetilde{Y} = \{y | (\mathsf{pk}, \perp, y) \in Q_{\widetilde{\mathsf{F}}}\}$. Then $|Y \cup \widetilde{Y}| > 2^{2i} - 2^i$ but $|Y| = 2^{2i} - 2^i$, hence there exists $y \in \widetilde{Y}$ with $y \notin Y$, thus the algorithm $\mathcal{A}$ manages to guess (pk, y) in step 5.(e) or 5.(f) for which there exists x with $\mathsf{F}(\mathsf{pk}, x) = y$ without querying F on (pk, x) before.

So we get that in every iteration, at least one of the followings happens:

- The algorithm $\mathcal{A}$ finds a solution to $C_n^{\mathcal{O}_{\mathsf{TDF}}}(\sigma, \cdot)$ (which will be checked in step 6).
- The event $\mathsf{HitGRange}^{\mathcal{O}_{\mathsf{TDF}}}_{\mathcal{A}(1^n, \mathsf{Gen}_n()), i}$ occurs to some $a(n) \leq i \leq q(n)$.
- The event $\mathsf{HitFRange}^{\mathcal{O}_{\mathsf{TDF}}}_{\mathcal{A}(1^n, \mathsf{Gen}_n()), i}$ occurs to some $a(n) \leq i \leq q(n)$.
- The algorithm $\mathcal{A}$ adds a new pair from Q^*_{G} to Q_{G}.
- The algorithm $\mathcal{A}$ adds a new triplet from Q^*_{F} to Q_{F}.

Denoting $\mathsf{HIT}_n^{\mathcal{O}_{\mathsf{TDF}}} = \bigvee_{i=\lceil a(n) \rceil}^{q(n)} (\mathsf{HitGRange}^{\mathcal{O}_{\mathsf{TDF}}}_{\mathcal{A}(1^n, \mathsf{Gen}_n()), i} \vee \mathsf{HitFRange}^{\mathcal{O}_{\mathsf{TDF}}}_{\mathcal{A}(1^n, \mathsf{Gen}_n()), i})$, and fixing $\mathcal{O}_{\mathsf{TDF}}$ and randomness for Gen_n for which $\mathsf{HIT}_n^{\mathcal{O}_{\mathsf{TDF}}}$ does not occur, we get that after $q(n)$ iteration, if $\mathcal{A}$ did not find a solution to $C_n^{\mathcal{O}_{\mathsf{TDF}}}(\sigma, \cdot)$ yet, then $Q_{\mathsf{G}} \supset Q^*_{\mathsf{G}}$ and $Q_{\mathsf{F}} \supset Q^*_{\mathsf{F}}$. Therefore, in the $q(n)+1$ iteration holds $\widetilde{C_n^{\mathcal{O}_{\mathsf{TDF}}}}(\sigma, x^*) = 1$ and the algorithm $\mathcal{A}$ finds the solution x^* to $C_n^{\mathcal{O}_{\mathsf{TDF}}}(\sigma, \cdot)$. Therefore,

$$
\begin{aligned}
&\Pr\left[\mathcal{A}^{\mathcal{O}_{\mathsf{TDF}}}(1^n, \sigma) = x \text{ s.t. } C_n^{\mathcal{O}_{\mathsf{TDF}}}(\sigma, x) = 0\right] \\
&\quad \leq \sum_{i=\lceil a(n) \rceil}^{q(n)} \Pr\left[\mathsf{HitGRange}^{\mathcal{O}_{\mathsf{TDF}}}_{\mathcal{A}(1^n, \mathsf{Gen}_n()), i} \vee \mathsf{HitFRange}^{\mathcal{O}_{\mathsf{TDF}}}_{\mathcal{A}(1^n, \mathsf{Gen}_n()), i}\right] \\
&\quad \leq \sum_{i=\lceil a(n) \rceil}^{q(n)} \frac{3 \cdot 4 \cdot (q(n)+1)^2 \cdot (k(n)+1)}{2^i - 4 \cdot (q(n)+1)^2 \cdot (k(n)+1)} \\
&\quad \leq \frac{12(q(n)+1)^3 \cdot (k(n)+1)}{2^{a(n)} - 4 \cdot (q(n)+1)^2 \cdot (k(n)+1)} \\
&\quad \leq p(n).
\end{aligned}
$$

where the probability is taken over the choice of the oracle $\mathcal{O}_{\mathsf{TDF}} = \{(\mathsf{G}_n, \mathsf{F}_n, \mathsf{F}_n^{-1})\}_{n \in \mathbb{N}}$ and over the choice of $\sigma \leftarrow \mathsf{Gen}_n^{\mathcal{O}_{\mathsf{TDF}}}()$. This settles the proof of Claim 6.4. ■

6.4 Proofs of Claims 6.8–6.10

Proof of Claim 6.8. Let M be a q-query algorithm. Fix $n \in \mathbb{N}$, $(\mathcal{O}_{\mathsf{TDF}})_{-n} = \{(\mathsf{G}_i, \mathsf{F}_i, \mathsf{F}_i^{-1})\}_{i \in \mathbb{N} \setminus \{n\}}$ (i.e., we fix the entire oracle $\mathcal{O}_{\mathsf{TDF}}$ except for the nth instance), and $\mathsf{G}_n : \{0,1\}^n \to \{0,1\}^n$. Thus, we only consider queries to the oracles F_n and guesses. We may assume without loss of generality that if M queried F_n with some (pk, x) and got y as a result, then it will not make the guess (pk, y). For every $i \in [q]$ denote by M_i the following i-query algorithm: Invoke the computation $M^{\widehat{\mathcal{O}_{\mathsf{TDF}_n}}}$, and terminate once i oracle queries have been performed. Note that since we do not place any restriction on the running time of M and since the oracle distribution is known, we can assume without loss of generality that M is deterministic. Therefore, for every $i \in [q]$ and every fixing of the oracle $\mathcal{O}_{\mathsf{TDF}}$, the computation $M_i^{\widehat{\mathcal{O}_{\mathsf{TDF}_n}}}$ is the "prefix" of the computation $M^{\widehat{\mathcal{O}_{\mathsf{TDF}_n}}}$ which contains its first i oracle queries. This implies that

$$\Pr\left[\mathsf{HitFRange}_{M,n}^{\widehat{\mathcal{O}_{\mathsf{TDF}_n}}}\right] \leq \Pr\left[\mathsf{HitFRange}_{M_1,n}^{\widehat{\mathcal{O}_{\mathsf{TDF}_n}}}\right] + \sum_{i=1}^{q-1} \Pr\left[\mathsf{HitFRange}_{M_{i+1},n}^{\widehat{\mathcal{O}_{\mathsf{TDF}_n}}} \middle| \overline{\mathsf{HitFRange}_{M_i,n}^{\widehat{\mathcal{O}_{\mathsf{TDF}_n}}}}\right],$$

where the probability is taken over the choice of F_n.

For bounding the probability of the event $\mathsf{HitFRange}_{M_1,n}^{\widehat{\mathcal{O}_{\mathsf{TDF}_n}}}$, note that this event corresponds to the fact that M, without any information on $\mathsf{F}_n : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^{2n}$ (since no oracle queries have been issued so far), manages to guess (pk_1, y_1) for which there exists x such that $\mathsf{F}_n(\mathsf{pk}_1, x) = y_1$.

Since the guess (pk_1, y_1) is fixed by the description of M, and we are now sampling $\mathsf{F}_n(\mathsf{pk}_1, \cdot) : \{0,1\}^n \to \{0,1\}^{2n}$ uniformly from the set of all injective functions mapping n-bit inputs to $2n$-bit outputs, we have that

$$\Pr\left[\mathsf{HitFRange}_{M_1,n}^{\widehat{\mathcal{O}_{\mathsf{TDF}_n}}}\right] \leq \frac{\binom{4^n-1}{2^n-1}}{\binom{4^n}{2^n}} = \frac{2^n}{4^n}.$$

For bounding the probability of the event $\mathsf{HitFRange}_{M_{i+1},n}^{\widehat{\mathcal{O}_{\mathsf{TDF}_n}}}$ given that $\overline{\mathsf{HitFRange}_{M_i,n}^{\widehat{\mathcal{O}_{\mathsf{TDF}_n}}}}$ occurred, we fix the queries or guesses $q_1, \ldots, q_i$, and fix the answers to the queries. Each q_j is a query to F_n and of the form (pk, x) or a guess of the form (pk, y). Suppose the query q_{i+1} is a guess of the form $(\mathsf{pk}_{i+1}, y_{i+1})$. Let $x_1, \ldots, x_a$ be the second argument of all previous queries to F_n of the form (pk_{i+1}, x), let $y_1, \ldots, y_a$ be the answers to these queries, and let $y_{a+1}, \ldots, y_{a+b}$ be the second argument of all previous guesses of the form (pk_{i+1}, y) (so $a + b \leq i$). By the assumption that $\overline{\mathsf{HitFRange}_{M_i,n}^{\widehat{\mathcal{O}_{\mathsf{TDF}_n}}}}$ we know that $y_{a+1}, \ldots, y_{a+b}$ are not in the image of $\mathsf{F}_n(\mathsf{pk}_{i+1}, \cdot)$, thus we are now sampling a function $\{0,1\}^n \setminus \{x_1, \ldots, x_a\} \to \{0,1\}^{2n} \setminus \{y_1, \ldots, y_{a+b}\}$ uniformly from the set of all injective functions on those domain and range. So we have that

$$\Pr\left[\mathsf{HitFRange}_{M_i,n}^{\widehat{\mathcal{O}_{\mathsf{TDF}_n}}}\right] \leq \frac{2^n - a}{4^n - a - b} \leq \frac{2^n}{4^n - i}.$$

We conclude that

$$\begin{aligned}\Pr\left[\mathsf{HitFRange}_{M,n}^{\widehat{\mathcal{O}_{\mathsf{TDF}_n}}}\right] &\leq \Pr\left[\mathsf{HitFRange}_{M_1,n}^{\widehat{\mathcal{O}_{\mathsf{TDF}_n}}}\right] + \sum_{i=1}^{q-1}\Pr\left[\mathsf{HitFRange}_{M_{i+1},n}^{\widehat{\mathcal{O}_{\mathsf{TDF}_n}}}\middle|\overline{\mathsf{HitFRange}_{M_i,n}^{\widehat{\mathcal{O}_{\mathsf{TDF}_n}}}}\right]\\ &\leq \sum_{i=0}^{q-1}\frac{2^n}{4^n-i}\\ &\leq \frac{q\cdot 2^n}{4^n-q}\\ &\leq \frac{q}{2^n-q}.\end{aligned}$$

■

Proof of Claim 6.9. Consider the following algorithm N with oracle access to $\widehat{\mathcal{O}_{\mathsf{TDF}_n}}$:

1. The algorithm N runs the algorithm M, where oracle queries are answered according to $\widehat{\mathcal{O}_{\mathsf{TDF}_n}}$, except for queries to F_n^{-1} of the form (td', y') which are answered in the following manner:
   (a) If a previous query to F_n of the form $(\mathsf{G}_n(\mathsf{td}'), x')$ resulted with y', then algorithm N answers the query with x'.
   (b) Otherwise, then algorithm N answers the query with answer $\perp$.
2. The algorithm N outputs the output of the execution of the algorithm M.

If $\mathsf{HitInv}_{M,n}^{\mathcal{O}_{\mathsf{TDF}}}$ does not occur then N answers all the oracle calls correctly, therefore $\mathsf{HitFRange}_{M,n}^{\mathcal{O}_{\mathsf{TDF}}}$ occurs if and only if $\mathsf{HitFRange}_{N,n}^{\widehat{\mathcal{O}_{\mathsf{TDF}_n}}}$ occurs. Hence by Claim 6.5 and Claim 6.8 it holds that

$$\begin{aligned}\Pr\left[\mathsf{HitInv}_{M,n}^{\mathcal{O}_{\mathsf{TDF}}} \vee \mathsf{HitFRange}_{M,n}^{\mathcal{O}_{\mathsf{TDF}}}\right] &= \Pr\left[\mathsf{HitInv}_{M,n}^{\mathcal{O}_{\mathsf{TDF}}}\right] + \Pr\left[\overline{\mathsf{HitInv}_{M,n}^{\mathcal{O}_{\mathsf{TDF}}}} \vee \mathsf{HitFRange}_{M,n}^{\mathcal{O}_{\mathsf{TDF}}}\right]\\ &= \Pr\left[\mathsf{HitInv}_{M,n}^{\mathcal{O}_{\mathsf{TDF}}}\right] + \Pr\left[\overline{\mathsf{HitInv}_{M,n}^{\mathcal{O}_{\mathsf{TDF}}}} \vee \mathsf{HitFRange}_{N,n}^{\widehat{\mathcal{O}_{\mathsf{TDF}_n}}}\right]\\ &\leq \Pr\left[\mathsf{HitInv}_{M,n}^{\mathcal{O}_{\mathsf{TDF}}}\right] + \Pr\left[\mathsf{HitFRange}_{N,n}^{\widehat{\mathcal{O}_{\mathsf{TDF}_n}}}\right]\\ &\leq \frac{2q}{2^n-q}\end{aligned}$$

where the probability is taken over the choice of the oracle $\mathcal{O}_{\mathsf{TDF}} = \{(\mathsf{G}_i, \mathsf{F}_i, \mathsf{F}_i^{-1})\}_{i\in\mathbb{N}}$. ■

Proof of Claim 6.10. Let M be a q-query algorithm, fix $(\mathcal{O}_{\mathsf{TDF}})_{-n} = \{(\mathsf{G}_i, \mathsf{F}_i, \mathsf{F}_i^{-1})\}_{i\in\mathbb{N}\setminus\{n\}}$ (i.e., we fix the entire oracle $\mathcal{O}_{\mathsf{TDF}}$ except for the nth instance), and fix $\mathsf{F}_n : \{0,1\}^{2n}\times\{0,1\}^n \to \{0,1\}^{2n}$. Thus, we only consider queries to the oracle G_n. For every $i \in [q]$ denote by M_i the following i-query algorithm: Invoke the computation $M^{\mathcal{O}_{\mathsf{TDF}}}$, and terminate once i oracle queries have been performed. Note that since we do not place any restriction on the running time of M and since the oracle distribution is known, we can assume without loss of generality that M is deterministic. Therefore, for every $i \in [q]$ and every fixing of the oracle $\mathcal{O}_{\mathsf{TDF}}$, the computation $M_i^{\mathcal{O}_{\mathsf{TDF}}}$ is the "prefix" of the computation $M^{\mathcal{O}_{\mathsf{TDF}}}$ which contains its first i oracle queries. This implies that

$$\Pr\left[\mathsf{HitGRange}_{M,n}^{\mathcal{O}_{\mathsf{TDF}}}\right] \leq \Pr\left[\mathsf{HitGRange}_{M_1,n}^{\mathcal{O}_{\mathsf{TDF}}}\right] + \sum_{i=1}^{q-1}\Pr\left[\mathsf{HitGRange}_{M_{i+1},n}^{\mathcal{O}_{\mathsf{TDF}}}\middle|\overline{\mathsf{HitGRange}_{M_i,n}^{\mathcal{O}_{\mathsf{TDF}}}}\right],$$

where the probability is taken over the choice of F_n.

For bounding the probability of the event $\mathsf{HitGRange}^{\mathcal{O}_{\mathsf{TDF}}}_{M_1,n}$, note that this event corresponds to the fact that M, without any information on $\mathsf{G}_n : \{0,1\}^n \to \{0,1\}^{2n}$ (since no oracle queries have been issued so far), manages to guess $\mathsf{pk}_1 \in \{0,1\}^{2n}$ for which there exists $\mathsf{td} \in \{0,1\}^n$ such that $\mathsf{G}_n(\mathsf{td}_1) = \mathsf{pk}$. Since the value pk_1 is fixed by the description of M, and we are now sampling $\mathsf{G}_n : \{0,1\}^n \to \{0,1\}^{2n}$ uniformly from the set of all functions mapping n-bit inputs to $2n$-bit outputs, we have that

$$\Pr\left[\mathsf{HitGRange}^{\mathcal{O}_{\mathsf{TDF}}}_{M_1,n}\right] \leq \frac{2^n}{4^n}.$$

For bounding the probability of the event $\mathsf{HitGRange}^{\mathcal{O}_{\mathsf{TDF}}}_{M_{i+1},n}$ given that $\overline{\mathsf{HitGRange}^{\mathcal{O}_{\mathsf{TDF}}}_{M_i,n}}$ occurred, we fix the queries and guesses $q_1, \ldots, q_i$. Let $\mathsf{td}_1, \ldots, \mathsf{td}_a$ be the queries to G_n and $\mathsf{pk}_{a+1}, \ldots, \mathsf{pk}_i$ be the guesses. We fix the answers $\mathsf{pk}_1, \ldots, \mathsf{pk}_a$ to the queries. By the assumption that M is deterministic we know that the next query or guess q_{i+1} is fixed. Assume that it is a guess pk_{i+1}. By the assumption that $\overline{\mathsf{HitGRange}^{\mathcal{O}_{\mathsf{TDF}}}_{M_i,n}}$ we know that $\mathsf{pk}_{a+1}, \ldots, \mathsf{pk}_i$ are not in the range of G, thus we are sampling a function $\{0,1\}^n \setminus \{\mathsf{td}_1, \ldots \mathsf{td}_a\} \to \{0,1\}^{2n} \setminus \{\mathsf{pk}_{a+1}, \ldots, \mathsf{pk}_i\}$ uniformly from the set of all functions on those domain and range. So we have that

$$\Pr\left[\mathsf{HitGRange}^{\mathcal{O}_{\mathsf{TDF}}}_{M_i,n}\right] \leq \frac{2^n - a}{4^n - (i-a)} \leq \frac{2^n}{4^n - i}.$$

We conclude that

$$\begin{aligned}
\Pr\left[\mathsf{HitGRange}^{\mathcal{O}_{\mathsf{TDF}}}_{M,n}\right] &\leq \Pr\left[\mathsf{HitGRange}^{\mathcal{O}_{\mathsf{TDF}}}_{M_1,n}\right] + \sum_{i=1}^{q-1} \Pr\left[\mathsf{HitGRange}^{\mathcal{O}_{\mathsf{TDF}}}_{M_{i+1},n} \middle| \overline{\mathsf{HitInv}^{\mathcal{O}_{\mathsf{TDF}}}_{M_i,n}}\right] \\
&\leq \sum_{i=0}^{q-1} \frac{2^n}{4^n - i} \\
&\leq \frac{q \cdot 2^n}{4^n - q} \\
&\leq \frac{q}{2^n - q}.
\end{aligned}$$

■

Acknowledgments. We thank Nir Bitansky, Tim Roughgarden, Omer Paneth, and the anonymous reviewers for their insightful comments and suggestions.

References

[AKV04] T. Abbot, D. Kane, and P. Valiant. On algorithms for Nash equilibria. Unpublished manuscript available at `http://web.mit.edu/tabbott/Public/final.pdf`, 2004.

[AS15] G. Asharov and G. Segev. Limits on the power of indistinguishability obfuscation and functional encryption. In *Proceedings of the 56th Annual IEEE Symposium on Foundations of Computer Science*, pages 191–209, 2015.

[AS16] G. Asharov and G. Segev. On constructing one-way permutations from indistinguishability obfuscation. In *Proceedings of the 13th Theory of Cryptography Conference*, pages 512–541, 2016.

[BCE+95] P. Beame, S. A. Cook, J. Edmonds, R. Impagliazzo, and T. Pitassi. The relative complexity of NP search problems. In *Proceedings of the 27th Annual ACM Symposium on Theory of Computing*, pages 303–314, 1995.

[BGH+15] Z. Brakerski, C. Gentry, S. Halevi, T. Lepoint, A. Sahai, and M. Tibouchi. Cryptanalysis of the quadratic zero-testing of GGH. Cryptology ePrint Archive, Report 2015/845, 2015.

[BGI+01] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. P. Vadhan, and K. Yang. On the (im)possibility of obfuscating programs. In *Advances in Cryptology – CRYPTO '01*, pages 1–18, 2001.

[BGI+12] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. P. Vadhan, and K. Yang. On the (im)possibility of obfuscating programs. *Journal of the ACM*, 59(2):6, 2012.

[BM09] B. Barak and M. Mahmoody-Ghidary. Merkle puzzles are optimal - An $O(n^2)$-query attack on any key exchange from a random oracle. In *Advances in Cryptology – CRYPTO '09*, pages 374–390, 2009.

[BPR15] N. Bitansky, O. Paneth, and A. Rosen. On the cryptographic hardness of finding a Nash equilibrium. In *Proceedings of the 56th Annual IEEE Symposium on Foundations of Computer Science*, pages 1480–1498, 2015.

[BPW16] N. Bitansky, O. Paneth, and D. Wichs. Perfect structure on the edge of chaos – trapdoor permutations from indistinguishability obfuscation. In *Proceedings of the 13th Theory of Cryptography Conference*, pages 474–502, 2016.

[CDT09] X. Chen, X. Deng, and S. Teng. Settling the complexity of computing two-player Nash equilibria. *Journal of the ACM*, 56(3), 2009.

[CFL+16] J. H. Cheon, P.-A. Fouque, C. Lee, B. Minaud, and H. Ryu. Cryptanalysis of the new CLT multilinear map over the integers. Cryptology ePrint Archive, Report 2016/135, 2016.

[CGH+15] J. Coron, C. Gentry, S. Halevi, T. Lepoint, H. K. Maji, E. Miles, M. Raykova, A. Sahai, and M. Tibouchi. Zeroizing without low-level zeroes: New MMAP attacks and their limitations. In *Advances in Cryptology – CRYPTO '15*, pages 247–266, 2015.

[CHL+15] J. H. Cheon, K. Han, C. Lee, H. Ryu, and D. Stehlé. Cryptanalysis of the multilinear map over the integers. In *Advances in Cryptology – EUROCRYPT '15*, pages 3–12, 2015.

[CIY97] S. A. Cook, R. Impagliazzo, and T. Yamakami. A tight relationship between generic oracles and type-2 complexity theory. *Information and Computation*, 137(2):159–170, 1997.

[CJL16] J. H. Cheon, J. Jeong, and C. Lee. An algorithm for NTRU problems and cryptanalysis of the GGH multilinear map without an encoding of zero. Cryptology ePrint Archive, Report 2016/139, 2016.

[CLR15] J. H. Cheon, C. Lee, and H. Ryu. Cryptanalysis of the new CLT multilinear maps. Cryptology ePrint Archive, Report 2015/934, 2015.

[DGP09] C. Daskalakis, P. W. Goldberg, and C. H. Papadimitriou. The complexity of computing a Nash equilibrium. *SIAM Journal on Computing*, 39(1):195–259, 2009.

[GGH+13] S. Garg, C. Gentry, S. Halevi, M. Raykova, A. Sahai, and B. Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *Proceedings of the 54th Annual IEEE Symposium on Foundations of Computer Science*, pages 40–49, 2013.

[Gol00] O. Goldreich. On security preserving reductions – revised terminology. Cryptology ePrint Archive, Report 2000/001, 2000.

[Gol01] O. Goldreich. Foundations of Cryptography – Volume 1: Basic Techniques. Cambridge University Press, 2001.

[GPS16] S. Garg, O. Pandey, and A. Srinivasan. Revisiting the cryptographic hardness of finding a Nash equilibrium. In *Advances in Cryptology – CRYPTO '16*, pages 579–604, 2016.

[HHR+15] I. Haitner, J. J. Hoch, O. Reingold, and G. Segev. Finding collisions in interactive protocols – Tight lower bounds on the round and communication complexities of statistically hiding commitments. *SIAM Journal on Computing*, 44(1):193–242, 2015.

[HJ15] Y. Hu and H. Jia. Cryptanalysis of GGH map. Cryptology ePrint Archive, Report 2015/301, 2015.

[HNY17] P. Hubácek, M. Naor, and E. Yogev. The journey from NP to TFNP hardness. In *Proceedings of the 8th Innovations in Theoretical Computer Science Conference*, 2017.

[HPV89] M. D. Hirsch, C. H. Papadimitriou, and S. A. Vavasis. Exponential lower bounds for finding brouwer fix points. *Journal of Complexity*, 5(4):379–416, 1989.

[IR89] R. Impagliazzo and S. Rudich. Limits on the provable consequences of one-way permutations. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing*, pages 44–61, 1989.

[Lub96] M. Luby. Pseudorandomness and Cryptographic Applications. Princeton University Press, 1996.

[MF15] B. Minaud and P.-A. Fouque. Cryptanalysis of the new multilinear map over the integers. Cryptology ePrint Archive, Report 2015/941, 2015.

[MSZ16] E. Miles, A. Sahai, and M. Zhandry. Annihilation attacks for multilinear maps: Cryptanalysis of indistinguishability obfuscation over GGH13. Cryptology ePrint Archive, Report 2016/147, 2016.

[Pap94] C. H. Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. *Journal of Computer and System Sciences*, 48(3):498–532, 1994.

[RTV04] O. Reingold, L. Trevisan, and S. P. Vadhan. Notions of reducibility between cryptographic primitives. In *Proceedings of the 1st Theory of Cryptography Conference*, pages 1–20, 2004.

[Rud88] S. Rudich. Limits on the Provable Consequences of One-way Functions. PhD thesis, EECS Department, University of California, Berkeley, 1988.

[Sim98] D. R. Simon. Finding collisions on a one-way street: Can secure hash functions be based on general assumptions? In *Advances in Cryptology – EUROCRYPT '98*, pages 334–345, 1998.

[SvS04] R. Savani and B. von Stengel. Exponentially many steps for finding a Nash equilibrium in a bimatrix game. In *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science*, pages 258–267, 2004.

[SW14] A. Sahai and B. Waters. How to use indistinguishability obfuscation: Deniable encryption, and more. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*, pages 475–484, 2014.

A Average-Case SVL Hardness and OWFs Do Not Imply Key Agreement

Based on the techniques developed in Section 3, we show that average-case SVL hardness is useless for constructing a key-agreement protocol in a black-box manner, even when assuming the existence of one-way functions. Specifically, we show that in any black-box construction of a key-agreement protocol based on a one-way function and a hard-on-average distribution of SVL instances, we can eliminate the protocol's need for using the SVL instances. This leads to a black-box construction of key-agreement protocol based on a one-way function, which we can then rule out by invoking the classic result of Impagliazzo and Rudich [IR89] and its refinement by Barak and Mahmoody-Ghidary [BM09].

In this section we model a one-way function as a sequence $f = \{f_n\}_{n\in\mathbb{N}}$, where for every $n \in \mathbb{N}$ it holds that $f_n : \{0,1\}^n \to \{0,1\}^n$. The following definition tailors the standard notion of a fully black-box construction to the specific primitives under consideration.

Definition A.1. A fully black-box construction of a bit-agreement protocol with correctness $\rho = \rho(n)$ from a one-way function and a hard-on-average distribution of SVL instances consists of a pair of oracle-aided polynomial-time algorithm $(\mathcal{A}, \mathcal{B})$, an oracle-aided algorithm M that runs in time $T_M(\cdot)$, and functions $\epsilon_{M,1}(\cdot)$ and $\epsilon_{M,2}(\cdot)$, such that the following conditions hold:

- **Correctness**: For any function $f = \{f_n\}_{n\in\mathbb{N}}$, for any valid SVL instance $\mathcal{O}_{\mathsf{SVL}}$, and for any $n \in \mathbb{N}$ it holds that

$$\Pr_{r_{\mathcal{A}},r_{\mathcal{B}}}\left[\mathsf{k}_{\mathcal{A}} = \mathsf{k}_{\mathcal{B}} \,\middle|\, (\mathsf{k}_{\mathcal{A}}, \mathsf{k}_{\mathcal{B}}, \mathsf{Trans}) \leftarrow \langle \mathcal{A}^{f,\mathcal{O}_{\mathsf{SVL}}}(1^n; r_{\mathcal{A}}), \mathcal{B}^{f,\mathcal{O}_{\mathsf{SVL}}}(1^n; r_{\mathcal{B}})\rangle\right] \geq \frac{1}{2} + \rho(n).$$

- **Black-box proof of security:** For any function $f = \{f_n\}_{n\in\mathbb{N}}$, for any valid SVL instance $\mathcal{O}_{\mathsf{SVL}} = \{(\mathsf{Gen}_n, \mathsf{S}_n, \mathsf{V}_n, L(n))\}_{n\in\mathbb{N}}$, for any oracle-aided algorithm E that runs in time $T_E(\cdot)$, and for any function $\epsilon_E(\cdot)$, if

$$\left|\Pr\left[\mathsf{Exp}^{\mathsf{KA}}_{(\mathcal{A}^{f,\mathcal{O}_{\mathsf{SVL}}},\mathcal{B}^{f,\mathcal{O}_{\mathsf{SVL}}}),E^{f,\mathcal{O}_{\mathsf{SVL}}}}(n) = 1\right] - \frac{1}{2}\right| \geq \epsilon_E(n)$$

  for infinitely many values of $n \in \mathbb{N}$ (recall Definition 2.8 for the description of the experiment $\mathsf{Exp}^{\mathsf{KA}}_{(\mathcal{A}^{f,\mathcal{O}_{\mathsf{SVL}}},\mathcal{B}^{f,\mathcal{O}_{\mathsf{SVL}}}),E^{f,\mathcal{O}_{\mathsf{SVL}}}}$), then either

$$\Pr\left[M^{E,f,\mathcal{O}_{\mathsf{SVL}}}\left(f_n(x)\right) \in f_n^{-1}\left(f_n(x)\right)\right] \geq \epsilon_{M,1}\left(T_E(n)/\epsilon_E(n)\right) \cdot \epsilon_{M,2}(n)$$

  for infinitely many values of $n \in \mathbb{N}$, where the probability is taken over the choice of $x \leftarrow \{0,1\}^n$ and over the internal randomness of M, or

$$\Pr\left[M^{E,f,\mathcal{O}_{\mathsf{SVL}}}\left(1^n, \sigma\right) \text{ solves } \left(\mathsf{S}_n(\sigma,\cdot), \mathsf{V}_n(\sigma,\cdot), L(n)\right)\right] \geq \epsilon_{M,1}\left(T_E(n)/\epsilon_E(n)\right) \cdot \epsilon_{M,2}(n)$$

  for infinitely many values of $n \in \mathbb{N}$, where the probability is taken over the choice of $\sigma \leftarrow \mathsf{Gen}_n()$ and over the internal randomness of M.

As in Definition 3.1, we split the security loss in the above definition to an adversary-dependent security loss and an adversary-independent security loss, as this allows us to capture constructions where one of these losses is super-polynomial whereas the other is polynomial. Equipped with the above definition we prove the following theorem:

Theorem A.2. *Let $(\mathcal{A}, \mathcal{B}, M, T_M, \epsilon_{M,1}, \epsilon_{M,2})$ be a fully black-box construction of a bit-agreement protocol with correctness $\rho(n) = 1/\mathsf{poly}(n)$, for some (arbitrary) polynomial $\mathsf{poly}(n)$, from a one-way function and a hard-on-average SVL instance. Then, at least one of the following properties holds:*

1. *$T_M(n) \geq 2^{\zeta n}$ for some constant $\zeta > 0$ (i.e., the reduction runs in exponential time).*
2. *$\epsilon_{M,1}(n^c) \cdot \epsilon_{M,2}(n) \leq 2^{-n/10}$ for some constant $c > 1$ (i.e., the security loss is exponential).*

As with Theorem 3.2, also here Theorem A.2 rules out (in particular) standard "polynomial-time polynomial-loss" reductions. More generally, the theorem implies that if the running time $T_M(\cdot)$ of the reduction is sub-exponential and the adversary-dependent security loss $\epsilon_{M,1}(\cdot)$ is polynomial (as expected), then the adversary-independent security loss $\epsilon_{M,2}(\cdot)$ must be exponential (thus even ruling out constructions based on one-way function and SVL instances with sub-exponential hardness).

A.1 Proof Overview

In what follows we first describe the oracles, denoted f and $\mathcal{O}_{\mathsf{SVL}}$, on which we rely for proving Theorem A.2, and show that they indeed implement a one-way function and a hard-on-average distribution of SVL instances, respectively. Then, we show that any bit-agreement protocol that uses the oracles f and $\mathcal{O}_{\mathsf{SVL}}$ can be attacked. For the remainder of this section we remind the reader that a *q-query algorithm* is an oracle-aided algorithm A such that for any oracle $\mathcal{O}$ and input $x \in \{0,1\}^*$, the computation $A^{\mathcal{O}}(x)$ consists of at most $q(|x|)$ oracle calls to $\mathcal{O}$.

The oracles f and $\mathcal{O}_{\mathsf{SVL}}$. The oracle f is a sequence $\{f_n\}_{n\in\mathbb{N}}$ where for every $n \in \mathbb{N}$ the function $f_n : \{0,1\}^n \to \{0,1\}^n$ is sampled uniformly from the set of all functions mapping n-bit inputs to n-bit outputs. The oracle $\mathcal{O}_{\mathsf{SVL}}$, sampled independently of f, is as defined in Section 3.1. That is, it is a valid SVL instance $\{(\mathsf{S}_n, \mathsf{V}_n, L(n))\}_{n\in\mathbb{N}}$ that is sampled via the following process for every $n \in \mathbb{N}$:

- Let $L(n) = 2^{n/2}$, $x_0 = 0^n$, and uniformly sample distinct elements $x_1, \ldots, x_{L(n)} \leftarrow \{0,1\}^n \setminus \{0^n\}$.
- The successor function $\mathsf{S}_n : \{0,1\}^n \to \{0,1\}^n$ is defined as

$$\mathsf{S}_n(x) = \begin{cases} x_{i+1} & \text{if } x = x_i \text{ for some } i \in \{0, \ldots, L(n)-1\} \\ x & \text{otherwise} \end{cases} .$$

- The verification function $\mathsf{V}_n : \{0,1\}^n \times [2^n] \to \{0,1\}$ is defined in a manner that is consistent with S_n (i.e., V_n is defined such that the instance is valid).

The oracles f and $\mathcal{O}_{\mathsf{SVL}}$ are sampled independently, and therefore we immediately obtain the following two corollaries from Claims 3.3 and 5.3 (the first corollary states that f is indeed hard to invert relative to f and $\mathcal{O}_{\mathsf{SVL}}$, and the second corollary A.4 states that $\mathcal{O}_{\mathsf{SVL}}$ is indeed a hard-on-average SVL instance relative to f and $\mathcal{O}_{\mathsf{SVL}}$):

Corollary A.3. *For any fixing of the oracle $\mathcal{O}_{\mathsf{SVL}}$, and for any $q(n)$-query algorithm M, it holds that*

$$\Pr\left[M^{f,\mathcal{O}_{\mathsf{SVL}}}\left(f_n(x)\right) \in f_n^{-1}\left(f_n(x)\right)\right] \leq \frac{2(q(n)+1)}{2^n - q(n)}$$

for all sufficiently large $n \in \mathbb{N}$, where the probability is taken over the choice of $x \leftarrow \{0,1\}^n$, and over the choice of the oracle $f = \{f_n\}_{n\in\mathbb{N}}$ as described above.

Corollary A.4. *For any fixing of the oracle f, and for any $q(n)$-query algorithm M, where $q(n) \leq L(n) - 1$, it holds that*

$$\Pr\left[M^{f,\mathcal{O}_{\mathsf{SVL}}}(1^n) \text{ solves } (\mathsf{S}_n, \mathsf{V}_n, L(n))\right] \leq \frac{(q(n)+1)\cdot L(n)}{2^n - q(n) - 1}$$

for all sufficiently large $n \in \mathbb{N}$, where the probability is taken over the choice of the oracle $\mathcal{O}_{\mathsf{SVL}} = \{(\mathsf{S}_n, \mathsf{V}_n, L(n))\}_{n\in\mathbb{N}}$ as described above.

Attacking bit-agreement protocols relative to f and $\mathcal{O}_{\mathsf{SVL}}$. We show that for any oracle-aided bit-agreement protocol $(\mathcal{A}, \mathcal{B})$ with correctness $\rho(n) = 1/\mathsf{poly}(n)$, in which the parties issue at most $q(n)$ oracle queries, and for any $\delta = \delta(n) > 0$, there exists an attacker that issues roughly q^2/δ^2 oracle queries, whose output agrees with Alice's output with probability $1/2 + \rho(n) - \delta(n)$. We prove the following claim:

Claim A.5. *Let $(\mathcal{A}, \mathcal{B})$ be an oracle-aided bit-agreement protocol, in which the parties issue at most $q = q(n)$ oracle queries, where the input for each query is of length at most $q(n)$ bits, and assume that*

$$\Pr_{\substack{f,\mathcal{O}_{\mathsf{SVL}}\\ r_{\mathcal{A}},r_{\mathcal{B}}}}\left[\mathsf{k}_{\mathcal{A}} = \mathsf{k}_{\mathcal{B}} \,\middle|\, (\mathsf{k}_{\mathcal{A}}, \mathsf{k}_{\mathcal{B}}, \mathsf{Trans}) \leftarrow \langle \mathcal{A}^{f,\mathcal{O}_{\mathsf{SVL}}}(1^n; r_{\mathcal{A}}), \mathcal{B}^{f,\mathcal{O}_{\mathsf{SVL}}}(1^n; r_{\mathcal{B}})\rangle\right] \geq \frac{1}{2} + \rho(n)$$

for all sufficiently large $n \in \mathbb{N}$ and for some function $\rho(n) > 0$. Then, for any $\delta = \delta(n) > 0$, there exists an $\widetilde{O}(q^2/\delta^2)$-query algorithm E, such that

$$\left|\Pr\left[\mathsf{Exp}^{\mathsf{KA}}_{(\mathcal{A}^{f,\mathcal{O}_{\mathsf{SVL}}}, \mathcal{B}^{f,\mathcal{O}_{\mathsf{SVL}}}), E^{f,\mathcal{O}_{\mathsf{SVL}}}}(n) = 1\right] - \frac{1}{2}\right| \geq \rho(n) - \delta(n)$$

for all sufficiently large $n \in \mathbb{N}$, where the probability is taken over the choice of the oracles f and $\mathcal{O}_{\mathsf{SVL}}$, and over the internal randomness of $\mathcal{A}$ and $\mathcal{B}$. Moreover, the algorithm E can be implemented in time polynomial in n, $q(n)$ and $1/\delta(n)$ given access to a PSPACE*-complete oracle.*

The proof of the claim, which is provided below, is based on adapting the approach underlying our proof of Claim 3.4 to the setting of key-agreement protocols, and then invoking the classic result of Impagliazzo and Rudich [IR89] and its refinement by Barak and Mahmoody-Ghidary [BM09]. Specifically, as discussed in Section 1.3, during an execution $(\mathcal{A}^{f,\mathcal{O}_{\mathsf{SVL}}}, \mathcal{B}^{f,\mathcal{O}_{\mathsf{SVL}}})$ of a given bit-agreement protocol, with an overwhelming probability over the choice of the oracle $\mathcal{O}_{\mathsf{SVL}}$, the parties $\mathcal{A}$ and $\mathcal{B}$ should not query $\mathcal{O}_{\mathsf{SVL}}$ with any elements on the line $0^n \to x_1 \to \cdots \to x_{L(n)}$ except for the first q elements $x_0, x_1, \ldots, x_{q-1}$. This gives rise to a bit-agreement protocol $(\widetilde{\mathcal{A}}^f, \widetilde{\mathcal{B}}^f)$ that does not require access to the oracle $\mathcal{O}_{\mathsf{SVL}}$: First, $\widetilde{A}$ samples a sequence $x_1, \ldots, x_q$ of q values, and sends these values to $\widetilde{B}$. Then, $\widetilde{\mathcal{A}}$ and $\widetilde{\mathcal{B}}$ run the protocol $(\mathcal{A}^{f,\mathcal{O}_{\mathsf{SVL}}}, \mathcal{B}^{f,\mathcal{O}_{\mathsf{SVL}}})$ by using the values $x_1, \ldots, x_q$ instead of accessing $\mathcal{O}_{\mathsf{SVL}}$. At this point, we have a bit-agreement protocol where the parties have access only to a random function f, and thus we can apply the attacks of Impagliazzo and Rudich [IR89] and Barak and Mahmoody-Ghidary [BM09], which we can translate back to attacks on the underlying protocol. The proof of Theorem A.2 then follows from Corollaries A.3 and A.4 and Claim A.5 in a manner identical to the proof of Theorem 3.2 (see Section 3.4).

A.2 Attacking Key-Agreement Protocols Relative to f and $\mathcal{O}_{\mathsf{SVL}}$

In this section we prove Claim A.5. We start by defining an event capturing the above intuition of "hitting" elements on the line sampled for $\mathcal{O}_{\mathsf{SVL}}$, similarly to event defined in Section 3.

The event $\mathsf{HIT}^{f,\mathcal{O}_{\mathsf{SVL}}}_{M,n}$. Let the oracles f and $\mathcal{O}_{\mathsf{SVL}} = \{(\mathsf{S}_n, \mathsf{V}_n, L(n))\}_{n\in\mathbb{N}}$ be distributed as described in Section A.1. Let M be a q-query algorithm. We fix some $n \in \mathbb{N}$, and consider only the queries made to S_n and V_n. We denote by α_i the random variable corresponding to M's ith oracle query if this is an S_n-query, and denote by (α_i, k_i) the random variable corresponding to M's ith oracle query if this is a V_n-query. Let $x_0, \ldots, x_{L(n)}$ be the line sampled for $(\mathsf{S}_n, \mathsf{V}_n, L(n))$. As in Section 3, we denote by $\mathsf{HIT}^{f,\mathcal{O}_{\mathsf{SVL}}}_{M,n}$ the event in which there exist indices j and $i \in [L(n)]$ for which $\alpha_j = x_i$ but $x_{i-1} \notin \{\alpha_1, \ldots, \alpha_{j-1}\}$. That is, this is the event in which M queries $(\mathcal{O}_{\mathsf{SVL}})_n$ with some x_i *before* querying it on x_{i-1}. In particular, note that if the event $\mathsf{HIT}^{f,\mathcal{O}_{\mathsf{SVL}}}_{M,n}$ does not occur, then M does not query $(\mathcal{O}_{\mathsf{SVL}})_n$ with x_i for $i \in \{q, \ldots, L(n)\}$. Since the oracle $\mathcal{O}_{\mathsf{SVL}}$ is sampled independently of the oracle f, we deduce the following corollary from Claim 3.5:

Corollary A.6. *For any fixing of the oracle f, for any q-query algorithm M, and for any $n \in \mathbb{N}$, it holds that*

$$\Pr\left[\mathsf{HIT}^{f,\mathcal{O}_{\mathsf{SVL}}}_{M,n}\right] \leq \frac{q \cdot L(n)}{2^n - q}$$

where the probability is taken over the choice of the oracle $\mathcal{O}_{\mathsf{SVL}} = \{(\mathsf{S}_n, \mathsf{V}_n, L(n))\}_{n\in\mathbb{N}}$. Moreover, q can be a bound on the number of calls to S_n and V_n.

Removing the oracle $\mathcal{O}_{\mathsf{SVL}}$. Let $(\mathcal{A}, \mathcal{B})$ be an oracle-aided bit-agreement protocol as in Claim A.5. For a loss parameter $\epsilon = \epsilon(n) > 0$, we define an oracle-aided bit-agreement protocol $(\widetilde{\mathcal{A}}, \widetilde{\mathcal{B}})$ that on input security parameter 1^n, and with oracle access to f *only*, works as follows. First, $\widetilde{\mathcal{A}}$ performs the following initialization routine:

1. Set $a(n) = 2\log(q(n)^2/\epsilon(n) + 1)$.
2. For $1 \leq i \leq a(n)$:
   (a) Set $x^i_0 = 0^i$.
   (b) Uniformly sample distinct elements $x^i_1, \ldots, x^i_{L(i)} \leftarrow \{0,1\}^i \setminus \{0^i\}$.
   (c) Send the elements $x^i_1, \ldots, x^i_{L(i)}$ to $\widetilde{\mathcal{B}}$.
   (d) Define the successor function $\widetilde{\mathsf{S}}_i : \{0,1\}^i \to \{0,1\}^i$ as

$$\widetilde{\mathsf{S}}_i(x) = \begin{cases} x^i_{j+1} & \text{if } x = x^i_j \text{ for some } j \in \{0, \ldots, L(i)-1\} \\ x & \text{otherwise} \end{cases},$$

   and define the verification function $\widetilde{\mathsf{V}}_i : \{0,1\}^i \times [2^i] \to \{0,1\}$ in a manner that is consistent with $\widetilde{\mathsf{S}}_i$.
3. For $a(n) < i \leq q(n)$:
   (a) Set $x^i_0 = 0^i$.
   (b) Uniformly sample distinct elements $x^i_1, \ldots, x^i_{q(n)} \leftarrow \{0,1\}^i \setminus \{0^i\}$.
   (c) Send the elements $x^i_1, \ldots, x^i_{q(n)}$ to $\widetilde{\mathcal{B}}$.
   (d) Define the successor function $\widetilde{\mathsf{S}}_i : \{0,1\}^i \to \{0,1\}^i$ as

$$\widetilde{\mathsf{S}}_i(x) = \begin{cases} x^i_{j+1} & \text{if } x = x^i_j \text{ for some } j \in \{0, \ldots, q(n)-1\} \\ x & \text{otherwise} \end{cases},$$

and define the verification function $\widetilde{\mathsf{V}}_i : \{0,1\}^i \times [2^i] \to \{0,1\}$ in a manner that is consistent with $\widetilde{\mathsf{S}}_i$.

Next, $\widetilde{\mathcal{A}}$ and $\widetilde{\mathcal{B}}$ emulate the protocol $\langle \mathcal{A}(1^n), \mathcal{B}(1^n) \rangle$ with respect to the oracle f and the "fake" oracle $\widetilde{\mathcal{O}_{\mathsf{SVL}}} = \{(\widetilde{\mathsf{S}}_i, \widetilde{\mathsf{V}}_i, L(i))\}_{i=1}^{q(n)}$, and output the outputs of $\mathcal{A}$ and $\mathcal{B}$, respectively. We name this phase the emulation phase. Note that by our assumption, $\mathcal{A}$ and $\mathcal{B}$ do not query $(\widetilde{\mathcal{O}_{\mathsf{SVL}}})_i$ for $i > q(n)$, so it is okay to leave it undefined. After emulating the protocol, $\widetilde{\mathcal{A}}$ and $\widetilde{\mathcal{B}}$ output what $\mathcal{A}$ and $\mathcal{B}$ output respectively. Note that in the protocol $(\widetilde{\mathcal{A}}, \widetilde{B})$, the parties issue at most $q(n)$ queries. Also, note that in the initialization phase, $\widetilde{\mathcal{A}}$ draws $\sum_{i=1}^{\lfloor a(n) \rfloor} L(i) + q(n) \cdot (q(n) - \lfloor a(n) \rfloor)$ samples, and it holds that

$$\sum_{i=1}^{\lfloor a(n) \rfloor} L(i) + q(n) \cdot (q(n) - \lfloor a(n) \rfloor) \leq a(n) \cdot 2^{a(n)/2} + q(n)^2 = \widetilde{O}(q(n)^2/\epsilon(n))$$

Coupling the protocols. Consider the executions $(\mathsf{k}_{\mathcal{A}}, \mathsf{k}_{\mathcal{B}}, \mathsf{Trans}) \leftarrow \langle \mathcal{A}^{f,\mathcal{O}_{\mathsf{SVL}}}(1^n; r_{\mathcal{A}}), \mathcal{B}^{f,\mathcal{O}_{\mathsf{SVL}}}(1^n; r_{\mathcal{B}}) \rangle$ and $(\mathsf{k}_{\widetilde{\mathcal{A}}}, \mathsf{k}_{\widetilde{\mathcal{B}}}, \widetilde{\mathsf{Trans}}) \leftarrow \langle \widetilde{\mathcal{A}}^f(1^n; r_{\widetilde{\mathcal{A}}}), \widetilde{\mathcal{B}}^f(1^n; r_{\widetilde{\mathcal{B}}}) \rangle$, where f and $\mathcal{O}_{\mathsf{SVL}}$ are sampled as described above. We couple these executions in the following way:[9]

- We sample and use the same oracle f for both executions.
- The randomness of $\widetilde{A}$ can be split into two part $r_{\widetilde{\mathcal{A}}} = (r_{\widetilde{\mathcal{A}},1}, r_{\widetilde{\mathcal{A}},2})$, where $r_{\widetilde{\mathcal{A}},1}$ is the randomness used in the initialization phase, and $r_{\widetilde{\mathcal{A}},2}$ is the randomness used in the emulation phase.
- We couple the randomness of the emulation phase with the randomness of the actual execution of $(\mathcal{A}, \mathcal{B})$ by $r_{\widetilde{\mathcal{A}},2} = r_{\mathcal{A}}$ and $r_{\widetilde{\mathcal{B}}} = r_{\mathcal{B}}$.
- We couple the oracle $\mathcal{O}_{\mathsf{SVL}}$ with $r_{\widetilde{\mathcal{A}},1}$ (hence with $\widetilde{\mathcal{O}_{\mathsf{SVL}}}$) as follows:
  - For $1 \leq i \leq a(n)$, we remind that $\widetilde{\mathcal{A}}$ uniformly samples distinct elements $x_1^i, \ldots, x_{L(i)}^i \leftarrow \{0,1\}^i \setminus \{0^i\}$, and that $x_0^i = 0^i$. So we set

$$\mathsf{S}_i(x) = \begin{cases} x_{j+1}^i & \text{if } x = x_j^i \text{ for some } j \in \{0, \ldots, L(i)-1\} \\ x & \text{otherwise} \end{cases},$$

    and set V_i in a manner consistent with S_i. As a result $(\mathcal{O}_{\mathsf{SVL}})_i = (\widetilde{\mathcal{O}_{\mathsf{SVL}}})_i$.
  - For $a(n) < i \leq q(n)$, we remind that $\widetilde{\mathcal{A}}$ uniformly samples distinct elements $x_1^i, \ldots, x_{q(n)}^i \leftarrow \{0,1\}^i \setminus \{0^i\}$, and that $x_0^i = 0^i$. So we uniformly sample distinct elements $x_{q(n)+1}^i, \ldots, x_{L(i)}^i \leftarrow \{0,1\}^i \setminus \{0^i, x_1^i, \ldots, x_{q(n)}^i\}$, set

$$\mathsf{S}_i(x) = \begin{cases} x_{j+1}^i & \text{if } x = x_j^i \text{ for some } j \in \{0, \ldots, L(i)-1\} \\ x & \text{otherwise} \end{cases},$$

    and set V_i in a manner consistent with S_i. As a result, the line of $(\widetilde{\mathcal{O}_{\mathsf{SVL}}})_i$ is a prefix of the line of $(\mathcal{O}_{\mathsf{SVL}})_i$.
  - For $i > q(n)$, $(\mathcal{O}_{\mathsf{SVL}})_i$ is sampled without any coupling.

We split the transcript of the execution of $(\widetilde{A}, \widetilde{B})$ into two parts $\widetilde{\mathsf{Trans}} = (\widetilde{\mathsf{Trans}}_1, \widetilde{\mathsf{Trans}}_2)$ where $\widetilde{\mathsf{Trans}}_1$ is the transcript of the initialization phase, and $\widetilde{\mathsf{Trans}}_2$ is the transcript of the emulation phase. Denote by $\mathsf{Same} = \mathsf{Same}_{(\mathcal{A},\mathcal{B}),n}$ the event in which $(k_{\mathcal{A}}, k_{\mathcal{B}}, \mathsf{Trans}) = (k_{\widetilde{A}}, k_{\widetilde{B}}, \widetilde{\mathsf{Trans}}_2)$ holds.

[9] To couple two probability distributions means to define a joint distribution whose marginals are exactly those two distributions.

We now estimate $\Pr[\mathsf{Same}]$. If for every $a(n) \leq i \leq q(n)$, $\mathsf{HIT}^{f,\mathcal{O}_{\mathsf{SVL}}}_{\langle \mathcal{A}(1^n),\mathcal{B}(1^n)\rangle,i}$ does no occur, then the emulation phase and the actual execution of $(\mathcal{A},\mathcal{B})$ behave the same, so Same occurs. Hence,

$$\begin{aligned}
\Pr[\neg\mathsf{Same}] &\leq \sum_{i=a(n)}^{q(n)} \Pr_{\mathcal{A},\mathcal{B},f,\mathcal{O}_{\mathsf{SVL}}}\left[\mathsf{HIT}^{f,\mathcal{O}_{\mathsf{SVL}}}_{\langle \mathcal{A}(1^n),\mathcal{B}(1^n)\rangle,i}\right] \\
&\leq \sum_{i=a(n)}^{q(n)} \frac{q(n)\cdot L(i)}{2^i - q(n)} \\
&\leq \sum_{i=a(n)}^{q(n)} \frac{q(n)}{2^{i/2}-1} \\
&\leq \frac{q(n)^2}{2^{a(n)/2}-1} \\
&= \epsilon(n).
\end{aligned}$$

In particular, it holds that

$$\Pr[\mathsf{k}_{\widetilde{\mathcal{A}}} = \mathsf{k}_{\widetilde{\mathcal{B}}}] \geq \Pr[\mathsf{k}_{\mathcal{A}} = \mathsf{k}_{\mathcal{B}}] - \Pr[\neg\mathsf{Same}] \geq \frac{1}{2} + \rho(n) - \epsilon(n)$$

The adversary E. For defining the adversary E for attacking the protocol $(\mathcal{A},\mathcal{B})$, we make use of the aforementioned result of Barak and Mahmoody-Ghidary.

Theorem A.7 ([IR89],[BM09])**.** *Let $(\widetilde{\mathcal{A}},\widetilde{\mathcal{B}})$ be an oracle-aided bit-agreement protocol, in which the parties issue at most $q = q(n)$ oracle queries*[10]*. Suppose that*

$$\Pr_{f,\widetilde{\mathcal{A}},\widetilde{\mathcal{B}}}[\mathsf{k}_{\widetilde{\mathcal{A}}} = \mathsf{k}_{\widetilde{\mathcal{B}}}] \geq \frac{1}{2} + \rho(n)$$

where the oracle f is sampled as above, and $(\mathsf{k}_{\widetilde{\mathcal{A}}}, \mathsf{k}_{\widetilde{\mathcal{B}}}, \widetilde{\mathsf{Trans}}) \leftarrow \langle \widetilde{\mathcal{A}}^f(1^n), \widetilde{\mathcal{B}}^f(1^n)\rangle$. Let $0 < \delta(n) < \frac{1}{2} + \rho(n)$. Then, there exists a $(16q/\delta)^2$-query adversary $\widetilde{E}$ such that

$$\Pr_{f,\widetilde{\mathcal{A}},\widetilde{\mathcal{B}}}[\mathsf{k}_{\widetilde{\mathcal{A}}} = \widetilde{E}^f(\widetilde{\mathsf{Trans}})] \geq \frac{1}{2} + \rho - \delta$$

Moreover, the algorithm $\widetilde{E}$ can be implemented in time polynomial in n, q and $1/\delta$ given access to a PSPACE*-complete oracle.*

Now, let $\widetilde{E}$ be the adversary from Theorem A.7 applied to our constructed protocol $(\widetilde{\mathcal{A}},\widetilde{\mathcal{B}})$, with loss of $\delta(n) = \epsilon(n)$. We define an adversary E to the protocol $(\mathcal{A},\mathcal{B})$, that on input Trans, and with oracle access to f and $\mathcal{O}_{\mathsf{SVL}} = \{(\mathsf{S}_n, \mathsf{V}_n, L(n))\}_{n\in\mathbb{N}}$, works as follows:

1. Set $a(n) = 2\log(q(n)^2/\epsilon(n) + 1)$.
2. Initialize an empty transcript $\widehat{\mathsf{Trans}}$.
3. For $1 \leq i \leq a(n)$:
   (a) Set $x_0^i = 0^i$.
   (b) For $j = 1,\ldots,L(i)$: Set $x_j^i = \mathsf{S}_i(x_{j-1}^i)$.

[10] In fact, it is enough to require that each party issues at most q queries.

(c) Append $x^i_1, \ldots, x^i_{L(i)}$ to the transcript $\widehat{\mathsf{Trans}}$ as they were send from Alice to Bob.

(d) Define the successor function $\widehat{\mathsf{S}}_i : \{0,1\}^i \to \{0,1\}^i$ as

$$\widehat{\mathsf{S}}_i(x) = \begin{cases} x^i_{j+1} & \text{if } x = x^i_j \text{ for some } j \in \{0, \ldots, L(i)-1\} \\ x & \text{otherwise} \end{cases} .$$

(e) Define the verification function $\widehat{\mathsf{V}}_i : \{0,1\}^i \times [2^i] \to \{0,1\}$ in a manner that is consistent with $\widehat{\mathsf{S}}_i$.

4. For $a(n) < i \leq q(n)$:

(a) Set $x^i_0 = 0^i$.

(b) For $j = 1, \ldots, q(n)$: Set $x^i_j = \mathsf{S}_i(x^i_{j-1})$.

(c) Append $x^i_1, \ldots, x^i_{q(n)}$ to the transcript $\widehat{\mathsf{Trans}}$ as they were send from Alice to Bob.

(d) Define the successor function $\widehat{\mathsf{S}}_i : \{0,1\}^i \to \{0,1\}^i$ as

$$\widehat{\mathsf{S}}_i(x) = \begin{cases} x^i_{j+1} & \text{if } x = x^i_j \text{ for some } j \in \{0, \ldots, q(n)-1\} \\ x & \text{otherwise} \end{cases} .$$

(e) Define the verification function $\widehat{\mathsf{V}}_i : \{0,1\}^i \times [2^i] \to \{0,1\}$ in a manner that is consistent with $\widehat{\mathsf{S}}_i$.

5. Run $\mathsf{k}_E \leftarrow \widetilde{E}^f((\widehat{\mathsf{Trans}}, \mathsf{Trans}))$ and output k_E.

Note that due to our coupling, the definition of x^i_j in the algorithm is consistent with the above definition of x^i_j as elements that $\widetilde{\mathcal{A}}$ samples. Also, by our coupling of $\mathcal{O}_{\mathsf{SVL}}$ and $r_{\widetilde{A},1}$, it holds that $\widetilde{\mathsf{Trans}}_1 = \widehat{\mathsf{Trans}}$. Furthermore, if the event Same occurs then it holds that $\widetilde{\mathsf{Trans}}_2 = \mathsf{Trans}$. Therefore, in that case the execution of $\mathsf{k}_E \leftarrow \widetilde{E}^f((\widehat{\mathsf{Trans}}, \mathsf{Trans}))$ is the same as $\mathsf{k}_{\widetilde{E}} \leftarrow \widetilde{E}^f((\widetilde{\mathsf{Trans}}_1, \widetilde{\mathsf{Trans}}_2))$, and we have

$$\Pr[\mathsf{k}_E \neq \mathsf{k}_{\mathcal{A}}] \leq \Pr[\mathsf{k}_{\widetilde{E}} \neq \mathsf{k}_{\widetilde{\mathcal{A}}}] + \Pr[\neg\mathsf{Same}] \leq \left(\frac{1}{2} - \rho(n) + \epsilon(n)\right) + \epsilon(n),$$

So it holds that $\Pr[\mathsf{k}_E = \mathsf{k}_{\mathcal{A}}] \geq \frac{1}{2} + \rho(n) - 2 \cdot \epsilon(n)$, and we choose $\epsilon(n) = \delta(n)/2$ where $\delta(n)$ is the desired loss from Claim A.5. The number of oracle queries that E performs is at most

$$\widetilde{O}(q(n)^2/\delta(n)) + (32q(n)/\delta(n))^2 \leq \widetilde{O}(q(n)^2/\delta(n)^2).$$

Moreover, given oracle access a PSPACE-complete oracle, the algorithm E can be implemented to run in time polynomial in n, q and $1/\delta$. This easily follows from Theorem A.7 and settles the proof of Claim A.5.

ECCC ISSN 1433-8092
https://eccc.weizmann.ac.il