FACULTY OF MATHEMATICS AND PHYSICS
Charles University

DOCTORAL THESIS

Veronika Slívová

On the Complexity of Search Problems with a Unique Solution

Computer Science Institute of Charles University

Supervisor of the doctoral thesis:	Mgr. Pavel Hubáček, Ph.D.
Study programme:	Computer Science
Study branch:	Theory of Computing, Discrete Models and Optimization

Prague 2021

Dedication. I would like to thank my advisor Pavel Hubáček for his guidance and a lot of valuable comments. I am grateful to all the collaborators with whom I could work on the results presented in this thesis, namely Bernd Gärtner, Thomas Duelhom Hansen, Arka Rai Chodhouri, Chethan Kamath, Karel Král and Hagar Mosaad. As well as to other collaborators with whom I enjoyed working on various other results during my graduate studies, namely Pavel Dvořák, Lubica Jančová, Tereza Klimošová, Michal Koucký, Josef Malík, Tomáš Masařík, Jana Novotná and Daniël Paulusma. Last but not least, I would like to thank my partner Karel Král and my family for cheering me up during the whole studies.

The research presented in this thesis was supported by the Grant Agency of the Czech Republic under the grant agreements no. 17-09142S and no. 19-27871X, by the Charles University projects PRIMUS/17/SCI/9, UNCE/SCI/004, GAUK 1568819, and by the Charles University grant SVV-2017-260452.

Title: On the Complexity of Search Problems with a Unique Solution

Author: Veronika Slívová

Institute: Computer Science Institute of Charles University

Supervisor: Mgr. Pavel Hubáček, Ph.D., Computer Science Institute of Charles University

Abstract: Meggido and Papadimitriou [Theor. Comput. Sci., 1991] introduced the class TFNP of search problems for which a solution always exists and is polynomially verifiable. In this thesis, we study the possibility of reducing different problems into problems in TFNP. The property which is in common for problems, for which we study the reducibility to TFNP, is that all instances of these problems have a unique solution (if there is any solution present).

In the first part of this thesis, we study a problem called ARRIVAL, which was introduced by Dohrau, Gärtner, Kohler, Matoušek and Welzl [A Journey Through Discrete Mathemathics: A Tribute to Jiří Matoušek, 2017]. ARRIVAL is the following decisional problem: Given a graph in which a train is moving according to prescribed rules does the train arrive to a given vertex? We first improve the result of Dohrau et al. who showed that the problem is in NP ∩ coNP. We show that there exists a unique certificate for being in the language and, thus, prove that it lies in UP ∩ coUP.

We also study the search version of the ARRIVAL problem, which asks for the transcript of number of traversals for each edge. It was known that the search version lies in PLS, which was proven by Karthik C. S. [Inf. Process. Lett., 2017]. We improve this result by showing a reduction from ARRIVAL to End-Of-Metered-Line (a problem introduced by Hubáček and Yogev [SIAM J. Comput., 2020]) and, thus, prove that it lies in the class CLS.

In the second half of this thesis, we study the possibility of showing hardness in TFNP based on cryptographic assumptions. We first rule out a fully black-box construction of a worst-case hard TFNP problem from a hard-on-average UP problem. Thus, we also rule out constructions of hard TFNP problems from hard-on-average problems in NP. Then, we consider more structured assumption of injective one-way functions (which imply a hard-on-average problem in UP). We show that, even in this case, it is not possible to construct a worst-case hard TFNP problem assuming that the reduction from injective one-way functions (OWF) is “simple”. More precisely, a security reduction is “simple” if it queries the TFNP instances non-adaptively and independently on the one-way function with respect to which it is running. Note that there are known “simple” constructions based on other cryptographic assumptions (such as collision resistant hash functions) and, thus, we believe that our restrictions on the security reduction are natural.

Keywords: Black-Box Separations, One-Way Functions, TFNP, CLS, ARRIVAL

Contents

Introduction

Megiddo and Papadimitriou [1991] introduced the class of total search problems called TFNP, that is problems for which we are guaranteed the existence of a solution and our task is to find it. Johnson et al. [1988], Megiddo and Papadimitriou [1991], Papadimitriou [1994] and Daskalakis and Papadimitriou [2011] introduced various subclasses of TFNP. The classification of these subclasses is based on the property used to argue their totality. This gives us a whole hierarchy of classes with various conjectured hardness. We focus on hardness of structured search problems which belong to TFNP and its subclasses. In Chapter 1 we define more formally the class TFNP and the subclasses that are most relevant to this thesis. We also summarize the known results which are tightly connected to our work.

The thesis has two main parts. Chapter 2 is dedicated to the natural reachability problem on switch graphs called `Arrival`, which was defined and first studied by Dohrau et al. [2017]. Switch graphs, also known under the name prop-machines as a special case of "deterministic" random walks, and their variants are well studied in combinatorics and automata theory. The problem `Arrival` can be described as: Given a graph G and a train which moves along the edges of G under some specific rules, does it ever reach one specified vertex called destination? Already since the paper of Dohrau et al. [2017] it was known that the problem is in $\mathsf{NP} \cap \mathsf{coNP}$. Later, Karthik C. S. [2017] proved that the search version of `Arrival` is contained in PLS. We improve both these results. First we prove that it is polynomially testable that a given vector is a *run-profile* (which describes per each edge the exact number of times it was traversed by the train on its route from origin to destination). Thus, proving containment of `Arrival` in $\mathsf{UP} \cap \mathsf{coUP}$ (a variant of $\mathsf{NP} \cap \mathsf{coNP}$ for which the certificate for being in the language is unique). Then we present a reduction from `Arrival` to `End-Of-Metered-Line`. Thus, showing that `Arrival` is contained also in CLS. This result makes it unlikely for `S-Arrival` to be PLS-hard, which was one of the possibilities suggested by the containment in PLS shown by Karthik C. S. [2017]. Since there are known black-box separations among subclasses of TFNP (see Morioka [2001] and Buresh-Oppenheim and Morioka [2004]), which suggest that CLS is a proper subclass of PLS. This part of the thesis is based on the paper Gärtner et al. [2018].

In the second half of the thesis, we focus on impossibility results. More precisely, in Chapter 3, we prove that there is no fully black-box construction of a worst-case hard TFNP problem from average-case hard problem in UP. Thus we also rule out fully black box-constructions from average-case hard problems in NP. Then, in Chapter 4, we consider constructions of hard problems in TFNP from injective one-way functions (which imply average-case hard problem in UP). Unfortunately, we are not able to rule out the existence of any fully black-box constructions in this case. But we achieve at least a partial result in this direction by showing that no "simple" fully black-box construction exists. On the other hand, most known constructions, e.g., the construction of `Pigeon` from one-way permutations by Papadimitriou [1994], satisfy our notion of "simple". Thus, our result can be viewed as an evidence that if there exists a construction it has to be quite non-standard. The impossibility of a construction from injective one-way functions (Chapter 4) was published in Hubáček et al. [2020] and the impossibility of a construction from average-case hard UP problem (Chapter 3) is based on unpublished joint work with Arka Rai Choudhuri, Pavel Hubáček, Chethan Kamath and Karel Král.

1. Definitions and Overview of Known Results

1.1 Complexity of Search Problems – TFNP

In computational complexity, we often study decision problems, i.e., we are given an instance of some problem and we want to decide whether it is a YES-instance or a NO-instance. Whereas, in practise, we often want not only to decide but to find the solution if it exists. This motivates the study of the complexity of search problems, i.e., problems where we want to return not only one bit (whether the solution exists or not) but rather the solution itself. There are many problems for which we do not know how to find a solution in polynomial time even though we are guaranteed that some solution exists. That is, the decision version of these problems is easy. These problems are captured by the TFNP class which was defined by Megiddo and Papadimitriou [1991].

TFNP stands for *Total Functions from NP* and it is a class of all *total* search problems, i.e., problems where all instances are syntactically guaranteed to have a solution. There are many intriguing problems with unresolved complexity which lie in TFNP and its subclasses, e.g., finding Nash equilibria, inverting a one-way permutation, integer factoring, finding fixed points, and many more.

Definition 1.1.1 (TFNP, Megiddo and Papadimitriou [1991])**.** *We say that a problem is in TFNP if the underlying relation R (that is for all $i, s \in \{0,1\}^*$, it holds that $(i, s) \in R$ if and only if s is a solution for instance i) satisfies:*

Verifiable in poly-time: *There exists a polynomial time deterministic algorithm C such that for all $i, s \in \{0,1\}^*$, it holds that $C(i, s) = 1$ if and only if $(i, s) \in R$, and*

Totality: *there exists a polynomial p such that for every $i \in \{0,1\}^*$, there exists $s \in \{0,1\}^*$ of length at most $p(|i|)$ such that $(i, s) \in R$.*

1.1.1 Overview of Subclasses of TFNP

The standard method for arguing computational hardness in TFNP is via clustering these problems into subclasses characterised by the existential argument guaranteeing their totality (see Papadimitriou [1994]). In the following paragraphs, we introduce subclasses of TFNP, where, for each subclass, we present one of its complete problems which helps to illustrate the kind of problems which belong to the class. The inclusions between subclasses are depicted in Figure 1.1. It is unlikely that there would be an inclusion which is not depicted in Figure 1.1, due to the known black-box separations between the classes (see Beame et al. [1998], Morioka [2001], Buresh-Oppenheim and Morioka [2004]). More precisely, the black-box separation $\mathsf{PLS} \not\subseteq \mathsf{PPP}$ has been shown by Buresh-Oppenheim and Morioka [2004] only for "nice" reductions. A "nice" reduction from a problem A to a problem B, as defined by Buresh-Oppenheim and Morioka [2004], is a Turing reduction, which makes at most one query to the oracle and, moreover, if the instance of A has a unique solution then so does the queried instance of B. Also no separation is known for UniqueEOPL and CLS, thus, it may hold that UniqueEOPL = CLS.

PPA: The class PPA is the class of problems for which the totality is given by the argument that any graph must have an even number of vertices of odd degree, from which the name of the class *Polynomial Parity Argument (PPA)* follows. This class was also defined by Papadimitriou [1994], who introduced the complete problem for this class called `Leaf`.

Definition 1.1.2 (`Leaf`, Papadimitriou [1994])**.** *Given a circuit N, which on an input $x \in \{0,1\}^n$ outputs a set of at most two strings from $\{0,1\}^n$, return:*

Type 1: *Either 0^n if $|N(0^n)| \neq 1$ or $N(0^n) = \{0^n\}$,*

Type 2: *or $x \in \{0,1\}^n \setminus \{0^n\}$ such that $|N(x) \setminus \{x\}| = 1$ or $\exists y \in N(x)$ such that $x \notin N(y)$.*

We interpret N as a circuit which gives the set of all potential neighbours of the input vertex. For any $x, y \in \{0,1\}^n$, the underlying graph contains an undirected edge $\{x, y\}$ if and only if $x \in N(y)$ and $y \in N(x)$. The solution of the first type ensures that 0^n is a leaf in the graph (we call it a trivial leaf as it should be always present). The solution of the second type corresponds to returning any nontrivial leaf or a vertex on which N does not describe an undirected graph (as y has a neighbour x but x is not a neighbour of y).

PPP: The class *PPP (Polynomial Pigeonhole Principle)* was defined by Papadimitriou [1994]. The totality of problems in PPP can be shown by applying a Pigeonhole principle, i.e., if there are n pigeons in at most $n-1$ holes, there must be a hole with at least two pigeons. Thus, the complete problem for this class, called `Pigeon`, is: Given a circuit which represents a length-preserving function, find either a preimage of zero (having no preimage of zero rules out one hole) or a collision (corresponding to a hole with two pigeons).

Definition 1.1.3 (`Pigeon`, Papadimitriou [1994])**.** *Given a circuit $P\colon \{0,1\}^n \to \{0,1\}^n$ return:*

Type 1: *Either $x \in \{0,1\}^n$ such that $P(x) = 0^n$,*

Type 2: *or $x, y \in \{0,1\}^n$ such that $P(x) = P(y)$.*

As an example of important problems known to lie in PPP we mention for instance inversion of one-way permutations for which reduction to `Pigeon` was proven by Papadimitriou [1994] or finding collisions of a function (folklore).

PPAD: The abbreviation *PPAD* stands for *Polynomial Parity Argument on Directed graphs* and the class was defined by Papadimitriou [1994]. The argument for totality is similar to PPA, but on directed graphs. Thus, instead of inspecting vertices of odd degree, we inspect unbalanced vertices, i.e., any vertex with indegree that differs from its outdegree. Again, it holds that if there is any unbalanced vertex, then the graph must contain at least two such vertices. A complete problem for this class is `Source-Or-Sink`, defined as follows:

Definition 1.1.4 (`Source-Or-Sink`, Papadimitriou [1994]). *Given two circuits P, S which on input $x \in \{0,1\}^n$ output a string of length n, return*

Type 1: *Either 0^n if $P(S(0^n)) \neq 0^n$ or $S(P(0^n)) = 0^n$,*

Type 2: *or $x \in \{0,1\}^n \setminus \{0^n\}$ such that $S(P(x)) \neq x$ or $P(S(x)) \neq x$.*

We can again interpret the circuits P, S as a directed graph on vertices $V = \{0,1\}^n$, where for any $u, v \in V$ there is a directed edge (u, v) if and only if $S(u) = v$ and $P(v) = u$. The absence of a solution of the first type guarantees that there is an oriented path starting at 0^n. The solutions of the second type give us either a source of a different directed path or a sink on some path in the graph.

As pointed out by Papadimitriou [1994], PPAD is a subset of PPA, since we can get an instance of `Leaf` by forgetting the orientation of the edges. That is, if N denotes the circuit from the definition of `Leaf` (see Definition 1.1.2) then we set $N(x) = \{P(x), S(x)\}$, for all $x \neq 0^n$, and $N(0^n) = \{S(0^n)\}$.

On the other hand, the orientation allows us to argue the totality using the pigeon-hole principle, too. Papadimitriou [1994] proved the inclusion PPAD $\subseteq$ PPP. Informally, an instance Q of `Pigeon`[1] can be created from an instance (P, S) of `Source-Or-Sink` (see Definitions 1.1.3 and 1.1.4) as follows: Assuming that the successor of every sink is a self-loop (i.e., $S(x) = x$ whenever $P(S(x)) \neq x$)[2], the circuit Q applies the successor twice and returns the result, i.e., $Q(x) = S(S(x))$. Thus, any sink x and its predecessor $P(x)$ introduce a collision under the circuit Q, since $Q(x) = x = Q(P(x))$. The class contains many important problems, for instance, finding Nash equilibria in bimatrix games (see Daskalakis et al. [2009], Chen et al. [2009]).

PLS: *PLS (Polynomial Local Search)* is a class capturing the hardness of searching for local optima. It was defined by Johnson et al. [1988], who describe any problem from the class PLS to be either minimization or maximization problem for which I is the set of instances recognizable in polynomial time, for each $i \in I$, we have a finite set S_i of all solutions, a cost function $\mathrm{cost}_i \colon S_i \to \mathbb{N}$ on the solutions, and, for any $s \in S_i$, we have sets $N_i(s)$ of "neighbouring" solutions to s (one can imagine solutions which can be retrieved by a small alternation of the solution s). Moreover, the following three algorithms must exists for the problem to be in PLS:

1. A_1 which given an instance $i \in I$ returns a generic solution $s \in S_i$,
2. A_2 which given an instance $i \in I$ and a potential solution s outputs whether s is a solution and the cost $\mathrm{cost}_i(s)$ of the solution and
3. A_3 which given an instance $i \in I$ and a solution $s \in S_i$ returns any solution $s' \in N(i, s)$ with better valuation (i.e., $\mathrm{cost}_i(s') < \mathrm{cost}_i(s)$ for minimization problem or $\mathrm{cost}_i(s') > \mathrm{cost}_i(s)$ for maximization problem) or returns that s is locally optimal.

[1]In Definition 1.1.3, Q corresponds to a circuit P, but as P is used for the predecessor circuit in the context of `Source-Or-Sink` we denote the `Pigeon` circuit by Q in this paragraph.

[2]This assumption is without loss of generality, as we can enforce it by a local modification of the successor circuit S.

Then the task of the PLS problem is to find a locally optimal solution, that is a solution $s \in S_i$ which has the minimal (for minimization problems), resp. the maximal (for maximization problems), valuation among its neighbours.

Thus, PLS contains all problems which are reducible to the following PLS-complete problem, called `LocalOpt`. Note that we may interpret the circuit V as the valuation and the circuit S provides us with a candidate for a better solution.

Definition 1.1.5 (`LocalOpt`, Johnson et al. [1988]). *Given two circuits S, V, where $S\colon \{0,1\}^n \to \{0,1\}^n$ and $V\colon \{0,1\}^n \to \{0,1\}^n$, return $x \in \{0,1\}^n$ such that $V(S(x)) \leq V(x)$, where we interpret the output of V as an integer.*

CLS: The class *CLS (Continuous Local Search)* was defined by Daskalakis and Papadimitriou [2011] to capture problems in local optimization over continuous domains. The class contains any total search problem which is reducible to the following problem called `Continuous-LocalOpt` (`CLOpt`).

Definition 1.1.6 (`CLOpt`, Daskalakis and Papadimitriou [2011]). *Given an arithmetic circuit $f\colon [0,1]^3 \to [0,1]^3$ and an arithmetic circuit $p\colon [0,1]^3 \to [0,1]$, two real constants $\varepsilon, \lambda \in (0,1)$, and unary representation of a natural number K, find either a point $x \in [0,1]^3$ such that $p(f(x)) \leq p(x) + \varepsilon$ or a pair of points $x, x' \in [0,1]^3$ certifying that either p or f is not λ-Lipschitz or a point $x \in [0,1]^3$ for which there exists a gate such that its output has size bigger than $(size(x)K)^3$.*

It is not easy to prove that a combinatorial problem is in CLS by showing a reduction directly to `CLOpt`. For such reductions, another problem called `End-Of-Metered-Line` (`EOML`) might be better suited. The problem was defined by Hubáček and Yogev [2020] who also showed that `EOML` reduces to `CLOpt` and, thus, lies in the class CLS. Although `End-Of-Metered-Line` seems like a good candidate for a CLS-complete problem and its containment in CLS allowed proving hardness of CLS from cryptographic assumptions (see Hubáček and Yogev [2020] and Section 1.3.1 where overview of hardness results is provided), it is currently not known whether it is CLS-hard or not.

Definition 1.1.7 (`End-Of-Metered-Line`, Hubáček and Yogev [2020]). *Given circuits $S, P\colon \{0,1\}^m \to \{0,1\}^m$, and $V\colon \{0,1\}^m \to [2^m] \cup \{0\}$ such that $P(0^m) = 0^m \neq S(0^m)$ and $V(0^m) = 1$, find a string $x \in \{0,1\}^m$ satisfying one of the following:*

Type 1: *Either $P(S(x)) \neq x$ or $S(P(x)) \neq x \neq 0^m$,*

Type 2: *$x \neq 0^m$ and $V(x) = 1$,*

Type 3: *either $V(x) > 0$ and $V(S(x)) - V(x) \neq 1$ or $V(x) > 1$ and $V(x) - V(P(x)) \neq 1$.*

[3]This type of solution was recently added by Daskalakis and Papadimitriou in corrigendum, see `http://people.csail.mit.edu/costis/CLS-corrigendum.pdf`, due to a technical issue with the definition. A subtle issue in the original definition (without the parameter K and the additional solution) is that the arithmetic circuits p, f could use repeated squaring to compute numbers doubly exponential in the size of their inputs, which would prevent p, f from being evaluated in polynomial time. The same issue was pointed out by Fearnley et al. [2021].

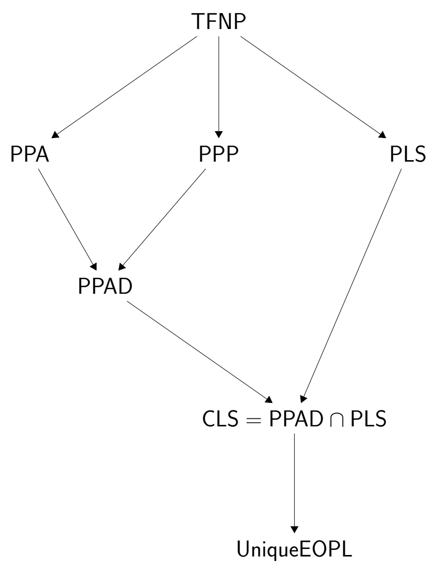

Figure 1.1: The structure of TFNP and its subclasses. A class A is included in a class B if there is an arrow from B to A.

The circuits S and P define a directed graph analogously as in the problem `Source-Or-Sink`. The circuit V is a valuation of the vertices and it should correspond to an "odometer" which counts how far on the path starting at 0^n is the vertex located. Then solutions of the first type correspond to the end point of the directed path, alternatively a starting point of a different path. Existence of a solution of the second or third type shows that the circuit V is not a proper "odometer".

To mention the connection to other classes, Daskalakis and Papadimitriou [2011] proved that CLS $\subseteq$ PPAD $\cap$ PLS. This was recently improved by Fearnley et al. [2021] who showed that there is an equality, i.e., CLS $=$ PPAD $\cap$ PLS.

UniqueEOPL: Recently, Fearnley et al. [2020] studied the problems in CLS which are known to have unique solutions. They defined the problem `UniqueEOPL` *(Unique End Of Potential Line)* using its promise variant `Promise-UniqueEOPL`.

The `Promise-UniqueEOPL` is as follows: Given a graph which is promised to consist of a unique line starting at 0^n and self loops on vertices outside of the line, find the sink of the unique line. The graph is given by circuits describing the predecessor of a vertex, the successor of a vertex and a so-called valuation which says how far on the described line from the trivial source the vertex lies.

To capture the uniqueness without the promise, in `UniqueEOPL` we are allowed to return also certificates for violation of this promise, i.e., two vertices which are provably on different lines (e.g., by giving a non-trivial source). Fearnley et al. [2020] defined the class UniqueEOPL as all problems which can be reduced to the problem of the same name. This class contains many important problems such as `Unique Sink Orientation` (`USO`) or solving `PLCP` (the `Linear Complementarity Problem` for P-matrices), see Fearnley et al. [2020]. Another problem from this class is `Arrival` which we describe in more detail in Section 1.2.

1.2 Arrival

The problem `Arrival` was introduced by Dohrau et al. [2017]. It is a natural computational problem on switch graphs, which Dohrau et al. [2017] informally described as follows:

> *Suppose that a train is running along a railway network, starting from a designated origin, with the goal of reaching a designated destination. The network, however, is of a special nature: every time the train traverses a switch, the switch will change its position immediately afterwards. Hence, the next time the train traverses the same switch, the other direction will be taken, so that directions alternate with each traversal of the switch.*
>
> *Given a network with origin and destination, what is the complexity of deciding whether the train, starting at the origin, will eventually reach the destination?* Dohrau et al. [2017]

1.2.1 Containment in Complexity Classes

Already in Dohrau et al. [2017], it was shown that the problem `Arrival` lies in NP ∩ coNP. To determine whether the train eventually reaches its destination, it is natural to consider a *run profile*, i.e., the complete transcript describing how many times the train traversed each edge. Dohrau et al. [2017] presented a natural integer programming interpretation of run profiles called *switching flows*, which have the advantage of being trivial to verify. The downside of switching flows is that they do not guarantee to faithfully represent the number of times each edge has been traversed; a switching flow might contain superfluous circulations compared to a valid run profile. Nevertheless, Dohrau et al. [2017] proved that the existence of a switching flow implies that the train reaches its destination, and, thus, a switching flow constitutes an NP witness for `Arrival`.

The coNP membership was shown by an insightful observation about the structure of switch graphs. Specifically, the train reaches its destination d if and only if it never enters a node from which there is no directed path to d. The railway network can thus be altered so that all such vertices point to an additional "dead-end" vertex $\overline{d}$. The coNP witness is then simply a switching flow from the origin to the dead-end $\overline{d}$.

Given that the decision variant of `Arrival` is in NP ∩ coNP, it is natural to study the search complexity of `Arrival` in the context of total search problems with the guaranteed existence of a solution, i.e., within the complexity class TFNP (which contains the search analogue of NP ∩ coNP). Karthik C. S. [2017] noticed that the search for a switching flow is a prime candidate to fit into the hierarchy of TFNP problems. He introduced `S-Arrival`, a search version of `Arrival` that seeks a switching flow to either the destination d or the dead-end vertex $\overline{d}$, and showed that it is contained in the complexity class PLS of total problems amenable to local search.

By reducing `S-Arrival` to `End-Of-Metered-Line` Gärtner et al. [2018] showed that `S-Arrival` lies in the class CLS. Gärtner et al. [2018] also proved that it is possible to verify in polynomial time whether the given switching flow is a run profile or not. Thus `Arrival` is in $\mathsf{UP} \cap \mathsf{coUP}$.[4]

Finally, Fearnley et al. [2020] pointed out that the `End-Of-Metered-Line` instance contains exactly one line and, thus, the proof actually shows the containment in the UniqueEOPL class.

1.2.2 Hardness Results

Fearnley et al. [2017] studied multiple variants of reachability games on switch graphs and, as one of their results, gave a lower bound on the complexity of deciding `Arrival`. Specifically, they showed that `Arrival` is NL-hard. That is any problem solvable by a non-deterministic Turing machine in logarithmic space can be reduced to `Arrival`, where the reduction uses only logarithmic amount of memory.

Fearnley et al. [2017] introduced a natural 1-player and 2-player variants of `Arrival`. In the one player variant, the vertices are split into two sets V_0, V_1. The player controls the movement of the train in the vertices V_1 whereas the movement through the vertices from the set V_0 is specified by the switching behaviour. The player wins if the train arrives in its destination. Fearnley et al. [2017] showed that, in this case, it is NP-complete to decide whether train arrives to its destination.

Similarly, in the two player variant, the vertices are split into three sets V_0, V_1 and V_2. The first player controls movement of the train in vertices from V_1 and wins if the train arrives to its destination. The other player controls vertices in V_2 and wins if the train never arrives. The movement through V_0 is still given by the switching behaviour. As Fearnley et al. [2017] showed, deciding this variant is PSPACE-hard.

1.2.3 Overview of the Algorithms

The first algorithm for `Arrival` was provided by Dohrau et al. [2017], who showed that the train run can be simulated in time $\mathcal{O}(n2^n)$. They showed that if the train visits any vertex more than 2^n times, it has to cycle and, thus, it never reaches the destination. On the other hand, they proved existence of switching graphs on which the simulation takes $\Omega(2^n)$ time.

The reduction of `S-Arrival` to `End-Of-Metered-Line` by Gärtner et al. [2018] allowed for a better algorithm for `S-Arrival`. Namely, the algorithm of Aldous [1983] can be used to find a solution for `S-Arrival`. This gave a randomized algorithm running in time $\mathcal{O}\left(\text{poly}(n)2^{n/2}\right)$ (see Gärtner et al. [2018])[5].

Currently the best known algorithm for `Arrival` was shown by Gärtner et al. [2021] who proved that `Arrival` can be solved in $2^{\mathcal{O}\left(\sqrt{n}\log(n)\right)}$. Moreover, their technique shows that `Arrival` can be solved in polynomial time if the graph has a feedback vertex set of constant size (i.e., if there are constantly many vertices such that after their removal the graph contains no directed cycle).

[4]These results are covered by this thesis, but we list them in this overview to draw a full picture of what is known.

[5]Note that, even though it is a part of Gärtner et al. [2018], this result is not covered by this thesis, because I was not involved in proving this particular result.

1.3 Constructing a Hard Problem in TFNP

The possibility of existence of polynomial time algorithms for all of TFNP is still open. Especially, disproving existence of polynomial algorithm would imply FP $\neq$ FNP[6] and, thus, proving an inequality between P and NP. On the other hand, all problems in TFNP might be solvable in polynomial time, even when P $\neq$ NP.

One way to give an indication that TFNP is not easy would be proving that it contains an FNP-hard problem. This would imply that TFNP is worst-case hard whenever FNP is worst-case hard. Unfortunately, it was shown already by Megiddo and Papadimitriou [1991] that the inclusion of an FNP-hard problem in TFNP would imply that NP = coNP as the totality of TFNP would provide us with a certificate for NO-instances. Note that this would be a breakthrough result since the relationship between NP and coNP as well as the question whether the polynomial hierarchy collapses are long-standing open problems in complexity theory. Thus, we need different assumptions to be able to indicate the hardness of problems in TFNP – for example, by showing that their hardness follows from some cryptographic assumptions.

To be able to present the hardness results, let us first explain two different notions of hardness, called *worst-case* and *average-case* hardness. Worst-case hardness states that there exist hard instances. To give an example of worst-case hardness one can imagine the standard reductions used for definition of NP which preserve worst-case hardness.

Intuitively, for any worst-case hard problem, there could be only few hard instances per each instance length, but it might be hard to generate any hard-instance. In fact, finding a hard instance might be even harder than solving it. The stronger notion of average-case hardness, introduced by Levin [1986], overcomes this issue by making sure that hard instances are easy to find. More specifically, a problem is average-case hard if there exists a polynomial-time algorithm which samples the hard instances. To make a reduction preserving average-case hardness it is not sufficient to transform instances of one problem A into instances of problem B but we also have to be able to provide a polynomial-time sampler of hard instances of B.

Many constructions of average-case hardness in TFNP and its subclasses have been shown based on various cryptographic primitives and we summarize these results in Section 1.3.1. Existence of these results raises the following question:

What is the weakest or least structured assumption under which we can prove average-case or worst-case hardness in TFNP?

We give an overview of existing negative results in Section 1.3.2.

1.3.1 Known Constructions

Hardness from standard cryptographic primitives or number theoretic assumptions was long known for the "higher" classes in TFNP like PPP and PPA. We

[6]FP is the class of search problems solvable in polynomial time, i.e., a search variant of P, and FNP is the class of search problems solvable in non-deterministic polynomial time, i.e., a search variant of NP.

have already mentioned the result of Papadimitriou [1994] that one-way permutations (OWP) imply average-case hardness in PPP. It is a folklore knowledge that existence of collision-resistant hashing (e.g., hardness of integer factoring or the discrete logarithm problem in prime-order groups) implies average-case hardness in PPP. In addition, Jeřábek [2016], building on the work of Buresh-Oppenheim [2006], showed that PPA as well as PPP is no easier than integer factoring.

However, it is only recently that we are better understanding the cryptographic hardness of the lower classes in TFNP. This was catalysed by the result of Bitansky et al. [2015], who, expanding on Abbot et al. [2004], devised an average-case hard PPAD instance created from quasi-polynomially hard indistinguishability obfuscation (iO) for P/poly and sub-exponentially hard injective one-way functions. This result was improved by Garg et al. [2016], who relaxed the security assumptions on the primitives and provided a construction from iO and OWP, which are hard only against polynomial algorithms and Komargodski and Segev [2017], who provided a construction of average-case hard PPAD instance from quasipolynomially secure private key functional encryption scheme for circuits of polynomial size and subexponentially secure injective one-way functions.

Hubáček and Yogev [2020] introduced the End-Of-Metered-Line problem, which, as they proved, lies in the class CLS. They showed that there exists an average-case hard distribution of End-Of-Metered-Line based on the existence of OWP and indistinguishability obfuscation for P/poly. Thus, they extended the results of Bitansky et al. [2015], Garg et al. [2016] all the way down to the class CLS.

The underlying assumptions were relaxed further to cryptographic assumptions that are more plausible than indistinguishability obfuscation like soundness of Fiat-Shamir transformation for sumcheck protocol and existence of worst-case hard problem in #P by Choudhuri et al. [2019a], hardness of iterated squaring modulo composite number by Choudhuri et al. [2019b] and a construction from continuous Verifiable Delay functions by Ephraim et al. [2020].

Using similar ideas, Bitansky and Gerichter [2020] presented an unconditional construction of hard-on-average distributions in the complexity class PLS in the random oracle model. They also proved worst-case hardness in PLS based on randomized Exponential Time Hypothesis and an assumption on bilinear maps by Kalai et al. [2019]. Building on these results, a flurry of recent works (e.g., Kalai et al. [2020], Kalai and Zhang [2020], Jawale and Khurana [2020], Jawale et al. [2020], Lombardi and Vaikuntanathan [2020]) further weakened the assumptions required for proving average-case hardness in CLS. More concretely, they constructed average-case hardness in CLS assuming (sub-exponential) hardness of learning with errors. Thus, bringing us closer to proving average-case hardness in CLS under a standard *concrete* cryptographic assumption.

Constructions from average-case hardness in NP: Perhaps the most relevant constructions with respect to results presented in this thesis are the ones of Hubáček et al. [2017] and Pass and Venkitasubramaniam [2019]. Hubáček et al. [2017] showed that average-case hardness in NP (which is implied by existence of one-way functions) implies average-case hardness in TFNP under complexity theoretic derandomization assumption. More specifically their construction of average-case hard TFNP assumes that there exists a function computable deterministically in time $2^{O(n)}$, but having non-deterministic circuit complexity at least $2^{\Omega(n)}$. Alternatively, they showed a construction of a hard-on-average problem in TFNP based on injective one-way functions and non-interactive witness indistinguishable proof systems for NP. Pass and Venkitasubramaniam [2019] complemented the Hubáček et al. [2017] result by showing that when one-way functions *do not exist*, average-case hardness in NP implies average-case hardness in TFNP.

1.3.2 Impossibility Results

On the opposite site, there are results giving evidence that average case hard problems cannot be based on certain types of assumptions. As already mentioned, Megiddo and Papadimitriou [1991] proved that worst-case hard TFNP problems cannot be based on worst-case hard FNP problems unless the polynomial hierarchy collapses since such a result would imply that NP = coNP (and analogous result but for constructing worst-case hard PLS problem is known already since Johnson et al. [1988]). Mahmoody and Xiao [2010] proved that a randomized reduction from SAT to any TFNP problem would imply that SAT is polynomially checkable.[7] These results were expanded by Buhrman et al. [2010] who showed that there exists an oracle under which TFNP is easy but polynomial hierarchy does not collapse. All these results give evidence for impossibility of constructions of worst-case hard problems in TFNP from a worst-case hard problem in NP or polynomial hierarchy.

The impossibility of constructing TFNP hardness from one-way functions was studied by Rosen et al. [2017]. They gave a partial answer by showing that there do not exist hard-on-average distributions of TFNP instances over $\{0,1\}^n$ with fewer than $2^{n^{o(1)}}$ solutions from any primitive which exists relative to a random injective trapdoor function oracle (in particular, from one-way functions or collision resistant hash functions). Their main observation was that the argument in Rudich [1988], which separates one-way functions from one-way permutations, can be strengthened to separate other unstructured primitives from structured primitives (such as problems in TFNP).

However, it seems that the Rudich [1988] argument has been exploited to its limits in Rosen et al. [2017], and, therefore, it is not clear whether their approach can be extended to fully separate one-way functions and TFNP. Thus, the situation is contrasting to NP ∩ coNP, the decision counterpart of TFNP, whose relationship with (injective) OWFs is much better studied. In particular, we know that hardness is implied by one-way permutations due to the existence of hard core predicates (see Goldreich and Levin [1989]). But Bitansky et al. [2021] showed that injective OWFs, even with indistinguishability obfuscation, (and, therefore, public-key encryption) cannot imply (worst-case) hardness in NP ∩ coNP in a black-box way.

[7] A problem P is polynomially checkable (as defined by Blum and Kannan [1995]) if there is a randomized polynomial algorithm A which given oracle access to any algorithm B and input $x \in \{0,1\}^*$: If B solves P correctly (i.e., $\forall x' \colon B(x') = P(x')$) then $A^B(x)$ returns "CORRECT" with high probability. Otherwise, when running on input x such that $B(x) \neq P(x)$, the algorithm $A^B(x)$ returns "INCORRECT" with high probability. Existence of such an algorithm for SAT is an open question for over 25 years (see Blum and Kannan [1995]).

Recently, Hubáček et al. [2020][8] attempted to extend the result of Bitansky et al. [2021] to TFNP and showed that there is no *simple* fully black-box construction of a worst-case hard TFNP problem from injective one-way functions. A formal definition of a (simple) fully black-box construction and the result of Hubáček et al. [2020] can be found in Chapter 4.

[8]This results is part of this thesis (see Chapter 4), but for completeness of this overview we also list the paper here.

2. `Arrival` Is Contained in CLS

One of the open problems suggested by Dohrau et al. [2017] was whether deciding the termination of `Arrival` is contained in UP∩coUP (recall that UP is a subclass of NP such that for each YES-instance there is a unique certificate). Recall that given a railway network with an origin o and a destination d, the transcript of the route of the train captured in the *run profile* from o to d (if it exists) is unique. In this chapter we show that it is possible to efficiently decide whether a *switching flow* (the certificate for being in NP defined by Dohrau et al. [2017]) corresponds to a run profile, which provides a positive answer to the above question and places `Arrival` inside UP ∩ coUP.

Additionally, we show that `S-Arrival` (the search version of `Arrival`) is contained in the complexity class CLS. We establish the containment in CLS through a reduction to `EOML` (see Definition 1.1.7). Recall that in `EOML` we are given a source in a directed graph with vertices of in-degree and out-degree at most one, and the task is to find a sink or a source different from the given trivial source. The access to the graph is given locally via information about the successor and predecessor of each vertex together with its distance from the trivial source. In our construction, the graph consists of a single path on vertices corresponding to partial run profiles (transcript of the route of the train after doing exactly k-steps from origin o for any $k \in \mathbb{N}$) where the edges are between two partial run profiles which differ exactly by one move of the train and many isolated vertices – vertices which do not correspond to partial run profiles.

We give an idea of our proofs in Section 2.1 and give the formal definitions of the most frequently used terms in Section 2.2. In Section 2.3, we prove that we can verify the run profiles in polynomial time. Finally, in Section 2.4, we put these results together and show the containment in UP ∩ coUP as well as in CLS.

2.1 Our Techniques

Recall that a switching flow is a run profile with additional superfluous circulations compared to the valid run profile. Our main technical observation is a characterization of switching flows that correspond to the valid run profile. Given a switch graph G and a switching flow f, we consider the subgraph G^* induced over the railway network by the "last-used" edges; for every vertex v, we include in G^* only the outgoing edge that was, according to the switching flow, used by the train last time it left from v. Note that such last-used edges can be efficiently identified simply by considering the parity of the total number of visits at every vertex. When f is a valid run profile, then it is straightforward to see that the subgraph G^* is acyclic. We show that this property is in fact a characterization, i.e., any switching flow for which the induced graph G^* is acyclic must be a run profile. Given that this property is easy to check, we can use it to efficiently verify run profiles as UP witnesses. (The coUP witness is then a run profile to the dead-end at $\bar{d}$.)

For our reduction from `S-Arrival` to `End-Of-Metered-Line`, we extend the above observation to partial switching flows that are not required to end at the destination. The vertices of the `EOML` graph created by our reduction correspond to partial switching flows in the `S-Arrival` instance. The directed edges connect partial run profiles to their natural successors and predecessors, i.e., the partial run extended or shortened by a single step of the train. Any switching flow that does not correspond to some partial run profile is an isolated vertex in the `EOML` graph. Finally, the trivial source is the empty switching flow, and the distance from it can be computed for any partial run simply as the number of steps taken by the train so far. Given that there is only a single path in the resulting `EOML` graph and that its sink is exactly the complete run, we get that the unique solution to the `EOML` instance gives us a solution for the original instance of `S-Arrival`.

To make the reduction efficiently computable, we need to address the verification of partial run profiles. As it turns out, partial run profiles can be efficiently verified using the graph G^*, in a similar way to complete run profiles discussed above. The main difference is that the graph of last-used edges for a partial run profile can contain a cycle, as the train might visit the same vertex multiple times on its route to the destination. However, we show that there is at most one cycle in G^*, which always contains the current end-vertex of the partial run. The complete characterization of partial run profiles (which covers also full run profiles) is given in Lemma 2.3.3, and the formal reduction is described in Section 2.4.2.

2.2 Preliminaries

In the rest of this chapter we use the following standard notation. For $k \in \mathbb{N}$, we denote by $[k]$ the set $\{1, \ldots, k\}$. For a graph $G = (V, E)$, we reserve $n = |V|$ for the number of vertices. The basic object that we study are switch graphs, as defined by Dohrau et al. [2017]. To emphasize that a variable corresponds to a vector we use bold letters (such as $\boldsymbol{r}, \boldsymbol{f}, \boldsymbol{\delta}$). We usually denote the n-th entry of a vector $\boldsymbol{x}$ by $\boldsymbol{x}_n$ but for better readability we sometimes use $\boldsymbol{x}(n)$ instead of $\boldsymbol{x}_n$ to avoid recursive subscripts.

Definition 2.2.1 (switch graph)**.** *A* switch graph *is a tuple* $G = (V, E, s_0, s_1)$ *where* $s_0, s_1 \colon V \to V$ *and* $E = \{(v, s_0(v)), (v, s_1(v)) \mid \forall v \in V\}$.[1]

In order to avoid cumbersome notation, we slightly overload the use of s_0, s_1 *and treat both as functions from vertices to edges; that is by* $s_b(v)$ *we denote the edge* $(v, s_b(v))$ *for* $b \in \{0, 1\}$*. We use this convention throughout the paper unless stated otherwise.*

The `Arrival` problem was formally defined by Dohrau et al. [2017] as follows.

Definition 2.2.2 (`Arrival`, Dohrau et al. [2017])**.** *Given a switch graph* $G = (V, E, s_0, s_1)$ *and two vertices* $o, d \in V$*, the* ***`Arrival`*** *problem is to decide whether the algorithm* Run *(Algorithm 1) terminates, i.e., whether the train reaches the destination* d *starting from the origin* o*.*

To simplify theorem statements and our proofs, we assume without loss of generality that both $s_0(d)$ and $s_1(d)$ end in d.

A natural witness for termination of the Run procedure considered in previous work (e.g., Dohrau et al. [2017]) is a switching flow. We extend the definition of a switching flow to allow for partial switching flows that do not necessarily end in the desired destination d.

[1]Whenever $s_0(v) = s_1(v)$ for some vertex $v \in V$ we depict them as multiple edges in figures.

Algorithm 1: Run

```
Input  : a switch graph $G = (V, E, s_0, s_1)$ and two vertices $o, d \in V$
Output: a vector $\boldsymbol{r} \in \mathbb{N}^{|E|}$ where $\boldsymbol{r}_e$ for each edge $e \in E$, corresponds to the
        number of times the train traversed $e$

1  $v \leftarrow o$                                           // position of the train
2  $\forall u \in V$ set $s_{\text{curr}}(u) \leftarrow s_0(u)$ and $s_{\text{next}}(u) \leftarrow s_1(u)$
3  $\forall e \in E$ set $\boldsymbol{r}_e \leftarrow 0$                      // initialize the run profile
4  $step \leftarrow 0$
5  while $v \neq d$ do
6      $(v, w) \leftarrow s_{\text{curr}}(v)$                        // compute the next vertex
7      $\boldsymbol{r}_{(v,w)} \leftarrow \boldsymbol{r}_{(v,w)} + 1$                     // update the run profile
8      swap$(s_{\text{curr}}(v), s_{\text{next}}(v))$
9      $v \leftarrow w$                                        // move the train
10     $step \leftarrow step + 1$
11 end
12 return $\boldsymbol{r}$
```

Definition 2.2.3 ((partial) switching flow, end-vertex)**.** *Let* $G = (V, E, s_0, s_1)$ *be a switch graph. For* $o, d \in V$*, we say that* $\boldsymbol{f} \in \mathbb{N}^{2n}$ *is a* switching flow from o to d *if the following two conditions hold.*

Kirchhoff's Law (flow conservation):

$$\forall v \in V\colon \sum_{e=(u,v)\in E} \boldsymbol{f}_e - \sum_{e=(v,w)\in E} \boldsymbol{f}_e = [v = d] - [v = o] \ ,$$

where $[\cdot]$ *is the indicator variable of the event in brackets.*

Parity Condition:

$$\forall v \in V\colon \boldsymbol{f}_{s_1(v)} \leq \boldsymbol{f}_{s_0(v)} \leq \boldsymbol{f}_{s_1(v)} + 1 \ .$$

Kirchoff's law means that o emits one unit of flow, d absorbs one unit of flow, and at all other vertices, in-flow equals out-flow. If $d = o$, we have a circulation.

Given an instance $(G = (V, E, s_0, s_1), o, d)$ of ***Arrival****, we say that* $\boldsymbol{f}$ *is a* switching flow *if it is a switching flow from o to d. A vector* $\boldsymbol{f} \in \mathbb{N}^{2n}$ *is called a* partial switching flow *iff* $\boldsymbol{f}$ *is a switching flow from o to v for some vertex $v \in V$. We say that v is the* end-vertex *of the partial switching flow. We denote the end-vertex of* $\boldsymbol{f}$ *by* $v_{\boldsymbol{f}}$*.*

Definition 2.2.4 ((partial) run profile)**.** *A* run profile *is the switching flow* $\boldsymbol{r}$ *returned by the algorithm* Run *(Algorithm 1) upon termination. A* partial run profile *is a partial switching flow corresponding to some intermediate value of* $\boldsymbol{r}$ *in the algorithm* Run *(Algorithm 1).*

Observation 2.2.5 ([Dohrau et al., 2017, Observation 1])**.** *Each (partial) run profile is a (partial) switching flow.*

Observation 2.2.6. *An end-vertex* $v_{\boldsymbol{f}}$ *of a switching flow* $\boldsymbol{f}$ *is computable in polynomial time.*

Proof. It is sufficient to determine which vertex has a net in-flow of one. □

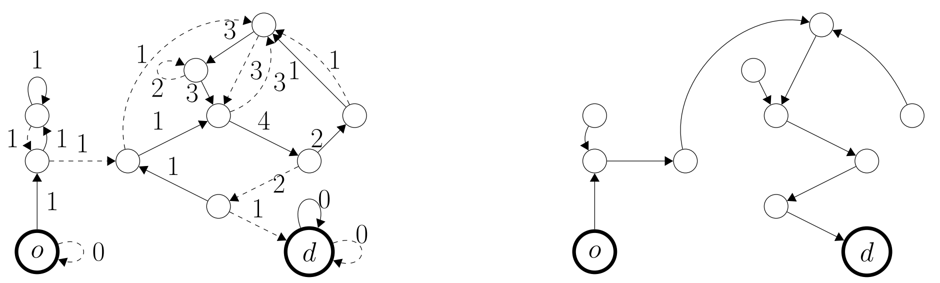

Figure 2.1: An example of a switch graph G with a switching flow $\boldsymbol{f}$ (on the left) and the corresponding graph $G^*_{\boldsymbol{f}}$ (on the right). The s_0 edges are denoted by full arrows and the s_1 edges are denoted by dashed arrows. For each edge in G, the switching flow $\boldsymbol{f}$ is specified by the adjacent integer value.

2.3 The Complexity of Run Profile Verification

Dohrau et al. [2017] proved that it is possible to efficiently verify whether a given vector is a switching flow. In this section we show that we can also efficiently verify whether a switching flow is a run profile. Combining this with the results by Dohrau et al. [2017], we prove that the decision problem of `Arrival` is in $\mathsf{UP} \cap \mathsf{coUP}$ (see Section 2.4.1) and that the search problem of `Arrival` lies in the complexity class CLS (see Section 2.4.2). As outlined in Section 2.1, our approach for verification of run profiles is based on finding a cycle in a natural subgraph of the railway network G defined below. Specifically, we consider the subgraph of G that contains only the last visited outgoing edge of each vertex, i.e., every vertex has out-degree at most one (see Figure 2.1 for an illustration).

Definition 2.3.1 ($G^*_{\boldsymbol{f}}$)**.** *Let $(G = (V, E, s_0, s_1), o, d)$ be an instance of* `Arrival`*, and let $\boldsymbol{f} \in \mathbb{N}^{2n}$ be a partial switching flow. We define a graph $G^*_{\boldsymbol{f}} = (V, E^*)$ as follows*

$$E^* = \left\{ s_0(v) \colon \forall v \in V \ \ s.t. \ \ \boldsymbol{f}_{s_0(v)} \neq \boldsymbol{f}_{s_1(v)} \right\} \cup \\ \left\{ s_1(v) \colon \forall v \in V \ \ s.t. \ \ \boldsymbol{f}_{s_0(v)} = \boldsymbol{f}_{s_1(v)} > 0 \right\}.$$

Observation 2.3.2. *Given a partial switching flow $\boldsymbol{f}$, the graph $G^*_{\boldsymbol{f}}$ can be computed in polynomial time.*

Lemma 2.3.3. *A partial switching flow $\boldsymbol{f}$ is a partial run profile iff $\boldsymbol{f}_{s_0(d)} = \boldsymbol{f}_{s_1(d)} = 0$ and one of the following two conditions holds:*

1. *There exists no cycle in $G^*_{\boldsymbol{f}}$.*
2. *There exists exactly one cycle in $G^*_{\boldsymbol{f}}$ and this cycle contains the end-vertex of $\boldsymbol{f}$.*

The main idea of the proof is based on the following fact: a switching flow $\boldsymbol{f}$ which is *not* a run profile must contain a circulation (as shown by Dohrau et al. [2017]). Let $\boldsymbol{f}$ be a switching flow that we get from a run profile $\boldsymbol{r}$ by adding some flows on cycles, then the last added circulation (the last added cycle) must form a cycle in the corresponding graph $G^*_{\boldsymbol{f}}$. On the other hand, a cycle containing the end-vertex is formed in $G^*_{\boldsymbol{f}}$ whenever the train arrives to a previously visited

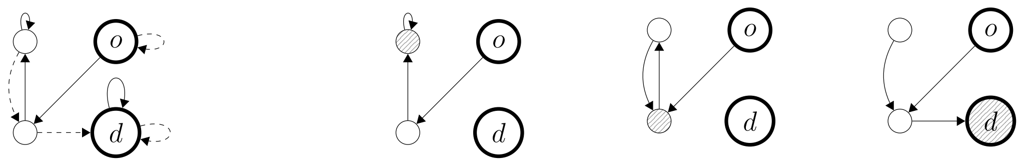

Figure 2.2: An example of a switch graph G and cycles in the graphs G^* corresponding to partial run profiles after 3, 4, and 5 steps of the train (respectively from left to right). We use hatching to highlight the current end-vertex.

vertex. An illustration of the graph G^* at consecutive steps of the algorithm Run, with the corresponding evolution of the end-vertices and cycles, is given in Figure 2.2.

We prove the "$\Leftarrow$" implication of Lemma 2.3.3 by contradiction. Given a switching flow $\boldsymbol{f}$ we consider the longest run profile $\boldsymbol{r}$ that is everywhere at most $\boldsymbol{f}$. We denote the difference $\boldsymbol{f}-\boldsymbol{r}$ by $\boldsymbol{\delta}$. We utilize the following observation at two points in our proof. First, to prove that both $\boldsymbol{f}$ and $\boldsymbol{r}$ have the same end-vertex, that is $v_{\boldsymbol{r}} = v_{\boldsymbol{f}}$. Second, to prove that any cycle we find in the intersection of $G^*_{\boldsymbol{f}}$ and the non-zero edges of $\boldsymbol{\delta}$ avoids $v_{\boldsymbol{f}}$.

Observation 2.3.4. *Let $\boldsymbol{f}$ be a switching flow and let $\boldsymbol{r}$ be the longest partial run profile such that all coordinates are at most $\boldsymbol{f}$ and denote $\boldsymbol{\delta} = \boldsymbol{f} - \boldsymbol{r}$. Then $\boldsymbol{\delta}_{s_0(v_{\boldsymbol{r}})} = \boldsymbol{\delta}_{s_1(v_{\boldsymbol{r}})} = 0$.*

Proof of Observation 2.3.4. Suppose that either $\boldsymbol{\delta}_{s_0(v_{\boldsymbol{r}})} \neq 0$ or $\boldsymbol{\delta}_{s_1(v_{\boldsymbol{r}})} \neq 0$, and that $\boldsymbol{r}$ is such that the next edge for the train to continue on is $s_0(v_{\boldsymbol{r}})$. From the maximality of $\boldsymbol{r}$ we get that $\boldsymbol{\delta}_{s_0(v_{\boldsymbol{r}})}$ is equal to zero. Then we get

$$\begin{aligned}
\boldsymbol{f}_{s_1(v_{\boldsymbol{r}})} &> \boldsymbol{r}_{s_1(v_{\boldsymbol{r}})} && \text{(since } \boldsymbol{\delta}_{s_1(v_{\boldsymbol{r}})} \text{ is non-zero)}\\
&= \boldsymbol{r}_{s_0(v_{\boldsymbol{r}})} && \text{(from the parity condition for } \boldsymbol{r}\text{)}\\
&= \boldsymbol{f}_{s_0(v_{\boldsymbol{r}})} && \text{(from the maximality of } \boldsymbol{r}\text{)}
\end{aligned}$$

This leads to a contradiction with the parity condition of Definition 2.2.3 as $\boldsymbol{f}_{s_1(v_{\boldsymbol{r}})} > \boldsymbol{f}_{s_0(v_{\boldsymbol{r}})}$.

The other case is similar: suppose that the train should continue with the edge $s_1(v_{\boldsymbol{r}})$. From the maximality of $\boldsymbol{r}$ we get that $\boldsymbol{\delta}_{s_1(v_{\boldsymbol{r}})}$ is equal to zero. And thus we get

$$\begin{aligned}
\boldsymbol{f}_{s_0(v_{\boldsymbol{r}})} &> \boldsymbol{r}_{s_0(v_{\boldsymbol{r}})} && \text{(since } \boldsymbol{\delta}_{s_0(v_{\boldsymbol{r}})} \text{ is non-zero)}\\
&= \boldsymbol{r}_{s_1(v_{\boldsymbol{r}})} + 1 && \text{(from the parity condition for } \boldsymbol{r}\text{)}\\
&= \boldsymbol{f}_{s_1(v_{\boldsymbol{r}})} + 1 && \text{(from the maximality of } \boldsymbol{r}\text{)}
\end{aligned}$$

This gives a contradiction with the parity condition of Definition 2.2.3 as $\boldsymbol{f}_{s_0(v_{\boldsymbol{r}})} > \boldsymbol{f}_{s_1(v_{\boldsymbol{r}})} + 1$. □

Proof of Lemma 2.3.3. We start by proving that any partial run profile $\boldsymbol{r}$ has at most one cycle and that this cycle always contains the end-vertex $v_{\boldsymbol{r}}$. We proceed by induction on the length of the run profile, i.e., the number of steps in the algorithm Run. The base case is the initial run profile 0^{2n} for which $G^*_{0^{2n}}$ contains no edge and thus also no cycle. For the induction step, let us assume that a vector $\boldsymbol{r}^i$ is a partial run profile and the vector $\boldsymbol{r}^{i+1}$ is the partial run profile after one step from $\boldsymbol{r}^i$.

Observe that the graph $G^*_{\boldsymbol{r}^{i+1}}$ can be obtained from the graph $G^*_{\boldsymbol{r}^i}$ by removing the outgoing edge from the end-vertex $v_{\boldsymbol{r}^i}$ (if there is any such edge) and adding an edge $(v_{\boldsymbol{r}^i}, v_{\boldsymbol{r}^{i+1}})$ (see Figure 2.2 for illustration). If $G^*_{\boldsymbol{r}^i}$ contains a cycle then removing the edge from the end-vertex $v_{\boldsymbol{r}^i}$ to its successor in $G^*_{\boldsymbol{r}^i}$ produces a cycle-free graph. Adding an edge between $v_{\boldsymbol{r}^i}$ and $v_{\boldsymbol{r}^{i+1}}$ creates a graph with at most one cycle. Moreover, if there is a cycle then it has to contain $v_{\boldsymbol{r}^{i+1}}$.

To prove the reverse implication, assume we have a partial switching flow $\boldsymbol{f}$ that satisfies the conditions from the statement of this lemma, i.e., $G^*_{\boldsymbol{f}}$ has at most one cycle and if the cycle is present it contains the end-vertex $v_{\boldsymbol{f}}$. Let us denote by $\boldsymbol{r}$ the longest partial run profile such that all its coordinates are at most $\boldsymbol{f}$, that is $\boldsymbol{r}_e \leq \boldsymbol{f}_e$ for each $e \in E$. Consider the difference $\boldsymbol{\delta} = \boldsymbol{f} - \boldsymbol{r}$ of the switching flow and its longest run profile. We complete the proof by the following case analysis:

1. If the end-vertex $v_{\boldsymbol{f}}$ is the same as the end-vertex $v_{\boldsymbol{r}}$ ($v_{\boldsymbol{f}} = v_{\boldsymbol{r}}$), then the difference $\boldsymbol{\delta} = \boldsymbol{f} - \boldsymbol{r}$ is a flow, and moreover it is a circulation. That is the Kirchhoff's law is satisfied in all vertices of G:

$$\forall v \in G\colon \sum_{e=(u,v)\in E} \boldsymbol{\delta}_e - \sum_{e=(v,w)\in E} \boldsymbol{\delta}_e = 0 \ .$$

   If $\boldsymbol{\delta}$ is identically zero we are done as $\boldsymbol{f} = \boldsymbol{r}$, and thus $\boldsymbol{f}$ is a partial run profile. Otherwise, we show that the circulation $\boldsymbol{\delta}$ will result in a cycle in $G^*_{\boldsymbol{f}}$. Namely we prove that for any vertex v if v has at least one outgoing edge with a non-zero value in $\boldsymbol{\delta}$ then $G^*_{\boldsymbol{f}}$ contains an outgoing edge from v which has non-zero value in $\boldsymbol{\delta}$. Using this and the fact that $\boldsymbol{\delta}$ satisfies the Kirchhoff's law everywhere, we find a cycle in $G^*_{\boldsymbol{f}}$.

   Let u be any vertex of G such that at least one edge $s_0(u)$ and $s_1(u)$ is non-zero in $\boldsymbol{\delta}$. Observe that if both $s_0(u)$ and $s_1(u)$ are non-zero in $\boldsymbol{\delta}$ then $G^*_{\boldsymbol{f}}$ contains one of them. Now assume that only one edge from $s_0(u)$, $s_1(u)$ is non-zero $\boldsymbol{\delta}$. We claim that then this non-zero edge is contained in the graph $G^*_{\boldsymbol{f}}$. There are only two possible cases:

   (a) Either $\boldsymbol{r}_{s_0(u)} = \boldsymbol{f}_{s_0(u)}$, and thus $\boldsymbol{r}_{s_0(u)} = \boldsymbol{r}_{s_1(u)} - 1$ and $\boldsymbol{f}_{s_0(u)} = \boldsymbol{f}_{s_1(u)}$ (see Table 2.1a), and $G^*_{\boldsymbol{f}}$ contains the edge $s_1(u)$,

   (b) or $\boldsymbol{r}_{s_1(u)} = \boldsymbol{f}_{s_1(u)}$, and thus $\boldsymbol{r}_{s_0(u)} = \boldsymbol{r}_{s_1(u)}$ and $\boldsymbol{f}_{s_0(u)} = \boldsymbol{f}_{s_1(u)} + 1$ (see Table 2.1b), and $G^*_{\boldsymbol{f}}$ contains the edge $s_0(u)$.

   To construct the cycle we proceed as follows:

   (a) First choose any edge $e = (u_0, u_1)$ in $G^*_{\boldsymbol{f}}$ which is non-zero in $\boldsymbol{\delta}$. Such an edge exists since for any vertex u such that $s_0(u)$ or $s_1(u)$ is non-zero in $\boldsymbol{\delta}$, the graph $G^*_{\boldsymbol{f}}$ contains $s_b(u)$ for $b \in \{0,1\}$ such that $s_b(u)$ is non-zero in $\boldsymbol{\delta}$.

   (b) Proceed by picking adjacent edge in $G^*_{\boldsymbol{f}}$ which is a non-zero edge in $\boldsymbol{\delta}$. Assume that we have chosen edges $(u_0, u_1), (u_1, u_2), \ldots (u_{i-1}, u_i)$, then we claim that either $s_0(u_i)$ or $s_1(u_i)$ is non-zero in $\boldsymbol{\delta}$ and is contained

	$s_0(u)$	$s_1(u)$
$\boldsymbol{\delta}$	0	1
$\boldsymbol{r}$	a	$a-1$
$\boldsymbol{f}$	a	a

(a) Case 1a.
The left column is filled in by the assumptions of Case 1a. To fill in the right column in such a way that δ is non-zero and the values preserve the parity condition, we have to set $\boldsymbol{r}_{s_1(u)} = a-1$ and $\boldsymbol{f}_{s_1(u)} = a$.

	$s_0(u)$	$s_1(u)$
$\boldsymbol{\delta}$	1	0
$\boldsymbol{r}$	a	a
$\boldsymbol{f}$	$a+1$	a

(b) Case 1b.
The right column is filled in by the assumptions of Case 1b. To fill in the left column in such a way that δ is non-zero and the values preserve the parity condition, we have to set $\boldsymbol{r}_{s_0(u)} = a$ and $\boldsymbol{f}_{s_0(u)} = a+1$.

Table 2.1: Case analysis from the proof of Lemma 2.3.3 when only one of the edges $s_0(u)$ and $s_1(u)$ is non-zero in $\boldsymbol{\delta}$.

in $G^*_{\boldsymbol{f}}$. This is because by Kirchhoff's law either $s_0(u_i)$ or $s_1(u_i)$ is non-zero in $\boldsymbol{\delta}$ and by the fact that $G^*_{\boldsymbol{f}}$ contains $s_b(u_i)$ for $b \in \{0,1\}$ such that $s_b(u_i)$ is non-zero in $\boldsymbol{\delta}$.

As there are only finitely many vertices the above mentioned procedure produces a directed cycle. By Observation 2.3.4, we know that all outgoing edges from $v_{\boldsymbol{r}} = v_{\boldsymbol{f}}$ are zero in $\boldsymbol{\delta}$, and thus the end-vertex $v_{\boldsymbol{f}}$ is not contained in the cycle we have found.

2. If the end-vertex $v_{\boldsymbol{f}}$ is not the same as the end-vertex $v_{\boldsymbol{r}}$ ($v_{\boldsymbol{f}} \neq v_{\boldsymbol{r}}$) then we get a contradiction with $\boldsymbol{f}$ being a partial switching flow. It follows from Observation 2.3.4 that

$$\sum_{e=(u,v_{\boldsymbol{r}})\in E} \boldsymbol{f}_e - \sum_{e=(v_{\boldsymbol{r}},w)\in E} \boldsymbol{f}_e \geq \sum_{e=(u,v_{\boldsymbol{r}})\in E} \boldsymbol{r}_e - \sum_{e=(v_{\boldsymbol{r}},w)\in E} \boldsymbol{r}_e = \begin{cases} 0 & v_{\boldsymbol{r}} = o, \\ 1 & \text{otherwise,} \end{cases}$$

which is in contradiction with $\boldsymbol{f}$ being a partial switching flow for which the end-vertex $v_{\boldsymbol{f}}$ differs from $v_{\boldsymbol{r}}$.

This concludes the proof of Lemma 2.3.3. □

Lemma 2.3.5. *It is possible to verify in polynomial time whether a vector is a run profile.*

Proof. We can check that a vector $\boldsymbol{f}$ is a switching flow in polynomial time due to Dohrau et al. [2017]. The construction of the graph $G^*_{\boldsymbol{f}}$ is polynomial by Observation 2.3.2. Lemma 2.3.3 gives us a polynomial time procedure to check if $\boldsymbol{f}$ is also a run profile as it is sufficient to check if $G^*_{\boldsymbol{f}}$ contains more than one cycle or whether it has a cycle not containing the end-vertex. This check can be done by a simple modification of the standard depth-first search on $G^*_{\boldsymbol{f}}$. □

2.4 The Computational Complexity of `Arrival`

In this section, we use our efficient structural characterization of run profiles from Lemma 2.3.3 to improve the known results about the computational complexity of `Arrival`. Specifically, we show that the decision version of `Arrival` is in $\mathsf{UP} \cap \mathsf{coUP}$ and the search version is in CLS.

2.4.1 The Decision Complexity of `Arrival`

Our upper bound on the decision complexity of `Arrival` follows directly from the work of Dohrau et al. [2017] by application of Lemma 2.3.5.

Theorem 2.4.1. *`Arrival` is in $\mathsf{UP} \cap \mathsf{coUP}$.*

Proof. The unique UP certificate for a YES-instance of `Arrival` is the run profile $\boldsymbol{r}$ returned by the algorithm Run. Clearly, for each YES-instance there exists only one such vector $\boldsymbol{r}$ and $\boldsymbol{r}$ does not exist for NO-instances. By Lemma 2.3.5, we can determine whether a candidate switching flow $\boldsymbol{r}$ is a run profile in polynomial time.

The coUP membership follows directly from the reduction of NO-instances of `Arrival` to YES-instances of `Arrival` as suggested by Dohrau et al. [2017]. The reduction adds to the original graph G a new vertex $\bar{d}$, and for each vertex $v \in V$ such that there is no directed path from v to the destination d, the edges $s_0(v)$ and $s_1(v)$ are replaced with edges $(v, \bar{d})$. This alteration of the original switch graph can be performed in polynomial time. Dohrau et al. [2017] proved that the train eventually arrives either at d or $\bar{d}$. The unique coUP witness for `Arrival` is then a run profile from o to the dead-end $\bar{d}$. $\square$

2.4.2 The Search Complexity of `Arrival`

The search complexity of `Arrival` was first studied by Karthik C. S. [2017], who introduced a total search variant of `Arrival` as follows.

Definition 2.4.2 (`S-Arrival`, Karthik C. S. [2017])**.** *Given a switch graph $G = (V, E, s_0, s_1)$ and a pair of vertices $o, d \in V$, define a graph G' as follows:*

1. *Add a new vertex $\bar{d}$.*
2. *For each vertex v such that there is no directed path from v to d, replace edges $s_0(v)$ and $s_1(v)$ with edges $(v, \bar{d})$.*
3. *Edges $s_0(d), s_1(d), s_0(\bar{d})$, and $s_1(\bar{d})$ are self-loops.*

The problem `S-Arrival` is to find a switching flow in G' either from o to d or from o to $\bar{d}$.

The above Definition 2.4.2 is motivated by the proof of membership in $\mathsf{NP} \cap \mathsf{coNP}$ by Dohrau et al. [2017]. Namely, in order to ensure that a solution for `S-Arrival` always exists, it was necessary to add the dead-end vertex $\bar{d}$ to the switch graph G.

Note that our method for efficient verification of run profiles from Lemma 2.3.5 allows us to define a more natural version of `S-Arrival` directly on the graph G

without any modifications. Instead of relying on the dead-end vertices, we can use the fact that a partial run profile with an edge that was visited for $2^n + 1$ times is an efficiently verifiable witness for NO-instances of `Arrival`.

Definition 2.4.3 (`S-Arrival`– simplified)**.** *The* `S-Arrival` *problem is: Given a switch graph $G = (V, E, s_0, s_1)$ and a pair of vertices $o, d \in V$, find one of the following:*

1. *a run profile $\boldsymbol{r} \in \mathbb{N}^{[2n]}$ from o to d, or*
2. *a run profile $\boldsymbol{r} \in \mathbb{N}^{[2n]}$ from o to any $v \in V$ such that*
   - *$\boldsymbol{r}_{(u,v)} = 2^n + 1$, where u is the last vertex visited by the train before it reached the end-vertex v of $\boldsymbol{r}$, and*
   - *$\boldsymbol{r}_{e'} \leq 2^n$ for all $e' \neq (u, v)$.*

The correspondence of the above version of `S-Arrival` to the original one follows formally from the following lemma.

Lemma 2.4.4 ([Karthik C. S., 2017, Lemma 1])**.** *For any $G = (V, E, s_0, s_1)$ and a pair of vertices $o, d \in V$. Let $\boldsymbol{r}$ be a run profile (thus $v_{\boldsymbol{r}} = d$), then $\boldsymbol{r}_e \leq 2^n$ for each edge $e \in E$.*

To argue membership of our version of `S-Arrival` in `TFNP`, we need to show that both types of solutions in Definition 2.4.3 can be verified efficiently. Solutions of the first type are simply run profiles, and we have already shown that they can be verified in polynomial time in Lemma 2.3.5. In order to be able to verify solutions of the second type, it remains to argue that for any partial run profile, the immediate predecessor of its end-vertex can be determined in polynomial time.

Lemma 2.4.5. *Let $\boldsymbol{r}$ be a partial run profile after $R \geq 1$ steps and u be the vertex visited by the train at step $R - 1$. Then*

1. *either u is the unique predecessor of $v_{\boldsymbol{r}}$ in $G^*_{\boldsymbol{r}}$, or*
2. *there is a single cycle in $G^*_{\boldsymbol{r}}$ containing $v_{\boldsymbol{r}}$ and u is the predecessor of $v_{\boldsymbol{r}}$ on this cycle.*

Proof. First, note that if u is the end-vertex one step before $v_{\boldsymbol{r}}$ becomes the end-vertex then $G^*_{\boldsymbol{r}}$ must contain the edge $(u, v_{\boldsymbol{r}})$, as it is the last edge used by the train to leave u. Thus, in the first case (when $v_{\boldsymbol{r}}$ has only one predecessor in $G^*_{\boldsymbol{r}}$) the immediate predecessor of $v_{\boldsymbol{r}}$ in the partial run $\boldsymbol{r}$ is unambiguously given by the only predecessor of $v_{\boldsymbol{r}}$ in $G^*_{\boldsymbol{r}}$.

For the second case we show that $G^*_{\boldsymbol{r}}$ contains a directed cycle C (containing the end-vertex $v_{\boldsymbol{r}}$) and u is unambiguously given by the predecessor of $v_{\boldsymbol{r}}$ in $G^*_{\boldsymbol{r}}$ that lies on C. We find the cycle C by constructing the longest possible directed path $c_0 = v_{\boldsymbol{r}}, c_1, \ldots, c_k$ in $G^*_{\boldsymbol{r}}$ without repeating vertices. Note that it cannot happen that c_k has no outgoing edge in $G^*_{\boldsymbol{r}}$. Otherwise, $\boldsymbol{r}$ would have two different end-vertices $v_{\boldsymbol{r}}$ and c_k (as having no outgoing edge in $G^*_{\boldsymbol{r}}$ means that the train has never left this vertex). By Lemma 2.3.3, the directed edge from c_k has to end in the end-vertex $v_{\boldsymbol{r}}$, or else there would be a cycle in $G^*_{\boldsymbol{r}}$ that avoids $v_{\boldsymbol{r}}$.

The algorithm Run takes R steps to generate the run profile $\boldsymbol{r}$, i.e., $\sum_{e \in E} \boldsymbol{r}_e = R$. Let $t_{\boldsymbol{r}}\colon V \to \{0, 1, \dots, R-1\}$ be the function returning the last step after which a vertex was left by the train in the partial run profile $\boldsymbol{r}$. Observe that, except for the edge through which the train arrived to $v_{\boldsymbol{r}}$,[2] it holds for all edges $(x, y) \in G^*_{\boldsymbol{r}}$ that $t_{\boldsymbol{r}}(x) < t_{\boldsymbol{r}}(x) + 1 \leq t_{\boldsymbol{r}}(y)$. However if we consider the vertices $c_0 = v_{\boldsymbol{r}}, c_1, \dots, c_k$ on the cycle C, clearly the above inequality must have been broken for some edge (c_i, c_j) (otherwise $t_{\boldsymbol{r}}(c_0) < t_{\boldsymbol{r}}(c_1) < \ldots < t_{\boldsymbol{r}}(c_k) < t_{\boldsymbol{r}}(c_0)$ which is a contradiction). As only the last used edge can break the inequality it must be the case that the train was in vertex c_k at the step $R-1$. $\square$

Observation 2.4.6. *`S-Arrival` (defined in Definition 2.4.2) reduces to simplified `S-Arrival` (defined in Definition 2.4.3).*

Proof. Given a solution of the second type of the simplified `S-Arrival`, i.e., the long run profile, we can get a run profile $\boldsymbol{r}$ to $\bar{d}$ in polynomial time. For each vertex u we can determine whether there is an oriented path from it to the destination d, and if there is no such path we set $\boldsymbol{r}_{s_0(v)} = \boldsymbol{r}_{s_1(v)} = 0$. We compute the end vertex $v_{\boldsymbol{r}}$ and set $s_0(v_{\boldsymbol{r}}) = 1$. All other components of $\boldsymbol{r}$ are set according to the original solution of the simplified `S-Arrival`. $\square$

`S-Arrival` is in CLS

Karthik C. S. [2017] showed that `S-Arrival` is contained in the class PLS. We improve this result and prove that `S-Arrival` is in fact contained in CLS. We show this result by reducing `S-Arrival` to a problem `End-Of-Metered-Line` (defined by Hubáček and Yogev [2020]) which is contained in CLS as shown by Hubáček and Yogev [2020] (see Section 1.1.1). For the better readability we restate the definition of `End-Of-Metered-Line` problem here too:

Definition 1.1.7. (Restated) *`End-Of-Metered-Line` is the following problem: Given circuits $S, P\colon \{0,1\}^m \to \{0,1\}^m$, and $V\colon \{0,1\}^m \to [2^m] \cup \{0\}$ such that $P(0^m) = 0^m \neq S(0^m)$ and $V(0^m) = 1$, find a string $x \in \{0,1\}^m$ satisfying one of the following:*

1. *Either $P(S(x)) \neq x$ or $S(P(x)) \neq x \neq 0^m$,*
2. *$x \neq 0^m$ and $V(x) = 1$,*
3. *either*
$$V(x) > 0 \text{ and } V(S(x)) - V(x) \neq 1$$
or
$$V(x) > 1 \text{ and } V(x) - V(P(x)) \neq 1.$$

The circuits S, P from Definition 1.1.7 implicitly represent a directed graph with vertices labelled by binary strings of length m, where each vertex has both out-degree and in-degree at most one. The circuit P represents the predecessor and the circuit S represents the successor of a given vertex as follows: there is an edge from a vertex u to a vertex v iff $S(u) = v$ and $P(v) = u$. Finally, the

[2]The inequality does not hold for the end vertex $v_{\boldsymbol{r}}$, since the train has not left it yet and thus $t_{\boldsymbol{r}}(v_{\boldsymbol{r}})$ has not been updated to the time R yet.

circuit V can be thought of as an odometer that returns the distance from the trivial source at 0^m or value 0 for vertices lying off the path starting at the trivial source. The task in `End-Of-Metered-Line` is to find a sink or a source different from the trivial one at 0^m (the solutions of the second and of the third type in Definition 1.1.7 ensure that V behaves as explained above). We are now ready to present our reduction from `S-Arrival` to `End-Of-Metered-Line`.

Theorem 2.4.7. *`S-Arrival` can be reduced to `End-Of-Metered-Line` and, thus, it is contained in `CLS`.*

Proof. Let (G, o, d) be an instance of `S-Arrival`. We construct an instance of `EOML` that contains a vertex for each candidate partial switching flow over the switch graph G, i.e., for each vector with $2n$ coordinates and values from $[2^n + 1] \cup \{0\}$. The `EOML` instance comprises of a directed path starting at the initial (empty) partial run profile 0^{2n}. Each vertex on the path has an outgoing edge to its consecutive partial run profile. Any vertex that does not correspond to a partial run profile becomes a self-loop. Finally, the valuation circuit V returns either the number of steps in the corresponding partial run profile or the zero value if the vertex does not correspond to a partial run profile.

Formal description of the circuits S, P, and V defining the above `EOML` graph is given by algorithms Successor (Algorithm 2), Predecessor (Algorithm 3), and Valuation (Algorithm 4)

Algorithm 2: Successor

```
   Input  : a vector $\boldsymbol{x} \in ([2^n+1] \cup \{0\})^{2n}$
   Output: a vector $\boldsymbol{y} \in ([2^n+1] \cup \{0\})^{2n}$

 1 if $\boldsymbol{x}$ is a partial run profile then // efficiently testable by Lemma 2.3.5
 2     compute the end-vertex $v_{\boldsymbol{x}}$
 3     if $v_{\boldsymbol{x}} = d$ or $\boldsymbol{x}_e = 2^n + 1$ for some $e \in E$ then
 4         return $\boldsymbol{x}$          // the train terminates or runs for too long
 5     else
 6         $b \leftarrow \boldsymbol{x}(s_1(v_{\boldsymbol{x}})) - \boldsymbol{x}(s_0(v_{\boldsymbol{x}}))$  // parity of the current visit at $v_{\boldsymbol{x}}$
 7         $e \leftarrow s_b(v_{\boldsymbol{x}})$                     // the next edge to traverse
 8         $\boldsymbol{y}_h \leftarrow \begin{cases} \boldsymbol{x}_h + 1 & \text{if } h = e \\ \boldsymbol{x}_h & \text{otherwise} \end{cases}$      // run profile update
 9         return $\boldsymbol{y}$
10     end
11 else
12     return $\boldsymbol{x}$                                   // self-loop
13 end
```

A polynomial bound on the size of the circuits S, P, and V follows directly from Observation 2.3.2 (computing G^*), Lemma 2.3.5 (testing whether a given vector is a partial run profile), Observation 2.2.6 (computing the end-vertex), and Lemma 2.4.5 (computing the previous position of the train).

Lemma 2.3.3 and Lemma 2.4.5 imply that the `EOML` graph indeed consists of a single directed path and isolated vertices with self-loops. By the construction of V (it outputs the number of steps of the train), there are no solutions of

Algorithm 3: PREDECESSOR

```
Input  : a vector $\boldsymbol{x} \in ([2^n+1] \cup \{0\})^{2n}$
Output: a vector $\boldsymbol{y} \in ([2^n+1] \cup \{0\})^{2n}$

1  if $\boldsymbol{x} = \mathbf{0}$ then
2      return $\mathbf{0}$                                    // the trivial source
3  end
4  if $\boldsymbol{x}$ is a partial run profile then // efficiently testable by Lemma 2.3.5
5      compute the end-vertex $v_{\boldsymbol{x}}$
       // correctness by Lemma 2.4.5
6      if $v_{\boldsymbol{x}}$ has a single predecessor in $G^*_{\boldsymbol{x}}$ then
7          $e \leftarrow$ the only incoming edge of $v_{\boldsymbol{x}}$ in $G^*_{\boldsymbol{x}}$
8      else
9          $e \leftarrow$ the only incoming edge of $v_{\boldsymbol{x}}$ which lies on a directed cycle
10     end
11     if $\boldsymbol{x}_e \leq 2^n+1$ and $\boldsymbol{x}_{e'} < 2^n+1$ for all $e' \neq e$ then
12         $\boldsymbol{y}_h \leftarrow \begin{cases} \boldsymbol{x}_h - 1 & \text{if } h = e \\ \boldsymbol{x}_h & \text{otherwise} \end{cases}$
13         return $\boldsymbol{y}$
14     end
15 end
16 return $\boldsymbol{x}$                                      // self-loop
```

Algorithm 4: VALUATION

```
Input  : a vector $\boldsymbol{x} \in ([2^n+1] \cup \{0\})^{2n}$
Output: a value $v \in \mathbb{N}$

1  $v \leftarrow 0$                              // default value for self-loops
2  if $\boldsymbol{x}$ is a partial run profile then // efficiently testable by Lemma 2.3.5
3      $v \leftarrow 1 + \sum_{i=1}^{2n} \boldsymbol{x}_i$
4  end
5  return $v$
```

the second or the third type (cf. Definition 1.1.7). Thus, the `EOML` instance has a unique solution which has to correspond to a run profile in the original `S-Arrival` instance or to a partial run profile certifying that the train ran for too long (see the second type of solution in Definition 2.4.3). □

3. Worst-Case Hardness in TFNP and Average-Case Hard UP Problem

In this chapter, we show that it is not possible to create in a black-box way a worst-case hard TFNP problem from average-case hard problem in NP or even in the class UP. This black-box separation could be seen as a warm-up for Chapter 4, where we rule out the existence of a simple fully black-box construction from injective one-way functions. Intuitively, it is more difficult to come up with a reduction from any average-case hard problem in NP than from a specific one – inversion of an injective one-way function. As the construction from injective one-way functions may utilize the restricted nature of the problem compared to any average-case hard problem in NP. Thus, the separation presented in this chapter is technically much easier than the one given in Chapter 4 and rules out any (not just "simple") fully black-box construction.

We describe our proof informally in Section 3.1. In Section 3.2, we formalize the notation and definitions used in this chapter. In Section 3.3, we formally define fully black-box separations, present our oracle Solve and show our main results – the separation between average-case hardness in NP and worst-case hardness in TFNP. The core of the argument is presented in Section 3.4, where we prove all the lemmata used to prove our main result.

3.1 Our Techniques

Our results employ the framework of black-box separations (see e.g., Impagliazzo and Rudich [1989], Reingold et al. [2004], Fischlin [2012]). The approach suggested in Impagliazzo and Rudich [1989] for demonstrating that there is no fully black-box construction of a primitive P from another primitive Q is to come up with an oracle O relative to which Q exists, but every black-box implementation C^Q of P is broken. However, as explained in Reingold et al. [2004], Matsuda and Matsuura [2011], this approach actually rules out so-called "relativized" constructions (which are less restricted than fully black-box constructions). To rule out just fully black-box constructions it suffices to use the so-called two-oracle technique introduced by Hsiao and Reyzin [2004]. Here, the oracle O usually consists of two parts: an idealised implementation of the primitive Q and a "breaker" oracle for primitive P. In our context, P corresponds to a TFNP problem and the oracle O corresponds to an oracle relative to which average-case hard NP problem exists and a procedure Solve which provides a solution for any instance of a TFNP problem.

As an oracle relative to which average-case hard NP problem exists, we use the random language oracle defined by Brzuska and Couteau [2020]. Brzuska and Couteau [2020] proved that the random language gives us an average-case hard problem in NP if we are given an oracle access to it. To rule out the existence of fully black-box constructions of worst-case hard (and, thus, also average-case hard) TFNP problems from average-case hard NP language, we have to argue that an oracle access to such a "breaker" oracle Solve for TFNP does not help any security reduction R in deciding whether the given instance is a YES-instance or a NO-instance.

The existence of a "useless" solution When constructing our oracle Solve, we want to make sure that it returns a solution which does not "help" security reduction to distinguish whether an instance x is a YES-instance or a NO-instance. For us, such a "useless" solution is a solution on which no pair $(x, w) \in W$ is queried, where W is the relation specifying the NP problem, i.e., $(x', w') \in W$ if and only if x' is YES-instance and w' is a witness for x'. Giving a security reduction R a solution which queries such a pair is very helpful for R as it reveals the witness w and R can for sure know that x is a YES-instance. On the other hand, if we avoid returning such solutions the information which is given to the security reduction R from Solve is only negative. In the sense that it rules out few witnesses w (i.e., indicate for them that $(x, w) \notin W$), but there might still exists another witness w' such that $(x, w') \in W$. That means that the returned solution does not reveal whether the instance is a YES-instance or not. The problem is that our oracle Solve does not know the exact challenge x on which the security reduction R is running. Thus, instead of protecting the pairs $(x, w) \in W$ for one particular x, the oracle Solve protects these challenge-witness pairs simultaneously for many different challenges x.

The existence of such a solution follows from the totality of the problem. As there has to be a solution present no matter the number of YES-instances. Thus, we consider a language which has only NO-instances (or more specifically all instances of at least some length are NO-instances). By the totality, we get a solution for such a language. And then show that this solution remains to be present with high probability even when we consider languages taken from the random distribution. To formalize that the solution is not helpful we provide a security experiment in which the reduction cannot do better than guess whether the instance is YES-instance or NO-instance and prove that the view for R is the same in the "real" execution with high probability.

3.2 Preliminaries

For any relation $W \in \{0,1\}^n \times \{0,1\}^n$, let χ_W denote the characteristic function on the first coordinate of W. That is, χ_W is the following function:

$$\forall x \in \{0,1\}^* : \chi_W(x) = \begin{cases} 1 & \text{if } \exists w \in \{0,1\}^* \text{ such that } (x, w) \in W, \\ 0 & \text{otherwise} \end{cases}$$

For strings $x, y \in \{0,1\}^*$, we use $x \leq_{\text{lex}} y$ or $y \geq_{\text{lex}} x$ to denote that x is lexicographically smaller than or equal to y. For any finite set A, $x \leftarrow A$ denotes that x is chosen uniformly at random from the elements of A.

Definition 3.2.1 ($\mathsf{UniqWitness}$). *We define a class of relations $\mathsf{UniqWitness}$, where a relation W belongs to $\mathsf{UniqWitness}$ if $W \subseteq \bigcup_{n=1}^{\infty} \left(\{0,1\}^n \times \{0,1\}^n\right)$ and for any $x \in \{0,1\}^*$, there exists at most one $w \in \{0,1\}^*$ such that $(x, w) \in W$.*

To give an intuition behind the above definition, we want the class $\mathsf{UniqWitness}$ to correspond to the class UP in a sense that given an oracle access to a relation $W \in \mathsf{UniqWitness}$ and given an input $x \in \{0,1\}^*$ we can decide in nondeterministic polynomial time whether there exits a witness $w \in \{0,1\}^*$ such that $(x, w) \in W$ and, moreover, there is at most one such w for each x. Then, we may define the YES-instances as the strings $x \in \{0,1\}^*$ for which there exists $w \in \{0,1\}^*$ such that $(x, w) \in W$ and NO-instances to correspond to those $x \in \{0,1\}^*$ for which no such witness $w \in \{0,1\}^*$ exists.

To restrict ourself to only a specific word length, we use the following notation.

Notation 3.2.2. *Let $n \in \mathbb{N}$. For any relation $W \in \bigcup_{n\in\mathbb{N}} \{0,1\}^n \times \{0,1\}^n$ we define*

- $W_{<n} = \{(x, w) \in W : |x| = |w| < n\}$,
- $W_n = \{(x, w) \in W : |x| = |w| = n\}$, *and*
- $W_{>n} = \{(x, w) \in W : |x| = |w| > n\}$.

We consider the following distribution over problems from $\mathsf{UniqWitness}$, which was defined by Brzuska and Couteau [2020].

Definition 3.2.3 (Distribution $\mathtt{RL}$, Brzuska and Couteau [2020])**.** *For any natural number $n \in \mathbb{N}$, define the following distribution over*

$$\mathsf{UniqWitness}_n = \{W_n \mid W \in \mathsf{UniqWitness}\}$$

called $\mathtt{RL}_n$. Where $W_n \leftarrow \mathtt{RL}_n$ is sampled as follows: For every $x \in \{0,1\}^n$, x is a YES-instance with probability $1/2$. For every YES-instance x, we choose a witness w, such that $(x, w) \in W_n$, uniformly at random from $\{0,1\}^n$ (let us emphasize that the random choices are independent for different choices of x).

We define the distribution $\mathtt{RL}$ over all languages from $\mathsf{UniqWitness}$ in such a way that for any word length $n \in \mathbb{N}$ we sample the relations W_n using $\mathtt{RL}_n$, i.e., $W \leftarrow \mathtt{RL}$ denotes that $W = \bigcup_{n\in\mathbb{N}} W_n$ where each $W_n \leftarrow \mathtt{RL}_n$.

3.3 Separating TFNP and Average-Case Hardness in UP

In this section, we prove that there is no fully black-box construction of a worst-case hard TFNP problem from an average-case hard NP (resp. UP) problem. Let us first formally define a fully black-box construction in this context.

Definition 3.3.1 (Fully black-box construction of a worst-case hard TFNP problem from an average-case hard UP problem)**.** *A* fully black-box construction of a worst-case hard TFNP problem from an average-case hard UP problem *is a tuple (R, T_R, C, T_C, p) of oracle-aided algorithms R, C, functions T_R, T_C, and a polynomial p satisfying the following properties:*

1. ***Efficiency:*** *For any $W \in \mathsf{UniqWitness}$, any input $x \in \{0,1\}^*$, and any choice of the internal randomness $r \in \{0,1\}^*$, the algorithm $R^W(x; r)$ halts in time $T_R(|x|)$.*

   For any $W \in \mathsf{UniqWitness}$, and any inputs $i, s \in \{0,1\}^*$, the algorithm $C^W(i, s)$ halts in time $T_C(|i| + |s|)$.

2. ***Correctness:*** *For each $W \in \mathsf{UniqWitness}$ and $i \in \{0,1\}^*$, there exists $s \in \{0,1\}^*$ such that $|s| \leq p(|i|)$ and $C^W(i,s) = 1$.*[1]

3. ***Black-box proof of security:*** *There exists a polynomial q such that for any $W \in \mathsf{UniqWitness}$, any sampler $\textsc{Samp}$, and any oracle-aided algorithm $\textsc{Solve}$, if for all instances $i \in \{0,1\}^*$*

$$\textsc{Solve}^W(i) \text{ returns } s \text{ such that } C^W(i,s) = 1$$

then there exists infinitely many $n \in \mathbb{N}$ such that for all $W \in \mathsf{UniqWitness}$:

$$\Pr_{\substack{x \leftarrow \textsc{Samp}(1^n),\\ r \leftarrow \{0,1\}^{T_R(n)}}} \left[R^{W,\textsc{Solve}}(x;r) = \chi_W(x) \right] \geq \frac{1}{2} + \frac{1}{q(n)},$$

where r represents the internal random choices of R.[2]

Definition 4.3.1 has the following semantics. The deterministic algorithm C specifies the TFNP problem and the algorithm R is the (security) reduction which, given access to any TFNP solver, solves the average-case hard NP problem. First, we illustrate the definition of fully black-box construction using the construction of a hard `Pigeon` problem (see Definition 1.1.3) from one-way permutations which was shown by Papadimitriou [1994]. Then, we provide some additional remarks on important points in the above definition.

Example of a fully black-box construction. The circuit P in the `Pigeon` construction of Papadimitriou [1994] is as follows: Given a permutation π and an inversion challenge y of length n, the `Pigeon` circuit is defined as

$$P_y^\pi(x) = \pi(x) \oplus y.$$

First, observe that as π is a permutation $P_y^\pi(x)$ has no collision, i.e., for any $x \neq x'$ it holds that $P_y^\pi(x) \neq P_y^\pi(x')$. Thus, there exists exactly one solution in this `Pigeon` instance, which is x such that $P_y^\pi(x) = 0^n$. Then $\pi(x) \oplus y = 0^n$, which implies that x is the preimage of y and the returned solution provides us the preimage of the challenge y.

To illustrate our definition of a fully black-box construction, for any permutation π and challenge y, the circuit P_y^π corresponds to an instance. The algorithm C on input $\left(P_y^\pi, x\right)$, respectively $\left(P_y^\pi, (x,x')\right)$, outputs 1 if and only if $P_y^\pi(x) = 0^n$, respectively if $P_y^\pi(x) = P_y^\pi(x')$. The reduction algorithm $R(1^n, y)$ simply queries the TFNP solver $\textsc{Solve}$ with the instance $i = P_y^\pi$, i.e., a circuit computing the function $P_y^\pi(x) = \pi(x) \oplus y$, and outputs the solution s returned by $\textsc{Solve}$ for which, by construction, $\pi(s) = y$.

[1]This corresponds to totality of the TFNP problem, see Definition 1.1.1.

[2]Note that we can restrict the length of the string r by $T_R(n)$ since $T_R(n)$ upper bounds the running time of the reduction R and, thus, it cannot do more than this many random choices. If the reduction does less choices we may simply ignore the last bits of r which would correspond to unused random choices of R.

Constructions of worst-case vs. average-case hardness in TFNP. Notice that our Definition 4.3.1 considers constructions of a *worst-case* hard TFNP problem – the reduction has access to Solve which is promised to return a solution to *any* instance of the TFNP problem. To capture constructions of *average-case* hardness in TFNP, we would need to extend the construction with an efficiently sampleable distribution D of instances of the TFNP problem and require the reduction to solve the problem from UniqWitness when given access to any Solve that returns solutions for instances coming from the specific distribution D (see Definition 5.1 in Rosen et al. [2017]). However, given that we are proving a black-box separation, showing impossibility for worst-case hardness is a stronger result.

The quality of Solve. Note that we consider security reductions which solve the problem from UniqWitness given access to Solve which outputs a solution with probability 1, whereas some definitions in the literature allow the reduction to work only with some non-negligible probability. This also makes our negative result stronger – it is potentially easier to give a reduction when given access to Solve which is guaranteed to always return a solution.

We prove the following theorem.

Theorem 3.3.2. *There is no fully black-box construction (R, T_R, C, T_C, p) of a worst-case hard TFNP problem from average-case hard UP problem with a success probability of the security reduction better than $1/2 + 2^{-0.9n}$ such that both running times $T_R, T_C \in O\left(\ell^{polylog(\ell)}\right)$, where n is the security parameter and ℓ corresponds to the length of the input of R, resp. C.*

As the average-case hard distribution in UP is also an average-case hard distribution for NP we immediately get the following corollary.

Corollary 3.3.3. *There is no fully black-box construction (R, T_R, C, T_C, p) of a worst-case hard TFNP problem from average-case hard NP problem with a success probability of the security reduction better than $1/2 + 2^{-0.9n}$ such that both running times $T_R, T_C \in O\left(\ell^{polylog(\ell)}\right)$, where n is the security parameter and ℓ corresponds to the length of the input of R, resp. C.*

The security reduction may retrieve the information whether a given word is in the language (i.e., whether for its challenge x there exists a witness w such that $(x, w) \in W$) in various ways. It may query the challenge-witness pair (i.e., the pair $(x, w) \in W$) directly or it may retrieve some information by querying to Solve. We say that it queries the challenge-witness pair indirectly if it is queried on the solution returned from Solve. We distinguish the following sets of queries based on where they originate and which oracle is queried.

Notation 3.3.4 (Query sets Q). *We distinguish the following sets of queries to oracles depending where these queries originated and which oracle is queried.*

- *Let $Q\left(C^W(i, s)\right)$ denote the set of all pairs $(x, w) \in \{0,1\}^* \times \{0,1\}^*$ on which the oracle W has been queried by C running on an input (i, s) .*
- *Let $Q_{\text{Solve}}\left(R^{W,\text{Solve}}(x; r)\right)$ denote the set of all instances $i \in \{0,1\}^*$ on which the oracle Solve has been queried by the reduction R running on an input $x \in \{0,1\}^*$ and with internal randomness $r \in \{0,1\}^*$.*

- *Let $Q_W^{dir}\left(R^{W,\textsc{Solve}}(x;r)\right)$ denote the set of all pairs $(x',w) \in \{0,1\}^* \times \{0,1\}^*$ on which the oracle W has been queried by the reduction R running on an input $x \in \{0,1\}^*$ and with internal random choices $r \in \{0,1\}^*$.*
- *Let $Q_W^{indir}\left(R^{W,\textsc{Solve}}(x;r)\right)$ denote the set of pairs $(x',w) \in \{0,1\}^* \times \{0,1\}^*$ on which the oracle W has been queried indirectly, i.e., all pairs that have been queried by the algorithm C when running on an input (i,s) where $i \in Q_{\textsc{Solve}}\left(R^{W,\textsc{Solve}}(x;r)\right)$ and $s = \textsc{Solve}^W(i)$.*
- *Let $Q_W\left(R^{W,\textsc{Solve}}(x;r)\right) = Q_W^{dir}\left(R^{W,\textsc{Solve}}(x;r)\right) \cup Q_W^{indir}\left(R^{W,\textsc{Solve}}(x;r)\right)$.*

Note that these sets may have non-empty intersection.

Solve algorithm: Our algorithm Solve is described in Algorithm 5. The intuition behind the algorithm is that we list all possible solutions which can be returned in the set $S_{i,W}$ and try to return any "benign" solution. That is, a solution which does not query a tuple (x,w) which is in the relation W. If such a solution exists, we return it and, intuitively, this solution should not be helpful for a security reduction as it does not reveal any witnesses.

Unfortunately, it may happen that no such "benign" solution exists. If there is no solution which could be returned, we have to relax the conditions on the "benign" solutions. In Section 3.4.2, we show that if we allow Solve to return some witnesses in the order of increasing challenge length, then Solve does not "help" the security reduction too much.

Algorithm 5: $\textsc{Solve}_C^W(i)$

```
Hardwired      : a fully black-box construction $(R, T_R, C, T_C, p)$ of a worst-case
                 hard TFNP problem from an average-case hard UP problem
Oracle access: an oracle for $W \in$ UniqWitness, i.e., an oracle which on input
                 $(x, w)$ returns 1 if and only if $(x, w) \in W$.
Input          : an instance $i \in \{0,1\}^*$
Output         : a solution $s \in \{0,1\}^*$ such that $C^W(i,s) = 1$ and $|s| \leq p(|i|)$

1 $S_{i,W} = \left\{ s \in \{0,1\}^* \;\middle|\; C^W(i,s) = 1 \text{ and } |s| \leq p(|i|) \right\}$
2 $\ell = 1$
3 while True do
4     $B_{i,W,\ell} = \left\{ s \in S_{i,W} \;\middle|\; Q\left(C^W(i,s)\right) \cap W_{\geq \ell} = \emptyset \right\}$
5     if $B_{i,W,\ell} \neq \emptyset$ then
6         return lexicographically smallest $s \in B_{i,W,\ell}$
7     end
8     $\ell = \ell + 1$
9 end
```

The correctness of the algorithm Solve (Algorithm 5) is discussed in Section 3.4.1, where we prove that Solve always halts and returns a string which is a solution for the queried instance. That is, we prove that TFNP is easy in the presence of oracle Solve. We state the lemma here and provide its proof in Section 3.4.1.

Lemma 3.3.5. *Let (R, T_R, C, T_C, p) be a fully black-box construction of a worst-case hard* TFNP *problem from an average-case hard* UP *problem. For every instance $i \in \{0,1\}^*$ and every relation $W \in$* UniqWitness*, algorithm* Solve *(Algorithm 5) returns a solution, i.e., a string $s \in \{0,1\}^*$ of length at most $p(|i|)$ such that $C^W(i,s) = 1$.*

The probability that the security reduction is successful in deciding whether its challenge x is a YES-instance (i.e., the probability that $R^{W,\textsc{Solve}}(x;r)$ returns the value $\chi_W(x)$) is upper bounded in Section 3.4.2, where we prove the following lemma.

Lemma 3.3.6. *Let (R, T_R, C, T_C, p) be a fully black-box construction of a worst-case hard* TFNP *problem from an average-case hard* UP *problem and*

$$q_C(n) = \max \left| Q\left(C^{W_{<n}}(i,s)\right)\right|,$$

where the maximum is over $W \in$ UniqWitness*, $x \in \{0,1\}^n$, $r \in \{0,1\}^{T_R(n)}$, $i \in Q_{\textsc{Solve}}\left(R^W(x;r)\right)$ and $s \in \{0,1\}^*$ such that $|s| \leq p(|i|)$. Then, for any $n \in \mathbb{N}$, there exists $W \in$* UniqWitness *such that:*

$$\Pr_{\substack{x \leftarrow \{0,1\}^n, \\ r \leftarrow \{0,1\}^{T_R(n)}}} \left[R^{W,\textsc{Solve}}(x;r) = \chi_W(x)\right] \leq \frac{1}{2} + \frac{T_R(n)\left(q_C(n)+1\right)}{2^{n+1}}.$$

We can easily upper bound the number of queries made by the algorithm C with respect to the length of the instance i. However, for the statement of the lemma we need to upper bound this number with respect to the length of the challenge on which R is running. Here the function q_C comes into play. Intuitively, this function upper bounds the number of queries of algorithm C on any relevant input (i,s). Where, by relevant input (i,s), we mean the instances i which are queried for some challenge of length n and strings s of length at most $p(|i|)$.

Combining the above lemmata, we prove our Theorem 3.3.2.

Proof of Theorem 3.3.2. Let (R, T_R, C, T_C, p) be any such construction. Then, by Lemma 3.3.5, algorithm Solve (Algorithm 5) returns a valid solution of the underlying TFNP problem with probability 1. On the other hand, by Lemma 3.3.6, for each n there exists a relation $W \in$ UniqWitness for which we can upper bound the probability that R solves the underlying UP problem by:

$$\Pr_{\substack{x \leftarrow \{0,1\}^n, \\ r \leftarrow \{0,1\}^{T_R(n)}}} \left[R^{W,\textsc{Solve}}(x;r) = \chi_W(x)\right] \leq \frac{1}{2} + \frac{T_R(n)\left(q_C(n)+1\right)}{2^{n+1}},$$

where $q_C(n)$ is the maximum of $\left|Q\left(C^{W'_{<n}}(i,s)\right)\right|$ over the choices of relation $W' \in$ UniqWitness, challenge $x' \in \{0,1\}^n$, randomness $r' \in \{0,1\}^{T_R(n)}$, instance $i \in Q_{\textsc{Solve}}\left(R^{W'}(x';r')\right)$ and a possible solution $s \in \{0,1\}^*$ such that $|s| \leq p(|i|)$.

We observe that, on an input of length n, the security reduction can query an instance of length at most $T_R(n)$ since T_R upper bounds its running time. Thus, we consider strings s of length at most $p(T_R(n))$ and the whole input (i,s) for the algorithm C can be upper bounded by $T_R(n) + p(T_R(n))$. Since T_C

upper bounds the running time of C, we can finally upper bound the size of the set $Q\left(C^{W'_{<n}}(i,s)\right)$ from the above statement and, thus, upper bound $q_C(n)$ by $T_C\left(T_R(n)+p\left(T_R(n)\right)\right)$.

As both T_R and T_C are assumed to be quasi-polynomial functions and p is a polynomial, we get that $q_C(n)$ is $O\left(n^{\mathrm{polylog}(n)}\right)$ and also that

$$T_R(n)\left(q_C(n)+1\right) \in O\left(n^{\mathrm{polylog}(n)}\right) \in o\left(2^{0.1n}\right).$$

Thus, for any large enough n, we can bound the success probability of R for uniform distribution on challenges (that is the probability over the internal randomness $r \leftarrow \{0,1\}^{T_R(n)}$ of the security reduction and the choice of a challenge $x \leftarrow \{0,1\}^n$ that $R^{W,\textsc{Solve}}(x;r)$ returns $\chi_W(x)$ for all $W \in \mathsf{UniqWitness}$) by

$$\begin{aligned}\Pr_{\substack{x\leftarrow\{0,1\}^n,\\ r\leftarrow\{0,1\}^{T_R(n)}}}\left[R^{W,\textsc{Solve}}(x;r)=\chi_W(x)\right] &\le \frac{1}{2}+\frac{T_R(n)\left(q_C(n)+1\right)}{2^{n+1}}\\ &\le \frac{1}{2}+2^{-0.9n}.\end{aligned}$$

Note that the reduction should be successful for any polynomial-time algorithm $\textsc{Samp}$, which samples the challenges $x \in \{0,1\}^n$. Especially it should be successful for our choice of $\textsc{Samp}$ which returns the uniform distribution on $\{0,1\}^n$. Let us also emphasize that in the proof of security (see Definition 3.3.1) we require the security reduction to be successful on infinitely many different $n \in \mathbb{N}$ for every choice of $W \in \mathsf{UniqWitness}$. Thus, the above inequality concludes the proof. $\square$

3.4 Proofs of Supporting Lemmata for Theorem 3.3.2

3.4.1 Solve Always Returns a Solution

Here we prove that algorithm $\textsc{Solve}$ (Algorithm 5) always halts and returns a valid solution. By the nature of the algorithm, it is clear that if it halts the returned solution is valid. Thus, the proof effectively simplifies to showing that $\textsc{Solve}$ halts.

Lemma 3.3.5. (Restated) *Let (R, T_R, C, T_C, p) be a fully black-box construction of a worst-case hard TFNP problem from an average-case hard UP problem. For every instance $i \in \{0,1\}^*$ and every relation $W \in \mathsf{UniqWitness}$, algorithm $\textsc{Solve}$ (Algorithm 5) returns a solution, i.e., a string $s \in \{0,1\}^*$ of length at most $p(|i|)$ such that $C^W(i,s)=1$.*

Proof. If algorithm $\textsc{Solve}$ halts then it returns a string $s \in B_{i,W,\ell}$ for some $\ell \in \mathbb{N}$ and $B_{i,W,\ell} \subseteq S_{i,W}$. Thus, s is a valid solution of length at most $p(|i|)$ satisfying $C^W(i,s)=1$. We only need to show that $\textsc{Solve}$ always returns.

By the totality of the construction C (see Correctness in Definition 3.3.1), the set $S_{i,W}$ is not empty. We consider any solution $s \in S_{i,W}$ and the queries C

makes to the oracle W during the computation of $C^W(i,s)$. Let k be the length of the longest query, that is

$$k = \max_{(x,w)\in Q(C^W(i,s))} |x|.$$

Then, after $k+1$ iterations of the while loop, $\ell = k+1$ and $Q\left(C^W(i,s)\right) \cap W_{\geq \ell} =$ (as, by the definition of k, there are no queries of length at least ℓ). Thus, $s \in B_{i,W,k+1}$ and Solve returns at the very latest at the $(k+1)$-st iteration. □

3.4.2 Success Probability of the Security Reduction Is Small Even in the Presence of Solve

In this section, we prove Lemma 3.3.6 and, thus, upper bound the success probability of the security reduction. To prove the lemma, we consider an experiment in which we invoke the reduction with a fake oracle W' instead of W. More specifically, instead of running R with the real oracle W, we give the reduction an oracle W' which "hides" all challenges of length at least n. That is, no pair (x,w) with x of length n or longer is contained in W' but W' agrees with W on all shorter challenges. In other words, it holds that $W' = W_{<n}$.

It is clear that the probability that $R^{W'}$ returns the value $\chi_W(x)$ correctly when it runs with respect to the oracle W' instead of W is bounded by $1/2$. Indeed, it could only guess whether the given challenge $x \in \{0,1\}^n$ is in the relation with some witness w (i.e., whether there exists $w \in \{0,1\}^n$ for which $(x,w) \in W$ holds). Then, we argue that, with high probability, the views of R when running with respect to W and W' are the same. Thus, proving that, even though R runs with respect to the real oracle W, its success probability is bounded by $1/2 + \epsilon$, where ϵ is the probability that the views differ.

To show that the views are identical with an overwhelming probability, we need to argue that all queries to the oracle Solve are answered identically. For any instance $i \in \{0,1\}^*$, we upper bound the probability that Solve answers differently in the following claim.

Claim 3.4.1. *Let $n \in \mathbb{N}$ be any natural number and $i \in \{0,1\}^*$. For any problem $W \in$ UniqWitness, let $s_{W,n} \in \{0,1\}^*$ denote the solution returned by Solve (Algorithm 5) when running on an instance $i \in \{0,1\}^*$ with respect to $W_{<n}$, that is $s_{W,n} = \text{Solve}^{W_{<n}}(i)$, and let $Q_{W,n}$ be the corresponding set of queries, i.e., $Q_{W,n} = Q\left(C^{W_{<n}}(i, s_{W,n})\right)$.*

Then,

$$\Pr_{W \leftarrow RL}\left[\text{Solve}^W(i) \neq s_{W,n}\right] \leq \frac{|Q_{W,n}|}{2^{n+1}}.$$

Proof. Let i be any instance and $n \in \mathbb{N}$. We show that for any relation $W \in$ UniqWitness:

$$\text{Solve}^W(i) = s_{W,n} \text{ if } Q_{w,n} \cap W_{\geq n} = \emptyset,$$

where $s_{W,n}$ and $Q_{W,n}$ are as defined in the statement of the lemma. Then, using the union bound, we get the desired upper bound as follows:

$$\begin{aligned}\Pr_{W\leftarrow \mathtt{RL}}\left[\textsc{Solve}^W(i) \neq s_{W,n}\right] &\leq \Pr_{W\leftarrow \mathtt{RL}}\left[Q_{W,n} \cap W_{\geq n} \neq \emptyset\right] \\ &\leq \sum_{\substack{(x,w)\in Q_{w,n} \\ |x|\geq n}} \Pr_{W\leftarrow \mathtt{RL}}\left[(x,w) \in W_{\geq n}\right] \\ &\leq \sum_{\substack{(x,w)\in Q_{W,n} \\ |x|\geq n}} \frac{1}{2^{n+1}} \\ &\leq \frac{|Q_{W,n}|}{2^{n+1}},\end{aligned}$$

where the second inequality holds because each x is in the language for some witness w with probability $1/2$ and the witness is chosen uniformly at random from all strings of length $|x| \geq n$. Thus, any pair (x, w) is in the language with probability exactly $1/2^{|x|+1} \leq 1/2^{n+1}$ if $|x| = |w|$ and probability zero otherwise.

Let us fix any relation $W \in \mathsf{UniqWitness}$ and a solution $s_{W,n}$, as well as, the query set $Q_{W,n}$ as in the statement of the lemma. It sufficess to show that $\textsc{Solve}^W(i) = s_{W,n}$ whenever

$$Q_{W,n} \cap W_{\geq n} = \emptyset. \tag{3.4.1}$$

Recall that, by the definition of the benign set from the algorithm $\textsc{Solve}$ (see line 4 in Algorithm 5), for any $\ell \in \mathbb{N}$:

$$B_{i,W,\ell} = \left\{s \in S_{i,W} \;\middle|\; Q\left(C^W(i,s)\right) \cap W_{\geq \ell} = \emptyset\right\},$$

where $S_{i,W} = \left\{s \in \{0,1\}^* \;\middle|\; |s| \leq p(|i|) \;\wedge\; C^W(i,s) = 1\right\}$. Let us denote the iteration in which $\textsc{Solve}^W(i)$ halts by t_W and the returned solution by s_W. Similarly, let $t_{W,n}$ denote the iteration in which $\textsc{Solve}^{W_{<n}}(i)$ halts and we already know that it returns $s_{W,n}$. We show a few observations about s_W, $s_{W,n}$, t_W and $t_{W,n}$ which, when combined together, give us that $s_W = s_{W,n}$.

$s_W \in B_{i,W,t_W}$ and $s_{W,n} \in B_{i,W_{<n},t_{W,n}}$: These containments follow trivially from the nature of the algorithm $\textsc{Solve}$. More specifically, if $\textsc{Solve}^W(i)$ halts in the t-th iteration then the set $B_{i,W,t}$ is non-empty and $\textsc{Solve}$ returns a solution from $B_{i,W,t}$ (see line 6 of Algorithm 5).

$t_{W,n} \leq n$: If we restrict the length of challenges to be strictly less than n as well as at least n we get empty relation, i.e., the set $(W_{<n})_{\geq n} = \emptyset$ and, thus, also its intersection with $Q\left(C^{W_{<n}}(i,s)\right)$ is empty for any $s \in \{0,1\}^*$. That is, $B_{i,W_{<n},t_{W,n}}$ contains all solutions of the instance i. By correctness (see Definition 3.3.1) there is at least one solution and, thus, $\textsc{Solve}^{W_{<n}}(i)$ halts (at the latest) in the n-th iteration of the while loop.

$s_{W,n} \in B_{i,W,t_{W,n}}$: First we observe that all queries $Q\left(C^{W_{<n}}(i, s_{W,n})\right) = Q_{W,n}$ are answered the same when running with respect to W instead of $W_{<n}$. That follows from the fact that $W_{<n}$ agrees with W on all queries (x, w)

where $|x| < n$ and the assumption that $Q_{W,n} \cap W_{\geq n} = \emptyset$ (see Equation (3.4.1)), which ensures that $W_{<n}$ agrees with W also on queries from $(x, w) \in Q_{W,n}$ where $|x| \geq n$. Thus, $s_{W,n}$ is a solution with respect to W (i.e., $C^W(i, s_{W,n}) = 1$) and, moreover, the set of queries also remains unchanged when C runs with respect to W instead of $W_{<n}$ that is $Q\left(C^{W_{<n}}(i, s_{W,n})\right) = Q_{W,n} = Q\left(C^W(i, s_{W,n})\right)$.

Since $t_{W,n} \leq n$ we have $W_{\geq t_{W,n}} = \left(\left(W_{\geq t_{W,n}} \cap W_{<n}\right) \cup W_{\geq n}\right)$. Using this equality and de Morgan rules we can rewrite the intersection from the definition of $B_{i,W,t_{W,n}}$ as follows:

$$\begin{aligned}
&Q\left(C^W(i, s_{W,n})\right) \cap W_{\geq t_{W,n}} \\
&\quad = Q\left(C^W(i, s_{W,n})\right) \cap \left(\left(W_{\geq t_{W,n}} \cap W_{<n}\right) \cup W_{\geq n}\right) \\
&\quad = \left(Q\left(C^W(i, s_{W,n})\right) \cap \left(W_{\geq t_{W,n}} \cap W_{<n}\right)\right) \cup \left(Q\left(C^W(i, s_{W,n})\right) \cap W_{\geq n}\right).
\end{aligned}$$

Since $W_{<n}$ agrees with W on all pairs $(x, w) \in \left(W_{\geq t_{W,n}} \cap W_{<n}\right)$ and $s_{W,n}$ is in $B_{i,W_{<n},t_{W,n}}$ the first intersection has to be empty. The second intersection follows from already shown equality $Q\left(C^W(i, s_{W,n})\right) = Q_{W,n}$ and the assumption that $Q_{W,n} \cap W_{\geq n} = \emptyset$ (see Equation (3.4.1)).

$t_W \leq t_{W,n}$: From $s_{W,n} \in B_{i,W,t_{W,n}}$ we know that $\textsc{Solve}^W(i)$ must return (at the latest) in the $t_{W,n}$-th iteration of the while loop as the “benign” set contains at least $s_{W,n}$ in this iteration.

$s_W \in B_{i,W_{<n},t_W}$: We have already proved that $s_W \in B_{i,W,t_W}$ and in particular $Q\left(C^W(i, s_W)\right) \cap W_{\geq t_W} = \emptyset$. As $t_w \leq t_{w,n} \leq n$ and $W_{<n}$ agrees with W on all queries (x, w) where $|x| < n$, all queries are answered the same by oracle $W_{<n}$ as by W. Thus, s_W is a solution with respect to $W_{<n}$ too and, moreover, $Q\left(C^W(i, s_W)\right) = Q\left(C^{W_{<n}}(i, s_W)\right)$.

Since $(W_{<n})_{\geq t_W} \subseteq W_{\geq t_W}$ and queries $Q\left(C^{W_{<n}}(i, s_W)\right) = Q\left(C^W(i, s_W)\right)$ the intersection

$$Q\left(C^{W_{<n}}(i, s_W)\right) \cap (W_{<n})_{\geq t_W} \subseteq Q\left(C^W(i, s_W)\right) \cap W_{\geq t_W} = \emptyset$$

and, thus, $s_W \in B_{i,W_{<n},t_W}$.

$t_W = t_{W,n}$: The fact that $s_W \in B_{i,W_{<n},t_W}$ gives us that $\textsc{Solve}^{W_{<n}}(i)$ returns in the iteration t_W or earlier. That is, $t_{W,n} \leq t_W$. Combining with the already proven inequality $t_W \leq t_{W,n}$ we get the equality.

$s_W = s_{W,n}$: Let t denote the iteration in which both algorithms return, that is $t = t_W = t_{W,n}$. If we combine the above observations we get:

$$s_W, s_{W,n} \in B_{i,W,t} \text{ as well as } s_W, s_{W,n} \in B_{i,W_{<n},t}$$

Recall that $\textsc{Solve}$ returns the lexicographically smallest solution (see line 6 of Algorithm 5). As $\textsc{Solve}^W(i)$ returns s_W in the t-th iteration and both $s_W, s_{W,n} \in B_{i,W,t}$ we get that $s_W \leq_{\text{lex}} s_{W,n}$. Similarly since $\textsc{Solve}^{W_{<n}}(i)$ returns $s_{W,n}$ in the t-th iteration and both $s_W, s_{W,n} \in B_{i,W_{<n},t}$ we get that $s_{W,n} \leq_{\text{lex}} s_W$. Combining these inequalities we get that $s_W = s_{W,n}$.

The last observation concludes the proof because we have shown that both algorithms return the same solution $s_W = s_{W,n}$ whenever $Q_{s,n} \cap W_{\geq n} = \emptyset$ and we have upper bounded the probability of this event by $|Q_{W,n}| / 2^{n+1}$. $\square$

Now we are ready to upper bound the success probability of the security reduction.

Lemma 3.3.6. (Restated) *Let (R, T_R, C, T_C, p) be a fully black-box construction of a worst-case hard TFNP problem from an average-case hard UP problem and*

$$q_C(n) = \max \left| Q\left(C^{W_{<n}}(i,s)\right)\right|,$$

where the maximum is over $W \in \mathsf{UniqWitness}$, $x \in \{0,1\}^n$, $r \in \{0,1\}^{T_R(n)}$, $i \in Q_{\text{Solve}}\left(R^W(x;r)\right)$ and $s \in \{0,1\}^*$ such that $|s| \leq p(|i|)$. Then, for any $n \in \mathbb{N}$ there exists $W \in \mathsf{UniqWitness}$ such that:

$$\Pr_{\substack{x \leftarrow \{0,1\}^n, \\ r \leftarrow \{0,1\}^{T_R(n)}}} \left[R^{W,\text{Solve}}(x;r) = \chi_W(x)\right] \leq \frac{1}{2} + \frac{T_R(n)\left(q_C(n)+1\right)}{2^{n+1}}.$$

Proof. Let us fix any construction (R, T_R, C, T_C, p) and a number $n \in \mathbb{N}$ as in the statement of the lemma. We first upper bound the probability that $R^{W,\text{Solve}}(x;r)$ returns $\chi_W(x)$, with respect to internal randomness r of the reduction R and the choices of both $x \leftarrow \{0,1\}^n$ as well as $W \leftarrow \mathtt{RL}$. That is, we prove the following bound rather than the one in the statement of the lemma.

$$\Pr_{\substack{x \leftarrow \{0,1\}^n, W \leftarrow \mathtt{RL}, \\ r \leftarrow \{0,1\}^{T_R(n)}}} \left[R^{W,\text{Solve}}(x;r) = \chi_W(x)\right] \leq \frac{1}{2} + \frac{T_R(n)\left(q_C(n)+1\right)}{2^{n+1}}.$$

Note that from this bound the lemma follows as by averaging there exists W such that the same bound holds when the probability is taken only over the choice of x.

To prove the above inequality we consider the following experiment:

1. We choose an auxiliary language $W' \leftarrow \mathtt{RL}_{<n}$, that is we sample $W'' \leftarrow \mathtt{RL}$ and set $W' = W''_{<n}$.
2. We run the security reduction R on our challenge x with respect to W' and R outputs a bit b.
3. After R outputs we sample the language $W \leftarrow \mathtt{RL}$ in such a way that $W_{<n} = W'$.

Observe that the distribution on languages $W \in \mathsf{UniqWitness}$ in the experiment is the same as if we choose $W \leftarrow \mathtt{RL}$ directly. We argue that with high probability the view of R when running with respect to W is the same as when running with respect to W'. Thus, the answer of R on the challenge x is the same.

Recall that when we sample $W \leftarrow \mathtt{RL}$ we add $(x, \cdot)$ to W with probability exactly $1/2$. Thus, once we fix a bit b_x, the probability over the choice of $W \leftarrow \mathtt{RL}$ that $\chi_W(x) = b_x$ is exactly $1/2$. This gives us that the probability that the reduction returns $\chi_W(x)$ even though it runs with respect to W' instead of W as follows:

$$\Pr_{\substack{x \leftarrow \{0,1\}^n, W \leftarrow \mathtt{RL}, \\ r \in \{0,1\}^{T_R(n)}}} \left[R^{W', \textsc{Solve}}(x; r) = \chi_W(x) \;\middle|\; W_{<n} = W' \right] = \frac{1}{2}$$

It suffices to show that the views of $R^{W',\textsc{Solve}}$ and $R^{W,\textsc{Solve}}$ are the same with high probability. To this end we need to show that in both cases all queries to Solve as well as to W are answered the same. First we show that the probability that a direct query should be answered differently is small. Observe that as $W' \subseteq W$ this is equivalent to upper bounding the following probability for any $x \in \{0,1\}^n$ and any $r \in \{0,1\}^{T_R(n)}$:

$$\Pr_{W \leftarrow \mathtt{RL}} \left[Q^{\mathrm{dir}}_{W'} \left(R^{W',\textsc{Solve}}(x;r) \right) \cap (W \setminus W') \neq \emptyset \;\middle|\; W_{<n} = W' \right].$$

By the definition of RL (see Definition 3.2.3), any pair (y, u) is in the relation W with probability $1/\left(2^{|y|+1}\right)$ if $|y| = |u|$ and with probability zero otherwise. Thus, we may bound the above inequality using union bound as follows:

$$\begin{aligned} \Pr_{W \leftarrow \mathtt{RL}} & \left[Q^{\mathrm{dir}}_{W'} \left(R^{W',\textsc{Solve}}(x;r) \right) \cap (W \setminus W') \neq \emptyset \;\middle|\; W_{<n} = W' \right] \\ & \leq \sum_{(y,u) \in Q^{\mathrm{dir}}_{W'} \left(R^{W',\textsc{Solve}}(x;r) \right)} \Pr_{W \leftarrow \mathtt{RL}} \left[(y,u) \in W \setminus W' \mid W_{<n} = W' \right] \\ & \leq \sum_{\substack{(y,u) \in Q^{\mathrm{dir}}_{W'} \left(R^{W',\textsc{Solve}}(x;r) \right), \\ |y| \geq n}} \frac{1}{2^{|y|+1}} \\ & \leq \frac{\left| Q^{\mathrm{dir}}_{W'} \left(R^{W',\textsc{Solve}}(x;r) \right) \right|}{2^{n+1}}, \end{aligned}$$

where the second inequality follows from the condition that $W_{<n} = W'$ and, thus, $W \setminus W'$ contains only pairs (y, u) such that $|y| \geq n$.

Now we bound the probability that any query to the oracle Solve is answered differently when the security reduction runs with respect to W' instead of W. That is, we bound the following probability for any $x \in \{0,1\}^n$ and any $r \in \{0,1\}^{T_R(n)}$:

$$\Pr_{W \leftarrow \mathtt{RL}} \left[\exists i \in Q_{\textsc{Solve}} \left(R^{W',\textsc{Solve}}(x;r) \right) : \textsc{Solve}^W(i) \neq \textsc{Solve}^{W'}(i) \;\middle|\; W_{<n} = W' \right].$$

By Claim 3.4.1 the probability that Solve answers any query i differently can be bounded by:

$$\Pr_{W \leftarrow \mathtt{RL}} \left[\textsc{Solve}^W(i) \neq \textsc{Solve}^{W'}(i) \right] \leq \frac{q}{2^{n+1}}.$$

where q denotes the number of queries $C^{W'}$ makes when running on the instance i and the solution $s = \textsc{Solve}^{W'}(i)$. By assumptions of the lemma we can upper bound q by $q_C(n)$. Thus, using union bound we get

$$\begin{aligned}
&\Pr_{W\leftarrow \mathtt{RL}}\left[\exists i \in Q_{\textsc{Solve}}\left(R^{W',\textsc{Solve}}(x;r)\right) : \textsc{Solve}^{W}(i) \neq \textsc{Solve}^{W'}(i) \;\middle|\; W_{<n} = W'\right] \\
&\quad\leq \sum_{i\in Q_{\textsc{Solve}}\left(R^{W',\textsc{Solve}}(x;r)\right)} \Pr_{W\leftarrow \mathtt{RL}}\left[\textsc{Solve}^{W}(i) \neq \textsc{Solve}^{W'}(i) \;\middle|\; W_{<n} = W'\right] \\
&\quad\leq \sum_{i\in Q_{\textsc{Solve}}\left(R^{W',\textsc{Solve}}(x;r)\right)} \frac{q_C(n)}{2^{n+1}} \\
&\quad\leq \left|Q_{\textsc{Solve}}\left(R^{W',\textsc{Solve}}(x;r)\right)\right| \frac{q_C(n)}{2^{n+1}}
\end{aligned}$$

If all queries to both oracles Solve and W are answered the same for W as well as for W', the returned bit is the same too. Thus, by adding all the inequalities proven above (and taking the probability also with respect to the challenge x and the randomness r) we get that:

$$\begin{aligned}
&\Pr_{\substack{x\leftarrow\{0,1\}^n,\\ r\leftarrow\{0,1\}^{T_R(n)},\\ W\leftarrow \mathtt{RL}}}\left[R^{W,\textsc{Solve}}(x;r) = \chi_W(x)\right] \\
&\quad\leq \frac{1}{2} + \max_{\substack{x\in\{0,1\}^n,\\ r\in\{0,1\}^{T_R(n)}}} \frac{\left|Q^{\mathrm{dir}}_{W'}\left(R^{W',\textsc{Solve}}(x;r)\right)\right| + \left|Q_{\textsc{Solve}}\left(R^{W',\textsc{Solve}}(x;r)\right)\right| q_C(n)}{2^{n+1}} \\
&\quad\leq \frac{1}{2} + \frac{T_R(n)\left(q_C(n)+1\right)}{2^{n+1}}.
\end{aligned}$$

where we bound the number of direct queries to oracle W as well as the number of queries to oracle Solve by the running time T_R of the reduction. Now by averaging, there is a choice of $W \in \mathsf{UniqWitness}$ such that the above bound holds even when the probability is only over the choice of the challenge x and the randomness r, which concludes the proof. □

4. Worst-Case Hardness in TFNP and Injective One-Way Functions

In this chapter we show impossibility of *simple* construction of a worst-case hard (and thus also average-case hard) TFNP problem from (injective) one-way functions. The proof follows a similar blueprint as the proof in Chapter 3. But as injective one-way functions (which imply average-case hardness in NP) are much more structured than any average-case hard problem we are not able to prove the impossibility in full. Instead we rule out only simple constructions.

Simple constructions. We recall the details of the construction of a hard-on-average distribution in TFNP from one-way permutations (see Papadimitriou [1994]) to highlight the restrictions on the type of reductions considered in our results.

Consider the total search problem `Pigeon` (see Definition 1.1.3) which is complete for a subclass of TFNP known as PPP, and Papadimitriou [1994] gave the following construction of a hard `Pigeon` problem from one-way permutations (discussed already in Section 3.3 in paragraph "Example fully black-box construction"). Given a (one-way) permutation $f\colon \{0,1\}^n \to \{0,1\}^n$ and a challenge $y \in \{0,1\}^n$ for inversion under f, the reduction algorithm defines an instance of `Pigeon` by the oracle-aided circuit P_y^f computing the function $P_y^f(x) = f(x) \oplus y$. It is not hard to see that the instance of `Pigeon` P_y^f has a unique solution corresponding to the preimage of y under f and, therefore, any algorithm solving it breaks the one-wayness of f.

Note that the above construction of a hard (on average) TFNP problem is extremely simple in various aspects:

- The construction is *fully black-box*, i.e., the `Pigeon` instance can be implemented via an oracle-aided circuit treating the one-way permutation as a black-box and the reduction inverts when given oracle access to an arbitrary solver for `Pigeon`.

- The reduction is *many-one*, i.e., a single call to a `Pigeon`-solving oracle suffices for finding the preimage of y.

- The reduction is *f-oblivious*, i.e., the oracle-aided circuit P_y^f defining the `Pigeon` instance depends only on the challenge y and does not depend on the one-way permutation f in the sense that P_y^f itself can be fully specified without querying f. In other words, given the challenge y, the instance P_y^f submitted to the `Pigeon` oracle by the reduction is, as an oracle-aided circuit, identical for all permutations f.

- The reduction is *deterministic*, i.e., it simply passes y to specify the `Pigeon` instance.

It is known that such a fully black-box construction of `Pigeon` with a deterministic f-oblivious many-one reduction exists also assuming collision-resistant hash functions exist (folklore). Specifically, for any hash function $h\colon \{0,1\}^n \to$

$\{0,1\}^{n-1}$ from the collision-resistant family, the `Pigeon` instance is defined as $P^h(x)$ which returns $h(x)$ concatenated with 1. Since P^h concatenates the value $h(x)$ with 1 for any input x, it never maps to the all-zero string and, therefore, has the same collisions as h. Note that, unlike in the above construction from one-way permutations, the instances resulting from collision-resistant hash functions do not have a unique solution. In fact, there are always at least 2^{n-1} non-trivial collisions (even in two-to-one functions where each $y \in \{0,1\}^{n-1}$ has exactly two preimages) and this structural property is inherent as shown by Rosen et al. [2017]. Importantly, the property of having nearly exponentially many solutions is not in contradiction with the resulting distribution being hard-on-average. Currently, there is no actual evidence suggesting that average-case hardness in TFNP cannot be based on the existence of injective one-way functions.

The above constructions motivate us to study whether there exist such "simple" constructions of an average-case hard TFNP problem under weaker cryptographic assumptions such as the existence of injective one-way functions, and we answer this question in negative (see Section 4.3.2, Theorem 4.3.4 for the formal statement). Even though restricted, our results are the first step towards the full-fledged black-box separation of TFNP and (injective) one-way functions. We note that the full-fledged separation would necessarily subsume the known separation of collision-resistant hash functions and injective one-way functions (see Simon [1998]), for which, despite the recent progress, there are only non-trivial proofs (see Matsuda and Matsuura [2011], Haitner et al. [2015], Bitansky and Degwekar [2019]).

We explain the differences between the proof presented here and the proof for impossibility of constructions from average-case hard problem in NP (i.e., the result presented in Chapter 3) in Section 4.1. Then we introduce the notation and definitions needed for the proof in Section 4.2. In Section 4.3 we present the formal definition of a "simple" fully black-box constructions as well as the proof of their impossibility for the deterministic f-oblivious many-one reductions. The proofs of all the lemmata are given in Section 4.4. Finally in Section 4.5 we extend our results from Section 4.3 to randomized and non-adaptive constructions.

4.1 Our Techniques

Similarly as in Chapter 3 we show the impossibility using the so-called two oracle technique by Hsiao and Reyzin [2004]. Here the oracle O comprises of a random injective function (which yields an injective one-way function) and a procedure Solve which provides a solution for any instance of a TFNP problem. To argue that Solve does not help inverting injective one-way function we use the reconstruction paradigm of Gennaro and Trevisan [2000]. Here we explain the structural insights that are key to our separation and guided us in the design of our oracle Solve.

The existence of a "useless" solution. At the core of our negative result is a new structural insight about TFNP instances constructed from (injective) one-way functions. Observe that any one-way function gives rise to a search problem with a hard-on-average distribution which is total over its support (but all instances outside its support have no solution). Specifically, for any one-way function

$f : \{0,1\}^n \to \{0,1\}^{n+1}$, an instance is any $y \in \{0,1\}^{n+1}$ and the solution for y is any $x \in \{0,1\}^n$ such that $f(x) = y$. The hard-on-average distribution then corresponds to sampling x uniformly from $\{0,1\}^n$ and outputting the instance $y = f(x)$ (as in the standard security experiment for one-way functions). When attempting to construct a hard search problem which is truly total and has a solution for all instances (not only for the support of the hard distribution), one has to face the frustrating obstacle in the form of "useless" solutions which do not help the reduction in inverting its challenge y. Note that, as the resulting TFNP problem must be total for all oracles f, there must exist a solution even for oracles with no preimage for the challenge y. By a simple probabilistic argument, it follows that for a random oracle f and a random challenge y, with overwhelming probability, there exists a solution to any TFNP instance which does not query a preimage of y, i.e., a "useless" solution from the perspective of the reduction.[1]

Thus, demonstrating a black-box separation would be straightforward if the TFNP solver knew which challenge y is the reduction attempting to invert. Our solver would simply output such a "useless" solution and we could argue via the reconstruction paradigm that no reduction can succeed in inverting y given access to our solver. In this work, we show that it is possible to construct a TFNP solver which returns such a "useless" solution with overwhelming probability even though the solver *does not know* the input challenge of the reduction.

Reduction-specific Solve. Note that a reduction in a fully black-box construction must succeed in breaking the one-way function f when given access to *any* oracle SOLVE (see Definition 4.3.1). In other words, to rule out the existence of constructions with a fully black-box reduction, it is sufficient to show that for every reduction there exists a SOLVE which is not helpful in inverting; in particular, SOLVE may depend on the reduction. To enable SOLVE to answer the reduction's query with a "useless" solution with overwhelming probability, we take exactly this approach and construct a reduction-specific SOLVE for any construction of a TFNP problem from injective one-way functions. We significantly differ in this aspect from the previous works which relied on the reconstruction paradigm of Gennaro and Trevisan [2000], e.g., the works which employed the collision-finding oracle of Simon [1998] (see Haitner et al. [2015], Pass and Venkitasubramaniam [2010], Rosen and Segev [2010], Brakerski et al. [2011]). We note that the possibility of designing a breaker oracle which depends on the fully black-box construction was exploited already by Gertner et al. [2001], who considered SOLVE which depends on the implementation rather than the reduction algorithm (as in our case). That is, to rule out the construction of a primitive P from a primitive Q, they considered an oracle SOLVE that depends on the implementation C^Q of the primitive P, whereas in our case SOLVE depends on the reduction algorithm R that is supposed to break Q given access to an algorithm that breaks C^Q. The possibility of proving black-box separations via reduction-specific oracles was also observed in the work of Hsiao and Reyzin [2004] who, nevertheless, did not have to leverage this observation in their proofs.

[1] Note that the above argument fails in the case of one-way permutations, where the challenge $y \in \{0,1\}^n$ is in the image for any permutation $f\colon \{0,1\}^n \to \{0,1\}^n$. The construction of a TFNP problem then simply does not have to deal with the case when the challenge y is not in the image of f, and it can ensure that every solution is useful for inverting the challenge y.

On a high level, given that Solve can use the code of the reduction R, Solve can simulate R on all possible challenges y to identify the set of challenges on which R queries the present instance that Solve needs to solve. As we show, the solution then can be chosen adversarially so that it avoids such solutions of interest to the reduction. To turn this intuition into a formal proof, one needs to show that our Solve indeed does not help in inverting injective one-way functions and we do so along the lines of the reconstruction paradigm of Gennaro and Trevisan [2000].

Applying the compression argument. Two important subtleties arise in the proof when we try to turn the reduction into a pair of compression and decompression algorithms, which we explain next. First, the reconstruction paradigm is conventionally applied to random permutations (as in Gennaro and Trevisan [2000], Haitner et al. [2015]), whereas the reduction R and the algorithm Solve are designed for random injective functions. The natural approach is to simply proceed with the same style of proof even in our setting. Specifically, one would presume that a similar incompressibility argument can be leveraged if we manage to somehow encode the image of the random injective function. While this intuition is correct in the sense that it allows *correct* compression and reconstruction, it turns out that the space required to encode the image is too prohibitive for reaching the desired contradiction with known information-theoretic lower bounds on the expected length of encoding for a random injective function. To resolve this issue, we construct compressor and decompressor algorithms for a random permutation, but we equip the algorithms with *shared randomness* in the form of a random injective function $h\colon \{0,1\}^n \to \{0,1\}^{n+1}$ independent of the random permutation $\pi\colon \{0,1\}^n \to \{0,1\}^n$ to be compressed. Whenever the compressor and decompressor need to provide the reduction or Solve with access to the injective function $f\colon \{0,1\}^n \to \{0,1\}^{n+1}$, they compose the permutation π with the shared injective function h and then pass off the composed injective function $f = h \circ \pi$ to the reduction. With this modification, we are able to show that any reduction which succeeds in inverting injective one-way functions given access to our Solve can be used to compress a random permutation on $\{0,1\}^n$ below a standard information-theoretic lower bound on the size of a prefix-free encoding of such random variable. We note that this is reminiscent of the approach used in Matsuda and Matsuura [2011] for separating *injective* one-way functions from one-way permutations.

Second, we cannot employ the actual oracle Solve in our compression and decompression algorithms: even though we can use the reduction when compressing and decompressing the random permutation, we must be able to *consistently* simulate Solve without accessing the whole permutation. In general, the choice of the "breaker" oracle that can be simulated efficiently without too many queries to the permutation is a crucial part of the whole proof, and, a priori, it is unclear how to design a TFNP solver which also has such a property. Nevertheless, we show that there exists a Solve which can be efficiently simulated given only (sufficiently small) partial information about the permutation.

f-oblivious reductions. As our Solve simulates the reduction on possible challenges y, we need for technical reasons that the reduction is f-oblivious (namely, for correctness of our encoding and decoding algorithms). However, we believe that f-obliviousness is not overly restrictive as it is a natural property of security reductions. Besides the two fully black-box constructions of `Pigeon` with f-oblivious reductions (described already at the beginning of Chapter 4), f-oblivious security reductions appear also in the cryptographic literature. See for example the standard security reduction in the Goldreich-Levin theorem (see Goldreich and Levin [1989]) establishing the existence of hard-core predicate for any one-way function (note that this particular security reduction is also non-adaptive). An orthogonal notion of a π-oblivious construction appears in the work of Wee [2007]. However, it is the implementation of the constructed primitive which is "oblivious" to the one-way permutation π in his work.

4.2 Preliminaries

Unless stated otherwise, all logarithms are base two. For $X \subseteq \{0,1\}^*$, we use $\overline{X}$ to denote the set $\{0,1\}^* \setminus X$. For strings $x, y \in \{0,1\}^*$, we use $x <_{\text{lex}} y$ or $y >_{\text{lex}} x$ to denote that x is lexicographically strictly smaller than y.

Notation 4.2.1 (Functions)**.** *Let $X, Y \subseteq \{0,1\}^*$, $f\colon X \to Y$ be a function and $X' \subseteq X$ be a set.*

1. *$f \restriction X'$ denotes the* restriction of f on X'*, i.e., the function $f'\colon X' \to Y$ such that $\forall x \in X'\colon f'(x) = f(x)$.*
2. *$Dom(f)$ denotes the* domain of f*, i.e., the set X.*
3. *$Im(f)$ denotes the* image of f*, i.e., the set $\{f(x) \mid x \in X\} \subseteq Y$.*
4. *$f[X']$ denotes the* image of the restriction of f on X'*, that is the set $Im(f \restriction X')$.*

Notation 4.2.2 (Injective functions)**.** *We denote by Inj_n^m the set of all injective functions from $\{0,1\}^n$ to $\{0,1\}^m$. For the special case when $n = m$ we get the set of all permutations on $\{0,1\}^n$.*

The set Inj is the set of all functions $f\colon \{0,1\}^* \to \{0,1\}^*$, such that f can be interpreted as a sequence $f = \left\{f_n \mid f_n \in Inj_n^{\mu(n)}\right\}_{n\in\mathbb{N}}$ of injective functions, where $\mu\colon \mathbb{N} \to \mathbb{N}$ is an injective function such that for all $n \in \mathbb{N}\colon \mu(n) > n$ and $\mu(n) \leq 100n^{\log n}$.

We say that the function μ is the type *of f and we define the corresponding* type operator $\tau\colon Inj \to (\mathbb{N} \to \mathbb{N})$ *such that $\tau(f) = \mu$.*

We denote the set of all possible types by T*, i.e.,*

$$\mathrm{T} = \{\mu\colon \mathbb{N} \to \mathbb{N} \mid \exists f \in Inj \text{ such that } \tau(f) = \mu\}.$$

Through the paper f_n denotes the function $f \restriction \{0,1\}^n$ (i.e., restriction of f to the domain $\{0,1\}^n$.), where $f \in Inj$.

Finally let $\mu \in \mathrm{T}$ be any type, then by Inj_μ we denote the set of all injective functions from Inj of type μ.

Notation 4.2.3 (Sampling)**.** *Let A be any finite set we use $x \leftarrow A$ to denote that x is taken uniformly at random from the set A.*

Moreover, for any type $\mu \in \mathrm{T}$ we use $f \leftarrow \mathit{Inj}_\mu$ to denote that functions $\{f_n = f \restriction \{0,1\}^n\}_{n\in\mathbb{N}}$ are chosen independently[2] *and uniformly at random from all functions from $\mathit{Inj}_n^{\mu(n)}$.*

In our proofs, we often compose a function defined on all binary strings with a function defined only for binary strings of certain length; namely, we often want to compose a function from Inj with a permutation of n-bit strings. The desired resulting function should always be a function from all binary strings. For the ease of exposition, we extend the standard notation for function composition as follows.

Notation 4.2.4 (Function composition)**.** *Let X, Y, Z be any sets such that $X \subseteq Y$ and let $f\colon X \to Y$ and $g\colon Y \to Z$. We define the function $g \circ f\colon Y \to Z$ as:*

$$(g \circ f)(x) = \begin{cases} g(f(x)) & \text{if } x \in X, \\ g(x) & \text{if } x \in Y \setminus X. \end{cases}$$

Finally, we recall some basic information-theoretic results about prefix-free codes.

Definition 4.2.5 (Prefix-free code)**.** *A set of code-words $C \subseteq \{0,1\}^*$ is a prefix-free code if there are no two distinct $c_1, c_2 \in C$ such that c_1 is a prefix (initial segment) of c_2, i.e., for any two distinct $c_1, c_2 \in C$ there exists $0 \leq j < \min(|c_1|, |c_2|)$ such that $(c_1)_j \neq (c_2)_j$.*

Proposition 4.2.6 (Theorem 5.3.1 in Cover and Thomas [2012])**.** *The expected length L of any prefix-free binary code for a random variable X is greater than or equal to the entropy $H(X)$.*

Corollary 4.2.7. *To encode a uniformly random permutation $\pi \in \mathit{Inj}_n^n$ using prefix-free encoding it takes at least $\log(2^n!)$ bits in expectation.*

Proof. The entropy of a uniformly randomly chosen permutation from Inj_n^n is $\log(2^n!)$ as we choose uniformly at random from $2^n!$ distinct permutations. By Proposition 4.2.6, we get that the expected size of the encoding is at least $\log(2^n!)$. □

4.3 Separating TFNP and Injective OWF

4.3.1 Fully Black-Box Construction of Hard TFNP Problem from Injective OWF

Below, we give a definition of fully black-box construction of a (worst-case) hard TFNP problem from an injective one-way function.

[2] Note that, since μ is injective, any $f \leftarrow \mathrm{Inj}_\mu$ satisfies $f \in \mathrm{Inj}_\mu$.

Definition 4.3.1 (Fully black-box construction of a worst-case hard TFNP problem from `injective-OWF`)**.** *Let $\mu \in \mathrm{T}$ be any type. A* fully black-box construction *of a worst-case hard* TFNP *problem from* injective one-way function *of type μ is a tuple (R, T_R, C, T_C, p) of oracle-aided algorithms R, C, functions T_R, T_C, and a polynomial p satisfying the following properties:*

1. *R **and** C **halt on all inputs:** For all $f \in Inj_\mu$, $n \in \mathbb{N}$, and $y, i, s \in \{0,1\}^*$, the algorithm $R^f(1^n, y)$ halts in time $T_R(n + |y|)$, and the algorithm $C^f(i, s)$ halts in time $T_C(|i| + |s|)$.*

2. ***Correctness:** For all $f \in Inj_\mu$ and for all $i \in \{0,1\}^*$, there exists $s \in \{0,1\}^*$ such that $|s| \leq p(|i|)$ and $C^f(i, s) = 1$, i.e., for any instance of the TFNP problem there exists a solution of polynomial length.*[3]

3. ***Fully black-box proof of security:** There exists a polynomial p' such that for all $f \in Inj_\mu$ and any oracle-aided algorithm $\textsc{Solve}$, if*

$$\forall i \in \{0,1\}^* : \textsc{Solve}^f(i) \textit{ returns } s \textit{ such that } C^f(i, s) = 1$$

then for infinitely many $n \in \mathbb{N}$,

$$\Pr_{x \leftarrow \{0,1\}^n}\left[R^{f,\textsc{Solve}}(1^n, f(x)) = x\right] \geq \frac{1}{p'(n)} .$$

Note that all the remarks mentioned in Section 3.3 apply to this definition too and are important for our proof. But we point out one more important property of the above definition on which we crucially rely in our proof.

Reduction-specific Solve. Let us emphasize the order of quantifiers restricting the security reduction in Definition 4.3.1:

$$\exists\,(R, T_R, C, T_C, p)\ \forall f\ \forall \textsc{Solve}:$$
$$\textsc{Solve}^f \text{ solves the } \mathsf{TFNP} \text{ problem } C \implies R^{f,\textsc{Solve}} \text{ inverts } f.$$

Importantly, the reduction must invert when given access to *any* oracle $\textsc{Solve}$. As a consequence, in order to establish a separation, the above statement is negated and it suffices to show that for every reduction there exists a solver (see proof of [Hsiao and Reyzin, 2004, Proposition 1] for more details). Thus, in the proof of an oracle separation, the oracle $\textsc{Solve}$ may even depend on the behaviour of the reduction R, and, in particular, $\textsc{Solve}$ can simulate the security reduction R on an arbitrary input. We exploit these properties in establishing our results.

Direct and indirect queries to f. The security reduction R can learn something about f in various ways. It may query f directly or the information might be deduced from the solution of some queried instance of the TFNP problem returned by $\textsc{Solve}$. We introduce the following notation in order to distinguish where queries originate (similarly as in Notation 3.3.4), which allows us to reason about the view the security reduction has over the function f in our proof of Theorem 4.3.4.

[3]The Correctness corresponds to the totality of the TFNP problem, see Definition 1.1.1.

Notation 4.3.2 (Query sets Q)**.** *We distinguish the following sets of oracle queries depending on where these queries originated and which oracle is queried.*

- *Let $Q\left(C^f(i,s)\right)$ denote the set of all preimages $x \in \{0,1\}^*$ on which the oracle f has been queried by C running on an input (i,s).*
- *Let $Q_{\textsc{Solve}}\left(R^{f,\textsc{Solve}}(1^n,y)\right)$ denote the set of all instances $i \in \{0,1\}^*$ on which the oracle* Solve *has been queried by R running on a security parameter n and challenge y.*
- *Let $Q_f^{dir}\left(R^{f,\textsc{Solve}}(1^n,y)\right)$ denote the set of preimages $x \in \{0,1\}^*$ on which the oracle f has been queried by R running on an input y and security parameter n.*
- *Let $Q_f^{indir}\left(R^{f,\textsc{Solve}}(1^n,y)\right)$ denote the set of all preimages $x \in \{0,1\}^*$ on which the oracle f has been queried indirectly, i.e., it has been queried by C running on an input (i,s) where $i \in Q_{\textsc{Solve}}\left(R^{f,\textsc{Solve}}(1^n,y)\right)$ and $s = \textsc{Solve}^f(i)$.*
- *Let $Q_f\left(R^{f,\textsc{Solve}}(1^n,y)\right) = Q_f^{dir}\left(R^{f,\textsc{Solve}}(1^n,y)\right) \cup Q_f^{indir}\left(R^{f,\textsc{Solve}}(1^n,y)\right)$.*

Note that these sets may not be disjoint. When f is a partial function (i.e., when f is not defined on all inputs) the query set contains all queries queried up to the point of the first undefined answer and the query with the undefined answer is included as well.

Restrictions on the power of the reduction. We consider various restricted classes of security reductions as defined below.

Definition 4.3.3 (Properties of security reductions)**.** *Let μ be any type. We distinguish the following properties of a security reduction R from* TFNP *to injective one-way functions of type μ.*

The security reduction can be either deterministic or randomized. For a randomized security reduction, we extend the input of R to a triple $(1^n, y; r)$, where the meaning of n, resp. y, remains unchanged (i.e., n is the security parameter, y is the challenge), and $r \in \{0,1\}^*$ is the randomness of the security reduction. The success probability of a randomized security reduction is taken not only over the choice of the challenge x but also over the choice of its randomness r similarly as in Definition 3.3.1.

The security reduction R is many-one[4] *if for all $f \in Inj_\mu$, for any oracle* Solve *and for all $y \in \{0,1\}^*$, $R^{f,\textsc{Solve}}(1^n,y)$ makes a single query to the oracle* Solve*.*

The security reduction R is non-adaptive *if for all $f \in Inj_\mu$, for any oracle* Solve *and for all $y \in \{0,1\}^*$, all the queries of $R^{f,\textsc{Solve}}(1^n,y)$ to the oracle* Solve *are submitted in parallel (i.e., the queries to* Solve *do not depend on the answers received from* Solve*).*

[4]Note that this may slightly differ from the standard definition of many-one reduction from complexity theory, in particular the one used for decision problems, where the algorithm returns the same output as it gets from the oracle query. Whereas we allow the security reduction to post-process the output from the Solve oracle.

*The security reduction R is f-*oblivious *if for all challenges $y \in \{0,1\}^*$, for any oracle* Solve, *and any pair of functions $f, f' \in Inj_\mu$, the distributions of queries $Q_{\textsc{Solve}}\left(R^{f,\textsc{Solve}}(1^n, y)\right)$ and $Q_{\textsc{Solve}}\left(R^{f',\textsc{Solve}}(1^n, y)\right)$ are identical (i.e., the queries to* Solve *depend only on the input y and are independent of the oracle f).* [5]

We say that a fully black-box construction (R, T_R, C, T_C, p) of a worst-case hard TFNP *problem from injective one-way functions is deterministic (resp. randomized), many-one (resp. non-adaptive) and f-oblivious if the corresponding security reduction R is deterministic (resp. randomized), many-one (resp. non-adaptive) and f-oblivious.*

4.3.2 Impossibility for a Deterministic f-Oblivious Many-One Reduction

In this section, we show that there is no fully black-box construction of a hard TFNP problem from injective one-way functions with a deterministic f-oblivious many-one reduction. The proof of this result is already non-trivial and highlights our main technical contributions. In Section 4.5, we explain how to extend this result to rule out fully black-box constructions even with a *randomized* f-oblivious *non-adaptive* reduction.

Theorem 4.3.4. *Let $\mu \in \mathrm{T}$ be any type. There is no fully black-box construction (R, T_R, C, T_C, p) of a worst-case hard* TFNP *problem from injective one-way functions of type μ with a deterministic f-oblivious many-one reduction with success probability at least $2^{-0.1n}$ such that both running times $T_R, T_C \in O\left(\ell^{polylog(\ell)}\right)$, where n is the security parameter and ℓ corresponds to the length of the input of R, resp. C.*

In the above theorem, the running time of both R and C is restricted to quasi-polynomial. Note that the standard notion of cryptographic constructions requires R, C to run in polynomial time in order to be considered efficient. We are ruling out a broader class of potentially less efficient reductions.

The proof of Theorem 4.3.4 uses, on a high level, a similar template as other black-box separations in the literature. That is, we design an oracle O relative to which (injective) one-way functions exist but TFNP is broken (even in the worst case). We follow the two-oracle approach of Hsiao and Reyzin [2004], and, therefore, our oracle $O = (f, \textsc{Solve})$ consist of:

1. $f \in \mathrm{Inj}_\mu$: a sequence of random injective functions which implements injective one-way functions; and
2. Solve: a reduction-specific oracle that solves TFNP instances.

It is a well-established result that a random injective function is one-way (see, e.g., Claim 5.3 in Rosen et al. [2017] for the more general case of random functions). The bulk of technical work revolves around showing that f remains one-way

[5]Alternatively the reader may think of the f-oblivious reduction as consisting of two algorithms $R = (R_1, R_2)$. Where the first one takes as input the security parameter n, the challenge y and may query only the oracle Solve. In contrast the algorithm R_2 is allowed to query only the oracle f and its input consist n, y and the output of R_1.

Algorithm 6: The oracle SOLVE.

```
Hardwired      : a deterministic $f$-oblivious many-one fully black-box
                 construction $(R, T_R, C, T_C, p)$ of a worst-case hard TFNP
                 problem from injective-OWF of type $\mu$
Oracle access: an injective function $f = \{f_n\}_{n\in\mathbb{N}} \in \mathrm{Inj}_\mu$
Input          : an instance $i \in \{0,1\}^*$
Output         : a solution $s \in \{0,1\}^*$ such that $C^f(i,s) = 1$ and $|s| \le p(|i|)$

1  Compute $Z_i = \bigcup_{n=1}^{T_C(|i|+p(|i|))} \left\{y \in \{0,1\}^{\mu(n)} \mid i \in Q_{\mathrm{SOLVE}}\left(R^{f,\mathrm{SOLVE}}(1^n, y)\right)\right\}$
2  for $n \in \{1, \dots, T_C(|i| + p(|i|))\}$ do
3      if $|\{0,1\}^{\mu(n)} \setminus Z_i| < 2^n$ then
4          Set $Z_i = Z_i \setminus \{0,1\}^{\mu(n)}$
5      end
6  end
7  Compute $Y_i = Z_i \cap \mathrm{Im}(f)$
8  Compute $N_i = \{n \in \mathbb{N} \mid Y_i \cap \mathrm{Im}(f_n) \neq \emptyset\}$
9  for $n \in N_i$ do
10     Compute $Y_{i,n} = Y_i \cap \mathrm{Im}(f_n)$
11 end
12 Compute $S_{i,f} = \left\{s \in \{0,1\}^* \mid |s| \le p(|i|) \wedge C^f(i,s) = 1\right\}$
13 while True do
14     $B_{i,f} = \left\{s \in S_{i,f} \mid f\left[Q\left(C^f(i,s)\right)\right] \cap Y_i = \emptyset\right\}$
15     if $B_{i,f} \neq \emptyset$ then
16         return lexicographically smallest $s \in B_{i,f}$
17     end
18     Choose $n \in N_i$ such that $\frac{|Y_{i,n}|}{2^n}$ is maximized.
19     Set $N_i = N_i \setminus \{n\}$
20     Set $Y_i = Y_i \setminus Y_{i,n}$
21 end
```

even in the presence of SOLVE. For any fully black-box construction with a deterministic f-oblivious many-one reduction, we provide an oracle SOLVE which finds a solution for any TFNP instance (i.e., TFNP is easy in the presence of SOLVE) and argue that it does not help the reduction in inverting injective one-way functions. The description of our oracle SOLVE is given in Algorithm 6 and it is explained below.

Oracle Solve: Let (R, T_R, C, T_C, p) be the construction of a hard TFNP problem from injective one-way function with a deterministic f-oblivious many-one security reduction. We define our oracle SOLVE for the particular choice of (R, T_R, C, T_C, p), especially we hard-wire both C and R in the algorithm SOLVE. Ideally, SOLVE should output a solution i which gives the reduction R no information about the inversion of its challenge y. Unfortunately, SOLVE is unaware of the particular challenge y on which $R^f(1^n, y)$ queried SOLVE with the instance i. Nevertheless, SOLVE can compute the set Z_i of all challenges y on which the reduction queries the instance i.[6] The challenges in Z_i become "protected" and SOLVE will attempt to provide a solution which does not reveal a preimage of any $y \in Z_i$, i.e., s such that $C^f(i,s)$ does not make an f-query on a preimage of any $y \in Z_i$.

Note that we could run into a potential technical issue when defining Z_i as the set of *all* challenges y on which R queries the instance i might be infinite. However when the instance i is queried by the security reduction R on some very long challenge y then C contributes no indirect query to $f^{-1}(y)$ as the running time of C depends only on the length of the instance i. More formally: the running time of C is bounded by $T_C(|i| + p(|i|))$ thus C cannot query f on longer inputs. Therefore, we can consider only possible challenges y from $\{0,1\}^{\mu(n)}$ for $n \le T_C(|i| + p(|i|))$.

Unfortunately we cannot protect all such challenges $y \in Z_i$. For technical reasons we do not protect any challenge of length $m = \mu(n)$ for some $n \in \mathbb{N}$ if almost all challenges of length m query the instance i. More specifically if there are less then 2^n challenges in the set $\{0,1\}^m \setminus Z_i$, then any injective function from Inj_μ contains at least one string from Z_i in its image. Then if the solution of instance i would be conditioned on querying a preimage of Z_i (i.e., $C^f(i,s) = 1$ if and only if $f\left[Q\left(C^f(i,s)\right)\right] \cap Z_i \neq \emptyset$) then no matter how we choose $f \in \mathrm{Inj}_\mu$ there is a solution to the instance i. Thus we cannot use Correctness (see Definition 4.3.1) to show existence of a solution which would not query preimage of any "protected" challenge $y \in Z_i$ in this case. We show that with high probability $\mathrm{Im}(f_n)$ contains many challenges from Z_i and that any solution does not significantly help R with inverting a random challenge $y \in \mathrm{Im}(f_n)$ since the instance i is queried by R on many distinct challenges. After removing such challenges from the protected set we restrict the set to only those challenges which are in the image of the one-way function f (see the set Y_i on line 7 of Algorithm 6).

On lines 8–12, SOLVE computes the following auxiliary sets N_i, $Y_{i,n}$, and $S_{i,f}$. The set N_i contains all the lengths of the preimages x such that the reduction $R^{f,\textsc{Solve}}(1^n, f(x))$ queries the instance i. SOLVE then splits Y_i into subsets $Y_{i,n}$ using the input lengths of interest in N_i. Finally, SOLVE computes the set $S_{i,f}$ which is the set of *all* possible solutions for the instance i.

The strategy of SOLVE is to return a solution from the set of "benign" solutions $B_{i,f}$, which do not induce any query to preimages of the protected challenges in Y_i. If there is any such "benign" solution then SOLVE simply halts and returns the lexicographically smallest one. Unfortunately, it might still be the case that every solution queries a preimage of some $y \in Y_i$, e.g., if the instance i is queried for all challenges $y \in \mathrm{Im}(f)$ of a given preimage length n and on each solution s at least one x of length n is queried (i.e., $B_{i,f} = \emptyset$ unless we remove $Y_{i,n}$ from Y_i). Since SOLVE in general cannot output a solution while protecting the whole set Y_i, it will proceed to gradually relax the condition on the set of protected challenges.

Note that we might allow SOLVE to return a solution even though it induces queries to preimages of protected challenges, as long as the reduction queries the instance i on the corresponding image length often enough, as any fixed solution induces only a bounded number of queries to f (bounded by T_C). Therefore, if the

[6] Here we crucially rely on f-obliviousness of the reduction algorithm R which ensures that Z_i depends only on the type of the function f, which allows SOLVESIM to simulate SOLVE without querying f on too many inputs.

set of challenges on which R queries i is dense enough w.r.t. some preimage length then, with overwhelming probability, an arbitrary solution will be "benign" for the random challenge y given to the reduction. Thus, we allow SOLVE to return a solution revealing preimages of challenges from the auxiliary set $Y_{i,n}$ maximizing $\frac{|Y_{i,n}|}{2^n}$. If the fraction $\frac{|Y_{i,n}|}{2^n}$ is small then SOLVE is able to find a "benign" solution which protects the preimages of length n (see Claim 4.4.10).

Whereas, if the fraction $\frac{|Y_{i,n}|}{2^n}$ is large enough then any fixed solution will be "benign" w.r.t. the actual challenge of R with overwhelming probability as each solution can induce queries to preimages of only a small number of challenges from the set $Y_{i,n}$ (see Claim 4.4.9).

In order to show formally that SOLVE does not help in inverting the injective one-way function, we employ an incompressibility argument similar to Gennaro and Trevisan [2000]. Specifically, we present algorithms ENCODE$_n$ (given in Algorithm 7) and DECODE$_n$ (given in Algorithm 8) which utilize the reduction R to allow compression of a random permutation more succinctly than what is information-theoretically possible. When compressing the random permutation by ENCODE$_n$, we have access to the whole permutation and we can effectively provide the reduction with access to SOLVE. However, to be able to use the reduction also in the DECODE$_n$, we have to be able to simulate access to our SOLVE oracle given access only to a partially defined oracle f (as we are reconstructing f). For the description of the algorithm SOLVESIM, which simulates the SOLVE oracle for the purpose of decoding in DECODE$_n$, see Algorithm 9.

Encode$_n$ algorithm: The algorithm ENCODE$_n$ (Algorithm 7) uses the reduction R to compress a random permutation π on bit strings of length n. Note that even though R succeeds in inverting an injective function, for technical reasons, we leverage its power in order to compress a permutation. One particular issue we would run into when trying to compress an injective function f which is not surjective is that the encoding would have to comprise also of the encoding of the image of f which might render the encoding inefficient.

Nevertheless, in order to use the reduction for compressing, we must provide it with oracle access to an injective function which is *not a bijection.* Thus, we equip ENCODE$_n$ (as well as DECODE$_n$) with an injective function h. ENCODE$_n$ then computes the function f as a composition of the functions $h \circ \pi$ and uses the reduction with respect to the composed oracle f. We emphasize that h is independent of π, therefore it cannot be used in order to compress π on its own.

First, ENCODE$_n$ computes the set INV_f which is the set of all challenges y on which the reduction successfully inverts (i.e., the reduction returns $f^{-1}(y)$). Then ENCODE$_n$ computes the set G_f, which is the set of "good" challenges y, on which the reduction successfully inverts even though SOLVE returns a solution which does not induce a query to any preimage of y. This set is used to reduce the size of the trivial encoding of f – the part of f corresponding to the challenges in G_f will be algorithmically reconstructed by DECODE$_n$ using the security reduction R.

Specifically, ENCODE$_n$ computes Y_f, the subset of G_f for which the preimages will be algorithmically reconstructed, as follows: ENCODE$_n$ processes the challenges y in G_f one by one in lexicographically increasing order and stores all f-queries needed for reconstruction by R (i.e., for any x' such that there was an f-query x', the element $f(x')$ is removed from the "good" set G_f as we cannot

Algorithm 7: The algorithm $\textsc{Encode}_n$.

```
Hardwired       : a deterministic $f$-oblivious many-one fully black-box
                  construction $(R, T_R, C, T_C, p)$ of a worst-case hard TFNP
                  problem from injective-OWF of type $\mu$
Common Input    : an injective function $h \in \mathrm{Inj}_\mu$ shared with $\textsc{Decode}_n$
Input           : a permutation $\pi \in \mathrm{Inj}_n^n$
Output          : an encoding $\mathcal{M}$ of $\pi$

1  $f = h \circ \pi$, i.e., $f(x) = \begin{cases} h(\pi(x)) & \text{for all } x \text{ of length } n \\ h(x) & \text{otherwise} \end{cases}$
2  $\mathrm{INV}_f = \left\{ y \in \mathrm{Im}(h_n) \mid R^{f,\textsc{Solve}}(1^n, y) = f^{-1}(y) \right\}$
3  $G_f = \left\{ y \in \mathrm{INV}_f \mid f^{-1}(y) \notin Q_f^{\mathrm{indir}}\left(R^{f,\textsc{Solve}}(1^n, y)\right) \right\}$
4  $Y_f = \emptyset$ and $X_f = \emptyset$
5  while $G_f \neq \emptyset$ do
6      Pick lexicographically smallest $y \in G_f$
7      $G_f = G_f \setminus \left( f\left[ Q_f\left( R^{f,\textsc{Solve}}(1^n, y) \right) \right] \cup \{y\} \right)$
8      $Y_f = Y_f \cup \{y\}$ and $X_f = X_f \cup \{f^{-1}(y)\}$
9  end
10 if $|X_f| < 2^{0.6n}$ then
11     return $\mathcal{M} = (0, \pi)$
12 end
13 else
14     return $\mathcal{M} = (1, |X_f|, Y_f, X_f, \sigma = f \restriction (\{0,1\}^n \setminus X_f)) \in$
         $\{0,1\}^{1+n+\left\lceil \log \binom{2^n}{|Y_f|} \right\rceil + \left\lceil \log \binom{2^n}{|X_f|} \right\rceil + \lceil \log(|\{0,1\}^n \setminus X_f|!) \rceil}$
15 end
```

reconstruct the preimage of y using R without knowing the image of x' under f).

$\textsc{Encode}_n$ outputs an encoding $\mathcal{M}$ which describes the size of X_f, the sets Y_f and X_f (where X_f is the set of preimages corresponding to Y_f), and the partial function representing the function f on inputs of length n outside of X_f. Thus, the encoding saves bits by not revealing the bijection between X_f and Y_f which is algorithmically reconstructed instead (we bound the savings in Lemma 4.3.8). Specifically, the size of X_f (equal to the size of Y_f) can be encoded using $\log 2^n = n$ bits. Y_f is a subset of $\mathrm{Im}(f_n) = \mathrm{Im}(h_n)$ and it is encoded using $\lceil \log \binom{2^n}{|Y_f|} \rceil$ bits as the index of the corresponding subset of size $|Y_f|$ (the set X_f is encoded in a similar manner). Finally, the bijection between $\{0,1\}^n \setminus X_f$ and $\mathrm{Im}(f) \setminus Y_f$ is encoded as the index of the corresponding permutation on a set of size $|\{0,1\}^n \setminus X_f|$ using $\lceil \log(|\{0,1\}^n \setminus X_f|!) \rceil$ bits.

A small technicality arises when the set X_f, respectively the set Y_f, is not large enough, the above mentioned encoding would be inefficient as the trivial encoding outputting the whole description of the permutation π would use fewer bits. Thus, $\textsc{Encode}_n$ simply outputs the trivial encoding when X_f is too small. The first bit of the encoding distinguishes between these two cases.

Decode$_n$ algorithm: The encoding returned by $\textsc{Encode}_n$ is uniquely decodable by $\textsc{Decode}_n$ given in Algorithm 8 (see Section 4.4.2). When the output

Algorithm 8: The algorithm $\textsc{Decode}_n$.

```
Hardwired     : a deterministic $f$-oblivious many-one fully black-box
                construction $(R, T_R, C, T_C, p)$ of a worst-case hard TFNP
                problem from injective-OWF of type $\mu$
Common Input: an injective function $h \in \mathrm{Inj}_\mu$ shared with $\textsc{Encode}_n$
Input         : an encoding $\mathcal{M}$
Output        : a permutation $\pi \in \mathrm{Inj}_n^n$

1  Parse $\mathcal{M} = (b, \mathcal{M}')$, where $b \in \{0,1\}$
2  if $b = 0$ then
3      Decode $\pi$ from $\mathcal{M}'$
4      return $\pi$
5  end
6  Parse $\mathcal{M}' = (|X_f|, Y_f, X_f, \sigma)$
   /* $f'$ is partial as it is defined only outside $X_f$ */
7  Set partial function $f' = \begin{cases} \sigma & \text{for inputs of length } n \\ h & \text{otherwise} \end{cases}$
8  while $Y_f \neq \emptyset$ do
9      Pick lexicographically smallest $y \in Y_f$
10     Let $f''(x) = \begin{cases} y & \text{for all } x \in \mathrm{Dom}(h) \setminus \mathrm{Dom}(f') \\ f'(x) & \text{otherwise} \end{cases}$
       /* By $\textsc{SolveSim}(h, f', \cdot)$ we denote the fact that we run SolveSim
          with first two inputs fixed to $h$ respectively $f'$ and only
          the last input (instance $i$) is provided by the reduction.
          */
11     $x = R^{f'', \textsc{SolveSim}(h, f', \cdot)}(1^n, y)$
12     Let $f'(x) = y$
13     Set $Y_f = Y_f \setminus \{y\}$
14 end
15 return $\pi = (h^{-1} \circ f') \restriction \{0,1\}^n$
```

of $\textsc{Encode}_n$ starts with “0” bit, the rest of the encoding is an explicit encoding of π and we are immediately done with its reconstruction. If the output starts with “1” bit, the following n bits represent $|X_f| = |Y_f|$. $\textsc{Decode}_n$ then reads the following $\left\lceil \log \binom{2^n}{|X_f|} \right\rceil$ bits of the encoding to reconstruct the set Y_f (as the j-th subset of $\mathrm{Im}(f_n) = \mathrm{Im}(h_n)$ of size $|X_f|$). Similarly, $\textsc{Decode}_n$ reconstructs the set X_f using the following $\left\lceil \log \binom{2^n}{|X_f|} \right\rceil$ bits. The remaining bits represent σ, a restriction of f on all the n-bit inputs outside of X_f, given by the index of the corresponding bijection between $\{0,1\}^n \setminus X_f$ and $\mathrm{Im}(f_n) \setminus Y_f$. Note that such encoding of σ does preserve the structure of the restriction but it looses the information about the domain and image of σ. However, both are easy to reconstruct. The domain is simply $\{0,1\}^n \setminus X_f$ and the image of σ can be computed from Y_f and the common input h as $\mathrm{Im}(\sigma) = \mathrm{Im}(f_n) \setminus Y_f = \mathrm{Im}(h_n \circ \pi) \setminus Y_f = \mathrm{Im}(h_n) \setminus Y_f$.

$\textsc{Decode}_n$ then computes the remaining preimages one by one in lexicographic order (on the remaining images in Y_f) using the security reduction R, adding the reconstructed mapping into a partial function f'. Note that during the computation of the preimage of y, the reduction might make an f-query on x which has no defined output. But as $\textsc{Decode}_n$ takes $y \in Y_f$ in the same order as $\textsc{Encode}_n$ added them to the set Y_f, this happens if and only if the preimage of the current challenge y is being queried. Thus, we answer any such query by y (it is crucial that this happens only for f-queries made directly by R) which is captured in the definition of the auxiliary function f'' defined by $\textsc{Decode}_n$ and used as the oracle for the security reduction R.

Once $\textsc{Encode}_n$ finds the preimages of all challenges y from Y_f, the function f' is defined everywhere. To reconstruct the permutation π on $\{0,1\}^n$, $\textsc{Decode}_n$ can simply compose the inverse of h with the reconstructed function f'.

SolveSim algorithm: For the ease of presentation we usually do not explicitly mention the oracle h as it is given by context (we run $\textsc{Decode}_n$ and $\textsc{SolveSim}$ with respect to only one h at a time).

The computation of the algorithm $\textsc{SolveSim}$ (Algorithm 9) is similar to the computation of $\textsc{Solve}$ (Algorithm 6). First, $\textsc{SolveSim}$ computes the sets Z_i, Y_i, N_i and $Y_{i,n}$ for all $n \in N_i$. There is one big difference between $\textsc{Solve}$ and $\textsc{SolveSim}$. As $\textsc{SolveSim}$ does not have access to the whole function f it uses h or the partial knowledge of f, namely the partial function f', everywhere f is used in the $\textsc{Solve}$ algorithm.

- We use h whenever we need to determine the image of f_n for some n. As $\forall n \in \mathbb{N}\colon \mathrm{Im}(h_n) = \mathrm{Im}(f_n)$ using $\mathrm{Im}(h_n)$ instead of $\mathrm{Im}(f_n)$ makes no difference to the computation.

- The second place where h is used instead of f is when $\textsc{SolveSim}$ computes the set Z_i. Specifically, when determining challenges y for which the security reduction R queries the given instance i, the algorithm $\textsc{SolveSim}$ computes the same Z_i as if it used f by the f-obliviousness of the security reduction.

- In all other places, $\textsc{SolveSim}$ uses the partial knowledge of f (i.e., the partial function f'). This causes a real difference in the computation. In particular, the set $S_{i,f'}$ (as computed by $\textsc{SolveSim}$) may differ a lot from $S_{i,f}$ (as computed by $\textsc{Solve}$) as some solutions from $S_{i,f}$ potentially query some unknown parts of f. Thus, the set $S_{i,f'}$ computed by $\textsc{SolveSim}$ is just a subset of the whole $S_{i,f}$. The set $S_{i,f'}$ contains only the solutions $\textsc{SolveSim}$ is "aware of" (i.e., those for which f' is defined for all queries and thus $\textsc{SolveSim}$ may verify the solution). The rest of the computation is practically the same, except that $\textsc{SolveSim}$ is restricted just to the set of solutions $S_{i,f'}$. The main trick is that we make sure that $\textsc{SolveSim}$ is aware of the solution which should be returned and it does not matter that it ignores other solutions of the instance.

Structure of the proof of Theorem 4.3.4. For ease of presentation and understanding, we divide the proof into four lemmata, Lemma 4.3.5 through 4.3.8. Lemma 4.3.5 shows that given an instance i of the TFNP problem represented by the algorithm C^f, our $\textsc{Solve}$ always returns a solution, i.e., an $s \in \{0,1\}^*$ such that $C^f(i,s) = 1$ (formal proof is given in Section 4.4.1). Thus, any distribution of instances of the TFNP problem is easy in the presence of $\textsc{Solve}$.

Algorithm 9: The algorithm SOLVESIM.

```
Hardwired : a deterministic $f$-oblivious many-one fully black-box
            construction $(R, T_R, C, T_C, p)$ of a worst-case hard TFNP
            problem from injective-OWF of type $\mu$
Input     : a function $h \in \mathrm{Inj}_\mu$, partial injective function $f' \in \mathrm{Inj}_\mu$, and
            an instance $i \in \{0,1\}^*$
Output    : a solution, i.e., $s \in \{0,1\}^* : C^{f'}(i,s) = 1 \wedge |s| \leq p(|i|)$

 1 Compute $Z_i = \bigcup_{n=1}^{T_C(|i|+p(|i|))} \left\{ y \in \{0,1\}^{\mu(n)} \mid i \in Q_{\mathrm{SOLVE}}\left(R^{h,\mathrm{SOLVE}}(1^n, y)\right)\right\}$
 2 for $n \in \{1, \ldots, T_C(|i| + p(|i|))\}$ do
 3     if $|\{0,1\}^{\mu(n)} \setminus Z_i| < 2^n$ then
 4         Set $Z_i = Z_i \setminus \{0,1\}^{\mu(n)}$
 5     end
 6 end
 7 Compute $Y_i = Z_i \cap \mathrm{Im}(h)$
 8 Compute $N_i = \{n \in \mathbb{N} \mid Y_i \cap \mathrm{Im}(h_n) \neq \emptyset\}$
 9 for $n \in N_i$ do
10     Compute $Y_{i,n} = Y_i \cap \mathrm{Im}(h_n)$
11 end
12 Compute
     $S_{i,f'} = \left\{ s \in \{0,1\}^* \;\middle|\; |s| \leq p(|i|) \wedge Q\left(C^{f'}(i,s)\right) \subseteq \mathrm{Dom}(f') \wedge C^{f'}(i,s) = 1 \right\}$
13 while True do
14     $B_{i,f'} = \left\{ s \in S_{i,f'} \mid f\left[Q\left(C^{f'}(i,s)\right)\right] \cap Y_i = \emptyset \right\}$ // "benign" solutions
15     if $B_{i,f'} \neq \emptyset$ then
16         return lexicographically smallest $s \in B_{i,f'}$
17     end
18     Choose $n \in N_i$ such that $\frac{|Y_{i,n}|}{2^n}$ is maximized.
19     Set $N_i = N_i \setminus \{n\}$
20     Set $Y_i = Y_i \setminus Y_{i,n}$
21 end
```

Lemma 4.3.5. *For any instance $i \in \{0,1\}^*$, any type $\mu \in \mathrm{T}$ and any $f \in \mathit{Inj}_\mu$, the algorithm $\mathrm{SOLVE}^f(i)$ halts and returns a solution, i.e., it returns a string $s \in \{0,1\}^*$ such that $|s| \leq p(|i|)$ and $C^f(i,s) = 1$.*

To argue that SOLVE does not help in inverting injective functions, we analyze the joint properties of the algorithms ENCODE_n and DECODE_n. First, we show that DECODE_n always returns the correct permutation encoded by ENCODE_n (see Section 4.4.2 for the formal proof).

Lemma 4.3.6. *For all $n \in \mathbb{N}$, $\mu \in \mathrm{T}$, $\pi \in \mathit{Inj}_n^n$, and $h \in \mathit{Inj}_\mu$,*

$$\mathrm{DECODE}_n^h\left(\mathrm{ENCODE}_n^h(\pi)\right) = \pi,$$

where ENCODE_n, respectively DECODE_n, is given in Algorithm 7, respectively Algorithm 8.

We crucially rely on f-obliviousness of the security reduction for the proof of Lemma 4.3.6. It is the property which allows us to simulate the algorithm

SOLVE during the decoding phase, as SOLVESIM needs to be able to compute the same set Z_i, respectively Y_i, as SOLVE does. Moreover, SOLVESIM cannot query f on all preimages as SOLVE does when computing Z_i, respectively Y_i. Due to f-obliviousness of the reduction, we may substitute f by h in the computation of Z_i and Y_i in SOLVESIM as the resulting sets depends only on the image of the function given to R as an oracle (and $\mathrm{Im}(f) = \mathrm{Im}(h)$).

Second, we show that the encoding output by ENCODE_n is prefix-free (see Section 4.4.3 for the formal proof).

Lemma 4.3.7. *Let $\mu \in \mathrm{T}$ be any type, $h \in Inj_\mu$ be any injective function and $n \in \mathbb{N}$, then the encoding given by the algorithm ENCODE_n (Algorithm 7) is prefix-free, i.e.,*

$$\forall \pi, \pi' \in Inj_n^n \text{ such that } \pi \neq \pi' \colon \text{ENCODE}_n^h(\pi) \text{ is not a prefix of } \text{ENCODE}_n^h(\pi').$$

Finally, we bound the expected size of the encoding given by ENCODE_n (see Section 4.4.4) which contradicts the information-theoretic bound implied by Corollary 4.2.7.

Lemma 4.3.8. *Let $\mu \in \mathrm{T}$ be any type and (R, T_R, C, T_C, p) be a deterministic f-oblivious many-one fully black-box construction of a worst-case hard TFNP problem from an injective one-way function of type μ. Assume $n \in \mathbb{N}$ is large enough so that $n \geq 50$ and $2q(n) + 2 \leq 2^{0.2n}$, where $q(n)$ is the maximal number of f-queries made by C on the queried instance (see Definition 4.4.5). Let the success probability of R on security parameter n be $\beta \geq 2^{-0.1n}$, i.e., for any $f \in Inj_\mu$ we have*

$$\Pr_{x \leftarrow \{0,1\}^n}\left[R^{f,\text{SOLVE}}(1^n, f(x)) = x\right] = \beta \geq 2^{-0.1n}.$$

Then

$$\exists h \in Inj_\mu \colon \mathbb{E}_{\pi \leftarrow Inj_n^n}\left[\left|\text{ENCODE}_n^h(\pi)\right|\right] \leq \log\left(2^n!\right) - \frac{8}{10} n 2^{0.1n}.$$

We claim (see Claim 4.4.6) that the upper bound $2q(n) + 2 \leq 2^{0.2n}$ used in the statement of the lemma is without loss of generality for large enough n and for all quasi-polynomial (and, hence, also for efficient) algorithms R, C. We use this fact again in the proof of the main theorem (Theorem 4.3.4), and refer the readers to Section 4.4.4 for the precise statement and its proof.

Equipped with the above lemmata, we prove Theorem 4.3.4.

Proof of Theorem 4.3.4. Suppose to the contrary that there is such a construction (R, T_R, C, T_C, p). By Lemma 4.3.5, the algorithm SOLVE (Algorithm 6) returns a valid solution with probability one. Thus, the reduction R must invert f with high probability when given access to any oracle $f \in \mathrm{Inj}_\mu$ and our oracle SOLVE, i.e.,

$$\Pr_{x \leftarrow \{0,1\}^n}\left[R^{f,\text{SOLVE}}(1^n, f(x)) = x\right] \geq \frac{1}{p'(n)}$$

for some polynomial p' and infinitely many $n \in \mathbb{N}$.

Let $n \in \mathbb{N}$ be large enough such that

1. $2q(n) + 2 \leq 2^{0.2n}$,
2. $\Pr_{x \leftarrow \{0,1\}^n} \left[R^{f,\textsc{Solve}}(1^n, f(x)) = x \right] \geq \frac{1}{p'(n)}$,

where $q(n)$ is the maximal number of f-queries made by C on the queried instance (see Definition 4.4.5). As already pointed out, the quasi-polynomial bounds on running times $T_C, T_R \in O\left(n^{\text{polylog}(n)}\right)$ imply that $q(n) \in o\left(2^{0.2n}\right)$ (see Claim 4.4.6). Thus, for large enough n, the upper bound $2q(n) + 2 \leq 2^{0.2n}$ holds without loss of generality.

For any $h \in \text{Inj}_\mu$, we can use the algorithm $\textsc{Encode}^h_n$ (Algorithm 7) to encode a given permutation $\pi \in \text{Inj}^n_n$. Decodability of the encoding follows from Lemma 4.3.6. Moreover, by Lemma 4.3.7, the encoding is a prefix-free code. By Lemma 4.3.8, there is a function $h \in \text{Inj}_\mu$ such that the pair of algorithms $\textsc{Encode}^h_n$ and $\textsc{Decode}^h_n$ defines an encoding of $\pi \leftarrow \text{Inj}^n_n$ with expected length at most $\log(2^n!) - \frac{8}{10} n 2^{0.1n}$. This contradicts the information-theoretic bound on the expected length of any prefix-free encoding of a random permutation on $\{0,1\}^n$ given by Corollary 4.2.7. □

4.4 Proofs of the Supporting Lemmata for Theorem 4.3.4

The first lemma we need to prove is that the algorithm Solve halts and returns a valid solution on any input TFNP instance i. Section 4.4.2 is devoted to proving the correctness of our encoding, i.e., the fact that the algorithm $\textsc{Decode}_n$ uniquely and correctly decodes $\textsc{Encode}_n$'s input permutation. We show that the encoding itself is prefix-free in Section 4.4.3. Section 4.4.4 is the core of the incompressibility argument. We show that the expected size of our encoding is smaller than the information-theoretic bound.

4.4.1 Solve Always Returns a Solution

In this section we show that algorithm Solve halts and returns a proper solution. When the algorithm Solve halts it is easy to show that the returned solution is a valid solution of the given TFNP problem and that it is of the correct length (such a solution has to exist by correctness of the reduction). Thus the proof essentially reduces to proving that Solve eventually halts, at latest when the set Y_i becomes empty.

Lemma 4.3.5. (Restated) *For any instance $i \in \{0,1\}^*$, any $\mu \in \mathrm{T}$ and any $f \in \mathit{Inj}_\mu$ the algorithm $\textsc{Solve}^f(i)$ halts and returns a solution, i.e., it returns a string $s \in \{0,1\}^*$ such that $|s| \leq p(|i|)$ and $C^f(i,s) = 1$.*

Proof. First observe that all sets $Z_i, Y_i, N_i, Y_{i,n}$ and $S_{i,f}$ are well defined and can be computed in finite time.

Notice that in each iteration of the while loop we remove one n from the set N_i. Thus after $|N_i|$ iterations the set N_i is empty. During the whole algorithm we keep the invariant that

$$Y_i \subseteq \bigcup_{n \in N_i} \{0,1\}^n$$

(see lines 7, 8, 19 and 20 of Algorithm 6). After $|N_i|$ iterations of the while loop the set Y_i is empty as well. Once $Y_i = \emptyset$, the set $B_i = S_{i,f}$. As the set $S_{i,f}$ is nonempty by correctness of the TFNP problem (see Definition 4.3.1 - Correctness) the algorithm Solve always halts.

Finally when the algorithm returns (only line 16 of Algorithm 6), it returns a string s from the set $B_i \subseteq S_{i,f}$. Thus $|s| \leq p(|i|)$ and $C^f(i,s) = 1$ by the definition of the set $S_{i,f}$ (see line 12 of Algorithm 6). $\square$

4.4.2 Encoding Is Uniquely Decodable

In this section, we show that the encoding given by Encode$_n$ (Algorithm 7) is uniquely decodable by Decode$_n$ (Algorithm 8), i.e., we prove the following lemma.

Lemma 4.3.6. (Restated) *For any $n \in \mathbb{N}$, any $\mu \in \mathrm{T}$, any $\pi \in Inj_n^n$ and for any $h \in Inj_\mu$*

$$\textit{Decode}_n^h\left(\textit{Encode}_n^h(\pi)\right) = \pi,$$

where Encode$_n$, Decode$_n$ are as described in Algorithms 7 and 8.

To prove this lemma, we work with a partial function f' computed during the decoding. We show by induction on the number of steps of Decode$_n$ that during the whole computation f' agrees with the function f. During decoding we fill in the undefined values of f'. To find the missing values we find preimages of challenges y from image of f one by one using the reduction algorithm R.

We need to show that the missing mapping is computed correctly. To this end, we show that the reduction as run during the decoding phase gives the same output as it gave when run during the encoding phase. To prove this we show that all f-queries (both direct and indirect) made by the reduction (when run in the decoding phase) are answered exactly as if f would be queried.

The first step is to show that all direct f-queries are answered correctly (Claim 4.4.1).

Claim 4.4.1. *Let $n \in \mathbb{N}$ be any number, $\mu \in \mathrm{T}$ be any type, $h \in Inj_\mu$ be any function, $\pi \in Inj_n^n$ be any permutation and let $f = h \circ \pi$. Let $\mathcal{M}$ be the output of Encode$_n^h(\pi)$ (see Algorithm 7) and assume that $\mathcal{M} = (1, |X_f|, Y_f, X_f, \sigma)$. Let $j \in \mathbb{N}$ be such that $1 \leq j \leq |X_f|$.*

Let y be the j-th lexicographically smallest string from Y_f (i.e., y is picked in the j-th iteration of the while loop, see line 8 of Algorithm 8). Let

$$X_{f,j} = \{x \in X_f | f(x) \textit{ is the } k\textit{-th lexicographically smallest in } Y_f \textit{ for } k < j\},$$

$$f' = f \restriction \left(X_{f,j} \cup \overline{X_f}\right)$$

and

$$f'' \colon \{0,1\}^* \to \{0,1\}^* \textit{ be such that } f''(x) = \begin{cases} f'(x) & \textit{for } x \in Dom(f') \\ y & \textit{otherwise} \end{cases}$$

Then

$$\forall x' \in Q_f^{dir}\left(R^{f,\textit{Solve}}(1^n, y)\right) \quad f''(x') = f(x').$$

Note that if Decode_n computed the first $j-1$ preimages then f', f'' from Algorithm 8 correspond to their definition given in Claim 4.4.1. This is formalized in the proof of Claim 4.4.3.

Proof of Claim 4.4.1. Let x' be any query from $Q_f^{\text{dir}}\left(R^{f,\text{Solve}}(1^n, y)\right)$. We distinguish the following four cases:

$x' \notin X_f$**:** In this case $f'(x')$ is defined (it is set before the first iteration of the while loop). Thus $f''(x') = f'(x') = f(x')$, where the last equality follows from the assumptions.

$x' \in X_f \wedge f(x') <_{\text{lex}} y$**:** Then $x' \in X_{f,j}$ and $f'(x')$ is defined in the j-th iteration, as Decode_n (Algorithm 8) computes preimage of $f(x')$ before y. Thus $f''(x') = f'(x') = f(x')$, where the last equality follows from the assumptions.

$x' \in X_f \wedge f(x') = y$**:** In this case $f'(x')$ is undefined (we are computing the preimage of y in j-th iteration of while loop), thus by definition of f'', $f''(x') = y = f(x')$.

$x' \in X_f \wedge f(x') >_{\text{lex}} y$**:** As $f(x') >_{\text{lex}} y$, the Encode_n (Algorithm 7) processes y before $f(x')$. Then if x' is in $Q_f^{\text{dir}}\left(R^{f,\text{Solve}(f,\cdot)}(1^n, y)\right)$, Encode_n removes $f(x')$ from G_f (line 7, Algorithm 7). This means that $f(x')$ is never added to Y_f and similarly x' is never added to X_f. Thus no such direct query to f can occur during the computation of $R^{f,\text{Solve}}(1^n, y)$.

$\square$

Claim 4.4.1 shows that each direct oracle query to f (that is a query done by the reduction itself) is answered correctly by f''. Next we show that both algorithms Solve (Algorithm 6) and SolveSim (Algorithm 9) answer the TFNP query with the same solution whenever the algorithm Decode_n (Algorithm 8) queries SolveSim during the simulation of the security reduction R.

The main observation is that the algorithm SolveSim is aware of the solution returned by Solve (i.e., f' is defined for all f-queries made during the computation). Then we show that the procedure which picks the solution out of all solutions in algorithm Solve gives the same result in algorithm SolveSim, which is possibly aware of only a few solutions of the instance. Here we utilize the fact that the algorithm Solve always returns lexicographically smallest "benign" solution.

Claim 4.4.2. *Let $n \in \mathbb{N}$ be any number, $\mu \in \mathrm{T}$ be any type, $h \in Inj_\mu$ be any function, $\pi \in Inj_n^n$ be any permutation and let $f = h \circ \pi$. Let $\mathcal{M}$ be the output of $\text{Encode}_n^h(\pi)$ (see Algorithm 7) and assume that $\mathcal{M} = (1, |X_f|, Y_f, X_f, \sigma)$. Let $j \in \mathbb{N}$ be such that $1 \leq j \leq |X_f|$. Let y be the j-th lexicographically smallest string from Y_f (i.e., y is picked in the j-th iteration of the while loop). Let*

$$X_{f,j} = \{x \in X_f \mid f(x) \textit{ is the } k\textit{-th lexicographically smallest in } Y_f \textit{ for } k < j\}$$

and

$$f' = f \restriction \left(X_{f,j} \cup \overline{X_f}\right).$$

Then the query to Solve *is answered correctly, i.e.,*

$$\forall i \in Q_{\textsc{Solve}}\left(R^{f,\textsc{Solve}}(1^n, y)\right) \quad \textsc{Solve}^f(i) = \textsc{SolveSim}(h, f', i)\,.$$

Proof. We fix $i \in Q_{\textsc{Solve}}\left(R^{f,\textsc{Solve}}(1^n, y)\right)$. Let $s_{\textsc{Solve}}$ be the solution returned by $\textsc{Solve}^f(i)$ and $s_{\textsc{SolveSim}}$ be the solution returned by $\textsc{SolveSim}(h, f', i)$. To prove $s_{\textsc{Solve}} = s_{\textsc{SolveSim}}$, we examine the sets $S_{f,i}, B_{f,i}, S_{f',i}$ and $B_{f',i}$ used in the algorithms Solve and SolveSim. For the ease of presentation we denote the sets as follows:

$S_{\textsc{Solve},f}$: Let $S_{\textsc{Solve},f}$ denote the set $S_{f,i}$ as computed by the algorithm Solve (Algorithm 6, line 12).

$S_{\textsc{SolveSim},f'}$: Let $S_{\textsc{SolveSim},f'}$ denote the set $S_{f',i}$ as computed by the algorithm SolveSim (Algorithm 9, line 12).

$B_{\textsc{Solve},f,k}$: Let $B_{\textsc{Solve},f,k}$ denote the set $B_{f,i}$ as computed in the k-th iteration of the while loop by the algorithm Solve (Algorithm 6, line 14).

$B_{\textsc{SolveSim},f',k}$: Let $B_{\textsc{SolveSim},f',k}$ denote the set $B_{f',i}$ as computed in the k-th iteration of the while loop of the algorithm SolveSim (Algorithm 9, line 14).

Moreover by f-obliviousness the sets $Z_i, Y_i, N_i, Y_{i,n}$ depend only on the type and the image of f which is identical to the image of h and h is known to SolveSim. Thus all sets used in algorithms Solve and SolveSim except for $S_{\textsc{Solve},f}$, $S_{\textsc{SolveSim},f'}$, $B_{\textsc{Solve},f,k}$, $B_{\textsc{SolveSim},f',k}$ are independent of f (just dependent on its image) and can be computed by both Solve and SolveSim.

First recall that $y \in Y_f$ thus by the definition of Y_f, respectively the definition of $G_f \supseteq Y_f$ (Algorithm 7, lines 3, 4, 6 and 8), we get that $f^{-1}(y) \notin Q\left(C^f(i, s)\right)$ as y would be never added to G_f otherwise. Also observe that images of all queries $x \in Q_f\left(R^{f,\textsc{Solve}}(1^n, y)\right) \supseteq Q\left(C^f(i, s)\right)$ are removed from G_f (Algorithm 7, line 7) while y is processed. This means that for any $x \in Q\left(C^f(i, s)\right)$ either $x \notin X_f$ or $f(x) <_{\text{lex}} y$, in both cases $f'(x)$ is defined and $f'(x) = f(x)$ by the assumption of the claim. Thus

$$s_{\textsc{Solve}} \in S_{\textsc{SolveSim},f'}. \tag{4.4.1}$$

On the other hand by the assumptions of the claim $f' \subseteq f$, thus every $s \in S_{\textsc{SolveSim},f'}$ is a solution also with respect to f. In other words $s_{\textsc{SolveSim}} \in S_{\textsc{Solve},f}$ and more generally the following property holds:

$$S_{\textsc{SolveSim},f'} \subseteq S_{\textsc{Solve},f}. \tag{4.4.2}$$

During the entire computation both Solve and SolveSim hold the exact same set Y_i, as that is computed and updated independently of f, respectively f'. Thus by the definition of $B_{\textsc{SolveSim},f',k}$ and $B_{\textsc{Solve},f,k}$ and Equation (4.4.2) we have that

$$\forall k \in \mathbb{N} \quad B_{\textsc{SolveSim},f',k} = B_{\textsc{Solve},f,k} \cap S_{\textsc{SolveSim},f'}. \tag{4.4.3}$$

Fix $k \in \mathbb{N}$ such that $B_{\textsc{Solve},f,k}$ or $B_{\textsc{SolveSim},f',k}$ is nonempty. We show that both sets $B_{\textsc{Solve},f,k}$ and $B_{\textsc{SolveSim},f',k}$ are non-empty. Moreover we show the following:

$$\{s_{\textsc{Solve}}, s_{\textsc{SolveSim}}\} \subseteq B_{\textsc{Solve},f,k} \cap B_{\textsc{SolveSim},f',k} \tag{4.4.4}$$

We distinguish the following cases:

$B_{\textsc{Solve},f,k} = \emptyset$: By the assumption $B_{\textsc{SolveSim},f',k} \neq \emptyset$. Thus SolveSim returns in the k-th iteration of the while loop and the returned solution $s_{\textsc{SolveSim}}$ has to be contained in $B_{\textsc{SolveSim},f',k}$. By Equation (4.4.3), we get that $s_{\textsc{SolveSim}} \in B_{\textsc{Solve},f,k}$. Which is a contradiction as we assumed that $B_{\textsc{Solve},f,k}$ is empty.

$B_{\textsc{Solve},f,k} \neq \emptyset$: As $B_{\textsc{Solve},f,k} \neq \emptyset$, the Solve algorithm returns in the k-th iteration of the while loop and thus $s_{\textsc{Solve}} \in B_{\textsc{Solve},f,k}$. By Equation (4.4.1), $s_{\textsc{Solve}}$ is contained in $S_{\textsc{SolveSim},f'}$ too. Thus by Equation 4.4.3, $s_{\textsc{Solve}} \in B_{\textsc{SolveSim},f',k}$. Now using the fact that $B_{\textsc{SolveSim},f',k} \neq \emptyset$ and the same argumentation as in the case above we get that $s_{\textsc{SolveSim}} \in B_{\textsc{SolveSim},f',k}$ and $s_{\textsc{SolveSim}} \in B_{\textsc{Solve},f,k}$. Which proves that both $s_{\textsc{Solve}}, s_{\textsc{SolveSim}} \in B_{\textsc{Solve},f,k} \cap B_{\textsc{SolveSim},f',k}$.

Thus we proved that both algorithms Solve and SolveSim return in the k-th iteration of their while loop. Recall that the lexicographically smallest solution from $B_{\textsc{Solve},f,k}$, respectively $B_{\textsc{SolveSim},f',k}$, is returned. Thus by Equation 4.4.4 both algorithms Solve and SolveSim return the same string $s_{\textsc{Solve}} = s_{\textsc{SolveSim}}$. □

We combine Claims 4.4.1 and 4.4.2 to show that the function f' is defined consistently with f during the whole computation.

Claim 4.4.3. *Let $n \in \mathbb{N}$ be any number, $\mu \in \mathrm{T}$ be any type, $h \in Inj_\mu$ be any function, $\pi \in Inj_n^n$ be any permutation and let $f = h \circ \pi$. Let $\mathcal{M}$ be the output of* Encode$_n^h(\pi)$ *(see Algorithm 7) and assume that $\mathcal{M} = (1, |X_f|, Y_f, X_f, \sigma)$. Let $j \in \mathbb{N}$ be such that $0 \leq j \leq |X_f|$. Let f' be the partial function computed by* Decode$_n^h(\mathcal{M})$ *after j iterations of the while loop (line 8, Algorithm 8). Then*

$$f' \subseteq f$$

i.e., for any $x \in Dom(f')$ we have $f'(x) = f(x)$.

Proof. We proceed by induction on j, i.e., the number of iterations of the while loop of the algorithm Decode$_n$ (Algorithm 8). Before the first iteration, we set f' according to h for all strings of length different from n. Thus f' is the same as f on these strings because f differs from h only by applying a permutation to $\mathrm{Im}(h_n)$ (more specifically by applying π to $\{0,1\}^n$ before applying h_n). For inputs of length n we set f' according to σ (see line 11 of Algorithm 7). Recall that $\sigma = f \restriction (\{0,1\}^n \setminus X_f)$, thus f' is consistent with f on all strings of length n for which f' is defined. Thus, we immediately get $f' \subseteq f$.

In each iteration of the while loop, the algorithm Decode$_n$ (Algorithm 8) sets the preimage of exactly one $y \in \mathrm{Im}(f)$. We show that the preimage is computed correctly. Suppose that we are in the j-th iteration of the while loop (line 8, Algorithm 8) and y is chosen in the j-th iteration as the lexicographically smallest string in Y_f (i.e., for all $y' \in \mathrm{Im}(f)$, $y' <_{\mathrm{lex}} y$, we know the preimage of y'). We set the preimage of y in f' to be the output of $R^{f'',\textsc{SolveSim}(h,f',.)}(1^n, y)$.

Note that the fact that y is in Y_f implies that $R^{f,\textsc{Solve}(f,\cdot)}(1^n, y) = f^{-1}(y)$, as $y \in Y_f$ are taken from the set INV_f (see line 2, Algorithm 7). As the preimage of y is computed as $R^{f'',\textsc{SolveSim}(h,f',.)}(1^n, y)$, we need to show that all oracle queries in the computation are answered the same way as in the computation $R^{f,\textsc{Solve}(f,.)}(1^n, y)$. Then

$$f'^{-1}(y) = R^{f'',\textsc{SolveSim}(h,f',.)}(1^n, y) = R^{f,\textsc{Solve}(f,.)}(1^n, y) = f^{-1}(y).$$

We use Claims 4.4.1 and 4.4.2 to argue that the queries were answered correctly. Note that the assumption on f' used in these claims follows from the induction hypothesis. As in the j-th iteration of the while loop f' is defined for all $x \notin X_f$ and for the $j-1$ preimages of the lexicographically smallest strings from Y_f. By the induction hypothesis f' is defined correctly (i.e., f' gives the same output as f on all $x \in \mathrm{Dom}(f')$). By Claim 4.4.1 all f-queries are answered correctly, i.e., for any $x' \in Q_f^{\mathrm{dir}}\left(R^{f,\textsc{Solve}(f,.)}(1^n, y)\right)$ we have $f''(x') = f(x')$. To prove that Solve query is answered correctly observe that by f-obliviousness of the reduction the same TFNP instance is queried, i.e.,

$$Q_{\textsc{Solve}}\left(R^{f'',\textsc{SolveSim}(h,f',\cdot)}(1^n, y)\right) = Q_{\textsc{Solve}}\left(R^{f,\textsc{Solve}}(1^n, y)\right).$$

By Claim 4.4.2 the query to SolveSim is answered identically as it would be answered by Solve. Thus $R^{f'',\textsc{SolveSim}(h,f',\cdot)}(1^n, y) = R^{f,\textsc{Solve}(f,\cdot)}(1^n, y)$ and the preimage of y is set correctly. □

Finally we use Claim 4.4.3 to prove that $\textsc{Decode}_n$ correctly decodes the message of $\textsc{Encode}_n$ and outputs the right permutation.

Proof of Lemma 4.3.6. We assume that the first bit of the output of $\textsc{Encode}_n$ (Algorithm 7) is 1, otherwise a full description of π has been sent to $\textsc{Decode}_n$ (Algorithm 8) and the lemma holds trivially. Let $f = h \circ \pi$ similarly as in the $\textsc{Encode}_n$ algorithm (Algorithm 7). We have to show that $\textsc{Decode}_n$ (Algorithm 8) on line 15 holds function f' such that $f' = f$. We use Claim 4.4.3 to get that $f' \subseteq f$. Now observe that on line 15 of Algorithm 8 the function f' is defined for all inputs. Thus $f' = f$. Finally the $\textsc{Decode}_n$ algorithm returns a function $(h^{-1} \circ f) \restriction \{0,1\}^n = (h^{-1} \circ h \circ \pi) \restriction \{0,1\}^n = \pi$. □

4.4.3 Encoding Is Prefix-free

To be able to directly leverage coding theory we show that our encoding is in fact prefix-free. If the set of invertible images Y_f is small the encoding starts with a "0" bit followed by a trivial (prefix-free) encoding of the given permutation. Otherwise we show that the two codewords either differ at the beginning or are of the same length. Unique decodability gives us that such codewords must differ. In the proof we use the following claim (Claim 4.4.4) which computes the size of an encoding of a permutation π as returned by $\textsc{Encode}_n$ (Algorithm 7).

Claim 4.4.4. *The size of the encoding computed by $\textsc{Encode}_n$ (Algorithm 7) is*

1. $1 + \lceil \log(2^n!) \rceil$ *bits in case* Encode$_n$ *(Algorithm 7) returned* $\mathcal{M} = (0, \pi)$ *on line 11, and*

2. $1 + n + 2\left\lceil \log \binom{2^n}{|X_f|} \right\rceil + \lceil \log((2^n - |X_f|)!) \rceil$ *bits in case* Encode$_n$ *(Algorithm 7) returned a message* $\mathcal{M} = (1, |X_f|, Y_f, X_f, \sigma)$ *on line 14.*

Proof. When we say that we encode an index of a combinatorial object, we mean that we enumerate all instances of this object in some natural order and then encode the index of our particular instance. Specifically for the case of k-element subsets of a set of size n, we enumerate all such subsets in the lexicographical order and then just give an index – a number between one and $\binom{n}{k}$ always using precisely $\left\lceil \log \binom{n}{k} \right\rceil$ bits (we use leading zeroes as a padding in case there would be less bits). Similarly, for the case of permutations of n objects, we enumerate all such permutations in lexicographical order and then give the particular index using exactly $\lceil \log(n!) \rceil$ bits (again, potentially padding the index with leading zeroes).

1. In the first case (return on line 11) the encoding consists of the single "0" bit and an index of a permutation on all binary strings of length n. Thus we use $1 + \lceil \log(2^n!) \rceil$ bits.

2. In the second case (return on line 14) the encoding consists of a single "1" bit, a size of the set $X_f \subseteq \{0,1\}^n$ (this is always encoded by n bits), two sets $X_f \subseteq \{0,1\}^n$ and $Y_f \subseteq \mathrm{Im}(f_n)$ both of size $|X_f|$ thus encoded using $\left\lceil \log \binom{2^n}{|X_f|} \right\rceil$ bits each, and an index of a bijection between $\{0,1\}^n \setminus X_f$ and $\mathrm{Im}(f_n) \setminus Y_f$. Note that we do not have to encode the image of f_n as well as the domain and the image of σ, as those can be computed ($\mathrm{Im}(f_n) = \mathrm{Im}(h_n)$, domain of σ is the set $\{0,1\}^n \setminus X_f$ and $\mathrm{Im}(\sigma) = \mathrm{Im}(f_n) \setminus Y_f$). Thus we always use $1 + n + 2\left\lceil \log \binom{2^n}{|X_f|} \right\rceil + \lceil \log((2^n - |X_f|)!) \rceil$ bits for the encoding.

□

Now we use Claim 4.4.4 to show that the encoding is prefix-free.

Lemma 4.3.7. (Restated) *Let* $\mu \in \mathrm{T}$ *be any type* $h \in \mathit{Inj}_\mu$ *be any injective function and* $n \in \mathbb{N}$*, then the encoding given by the algorithm* Encode$_n$ *(Algorithm 7) is prefix-free, i.e.,*

$$\forall \pi, \pi' \in \mathit{Inj}_n^n \text{ such that } \pi \neq \pi' \colon \textsc{Encode}_n^h(\pi) \text{ is not a prefix of } \textsc{Encode}_n^h(\pi').$$

Proof. Let $\pi, \pi' \in \mathrm{Inj}_n^n$ be any permutations such that $\pi \neq \pi'$. Let $E_h(\pi) = \left|\textsc{Encode}_n^h(\pi)\right|$ and $E_h(\pi') = \left|\textsc{Encode}_n^h(\pi')\right|$. Note that by Lemma 4.3.6,

$$\textsc{Encode}_n^h(\pi) \neq \textsc{Encode}_n^h(\pi'),$$

otherwise we could not uniquely decode the permutation. First observe that if both $\textsc{Encode}^h_n(\pi), \textsc{Encode}^h_n(\pi')$ start with "0" bit, then

$$E_h(\pi) = 1 + \lceil \log(2^n!) \rceil = E_h(\pi').$$

Combining the facts that $\textsc{Encode}^h_n(\pi) \neq \textsc{Encode}^h_n(\pi')$, but $E_h(\pi) = E_h(\pi')$, we get that $\textsc{Encode}^h_n(\pi)$ is not a prefix of $\textsc{Encode}^h_n(\pi')$.

We have described the encoding formally in the proof of Claim 4.4.4. It suffices to prove the lemma for the case when both encodings $\textsc{Encode}^h_n(\pi)$ and $\textsc{Encode}^h_n(\pi')$ start with "1" bit. Note that the following n bits denote the size of $X_{h\circ\pi}$. Thus, if $|X_{h\circ\pi}| \neq |X_{h\circ\pi'}|$, the encodings $\textsc{Encode}^h_n(\pi)$ and $\textsc{Encode}^h_n(\pi')$ differ on the first $n+1$ bits and $\textsc{Encode}^h_n(\pi)$ is not a prefix of $\textsc{Encode}^h_n(\pi')$. On the other hand, if $|X_{h\circ\pi}| = |X_{h\circ\pi'}| = a$ then the lengths $E_h(\pi)$ and $E_h(\pi')$ are equal:

$$E_h(\pi) = 1 + n + 2\left\lceil \log \binom{2^n}{a} \right\rceil + \lceil \log((2^n - a)!) \rceil = E_h(\pi').$$

In combination with $\textsc{Encode}^h_n(\pi) \neq \textsc{Encode}^h_n(\pi')$, we get $\textsc{Encode}^h_n(\pi)$ is not a prefix of $\textsc{Encode}^h_n(\pi')$, which concludes the proof of the lemma. □

4.4.4 Bounding the Size of the Encoding

The most important part of the whole proof is bounding the expected size of our encoding. To show the upper bound it is crucial to bound the probability that a random y is in the "good" set G_f (see Algorithm 7, line 3), i.e., the set of challenges y where the reduction inverts and the solution returned by $\textsc{Solve}$ does not query a preimage of the challenge itself. We bound this probability in Claim 4.4.7. The bound will rely on the fact that C can do only a few queries to the function f. Unfortunately the number of queries C can make to f is upper bounded by the size of the input of C whereas for us it is useful to upper bound the number of queries with respect to the security parameter n which is given to the security reduction R on the input. Thus we use the following function as an upper bound on the number of queries and we upper bound its values in Claim 4.4.6.

Definition 4.4.5. *Let $\mu \in \mathrm{T}$ be any type and (R, T_R, C, T_C, p) be any deterministic f-oblivious many-one fully black-box construction of a worst-case hard* TFNP *problem from* `injective-OWF` *of type μ. By $q \colon \mathbb{N} \to \mathbb{N}$ we denote the function which upper bounds the number of queries which might be done indirectly, i.e.,*

$$q(n) = \max_{\substack{f \in Inj_\mu,\ x \in \{0,1\}^n \\ s \in \{0,1\}^*,\ |s| \le p(|i|)}} \left| Q\left(C^f(i,s)\right)\right|,$$

where i is the instance queried on $f(x)$, i.e., $Q_{\textsc{Solve}}\left(R^{f,\textsc{Solve}}(1^n, f(x))\right) = \{i\}$.

Note that the queries considered in the above definition are indirect (i.e., they originated in C and we do not consider f-queries made directly from R). We highlight that the definition takes into account all inputs (i, s) where s is any string of length at most $p(|i|)$, i.e., even on strings s for which $C^f(i,s) = 0$.

Lemma 4.3.8. (Restated) *Let $\mu \in \mathrm{T}$ be any type and (R, T_R, C, T_C, p) be a deterministic f-oblivious many-one fully black-box construction of a worst-case hard* TFNP *problem from an injective one-way function of type μ. Assume $n \in \mathbb{N}$ is large enough so that $n \geq 50$ and $2q(n) + 2 \leq 2^{0.2n}$, where $q(n)$ is the maximal number of f-queries made by C on the queried instance (see Definition 4.4.5). Let the success probability of R on security parameter n be $\beta \geq 2^{-0.1n}$, i.e., for any $f \in \mathit{Inj}_\mu$ we have*

$$\Pr_{x \leftarrow \{0,1\}^n}\left[R^{f,\textsc{Solve}}(1^n, f(x)) = x\right] = \beta \geq 2^{-0.1n}.$$

Then

$$\exists h \in \mathit{Inj}_\mu \colon \mathbb{E}_{\pi \leftarrow \mathit{Inj}_n^n}\left[\left|\textsc{Encode}_n^h(\pi)\right|\right] \leq \log(2^n!) - \frac{8}{10} n 2^{0.1n}.$$

In Claim 4.4.6, we show that for quasi-polynomial algorithms R, C and for large enough n the bound on $q(n)$ used in the statement of Lemma 4.3.8 is without loss of generality.

Claim 4.4.6. *Let $\mu \in \mathrm{T}$ be any type and (R, T_R, C, T_C, p) be any deterministic f-oblivious many-one fully black-box construction of a worst-case hard* TFNP *problem from* `injective-OWF` *of type μ. We can bound $q(n)$, i.e., the maximal number of f-queries made by C on any queried instance (see Definition 4.4.5), by*

$$q(n) \leq T_C\left(2p\left(T_R\left(101 n^{\log n}\right)\right)\right). \tag{4.4.5}$$

Moreover, if both $T_C, T_R \in O\left(n^{\mathrm{polylog}(n)}\right)$ then

$$q(n) \in o\left(2^{0.2n}\right). \tag{4.4.6}$$

Proof. Here we use the restriction that $\forall n \in \mathbb{N} \colon \mu(n) \leq 100 n^{\log n}$ which upper bounds output lengths of the injective function (see Notation 4.2.2). Note that these restrictions are reasonable. The case where the length of output of a function depends only on the length of the input is perhaps the most natural. It also makes sense to upper bound the length of the output as for exponential stretch the reduction would just query all possible preimages and thus would be able to invert on its own. On the other hand we cover all natural injective functions – with stretch c, where c is a fixed constant.

For any $x \in \{0,1\}^n$ the reduction $R^{f,\textsc{Solve}}(1^n, f(x))$ is running on input of length at most $n + 100 n^{\log n} \leq 101 n^{\log n}$ (see Notation 4.2.2) and thus can query the TFNP instance i of length at most $T_R\left(101 n^{\log n}\right)$. Thus any considered input of C (consisting of i and $s \in \{0,1\}^*$ of length at most $p(|i|)$) is of length at most $T_R\left(101 n^{\log n}\right) + p\left(T_R\left(101 n^{\log n}\right)\right) \leq 2p\left(T_R\left(101 n^{\log n}\right)\right)$. Finally, we can bound the number of queries of C by its running time by $T_C\left(2p\left(T_R\left(101 n^{\log n}\right)\right)\right)$. This concludes the proof of Equation (4.4.5).

Note that when both T_C and T_R are in $O\left(n^{\mathrm{polylog}(n)}\right)$, then also:

$$q(n) \leq T_C\left(2p\left(T_R\left(101 n^{\log n}\right)\right)\right) \in O\left(n^{\mathrm{polylog}(n)}\right) \subseteq o\left(2^{0.2n}\right),$$

which proves the Equation (4.4.6). □

We prove the upper bound on the expected length of the encoding (stated in Lemma 4.3.8) with expectation taken not only over the choice of π but also over the choice of $h \leftarrow \mathrm{Inj}_\mu$ (the expectation could be also understood to be taken over the choice of $f \leftarrow \mathrm{Inj}_\mu$ where $f = h \circ \pi$). Then we use an averaging argument to show that there exists $h \in \mathrm{Inj}_\mu$ for which the expected length of encoding is short enough, where the expectation is only over the choice of π, i.e., the message to be encoded.

Now we bound the probability that a given y is "good". That is the reduction returns the preimage of y and, moreover, the Solve query is answered with a solution which is independent of the preimage of y (i.e., there is no query to the preimage of y during computation $C^f(i, s)$, where i is the queried instance and s is the returned solution). We distinguish two bad events:

1. FAIL: the reduction does not invert and
2. HIT: the Solve solution queries the preimage.

The probability of FAIL is immediately upper bounded by $(1 - \beta)$, where β is the success probability of the security reduction. Thus we focus on bounding the probability of HIT event to bound the overall probability of y not being "good".

Bounding the probability of HIT: the event that Solve solution queries the preimage of y, the following two bounds will come handy. We fix the queried instance i, look at only one preimage length n and distinguish two cases:

1. There are many challenges $y \in \mathrm{Im}(f_n)$ for which the reduction queries the instance i. In this case the bound in Claim 4.4.9 will be used to show that the solution does not "help" the reduction too much.
2. Or there exist only few challenges $y \in \mathrm{Im}(f_n)$, such that i is queried. In this case we want to bound the probability that $Y_{i,n} = Z_i \cap \{0,1\}^n$ (i.e., the "protected" images of f_n as in Algorithm 6) becomes "unprotected" (i.e., removed from Y_i and the "benign" solution may query the preimage of some y from $Y_{i,n}$) before the solution is returned. By Claim 4.4.11 we bound the probability that there are only few challenges in $Z_i \cap \mathrm{Im}(f_n)$ but strings of length $\mu(n)$ are "unprotected" right from the start (deleted on line 4 of Algorithm 6 due to existence of many strings of length $\mu(n)$ which query the instance i). The probability that $Y_{i,n}$ was "protected" at the beginning of the algorithm but later removed from the set Y_i due to absence of "benign" solution (see lines 21 and 22 of Algorithm 6) is bounded by Claim 4.4.10.

Then after combining these bounds we derive the following upper bound:

Claim 4.4.7. *Let $\mu \in \mathrm{T}$ be any type and (R, T_R, C, T_C, p) be any deterministic f-oblivious many-one fully black-box construction of a worst-case hard TFNP problem from `injective-OWF` of type μ. Let $n \geq 16$ be any natural number and let $\pi \in \mathit{Inj}_n^n$ be any permutation. Let $q(n)$ be the maximal number of f-queries made by C on the queried instance (see Definition 4.4.5). Then*

$$\Pr[f(x) \notin G_f] \leq \frac{2(q(n)+1)}{2^{n/2}} + \Pr\left[R^{f,\textsc{Solve}}(1^n, y) \neq f^{-1}(y)\right],$$

where the probabilities are taken over the choice of $x \leftarrow \{0,1\}^n$, $h \leftarrow Inj_\mu$ *and both* $f = h \circ \pi$ *and* G_f *are as defined in the algorithm* ENCODE$_n$ *(see lines 1 and 3 of Algorithm 7).*

In the proof of the claim and supporting claims we leverage the following fact from probability theory.

Lemma 4.4.8. *Let* A *be a random event and let* $\mathcal{C} = \{C_1, C_2, \ldots, C_n\}$ *be a finite set of disjoint random events that cover the whole probability space. That is*

$$\forall i \neq j\colon\ \Pr[C_i \cap C_j] = 0 \wedge \Pr[C_1 \cup C_2 \cup \ldots \cup C_n] = 1.$$

We define $\Pr[A \mid C] = 0$ *whenever* $\Pr[C] = 0$*, then the following inequality holds:*

$$\Pr[A] \leq \max_{C \in \mathcal{C}} \Pr[A \mid C].$$

Proof. Since $\mathcal{C}$ forms a partition of the probability space we may write:

$$\begin{aligned} \Pr[A] &= \sum_{C' \in \mathcal{C}} \Pr[A \mid C'] \Pr[C'] \\ &\leq \sum_{C' \in \mathcal{C}} \left(\max_{C \in \mathcal{C}} \Pr[A \mid C] \right) \Pr[C'] \\ &= \max_{C \in \mathcal{C}} \Pr[A \mid C] \end{aligned}$$

□

The following claim (Claim 4.4.9) upper bounds the probability that SOLVE returns a solution querying the preimage of length n for some challenge y on which the reduction is running. We bound the probability for any fixed instance i. The bound is meaningful only for the case that on a given preimage length the reduction queries the instance i often (i.e., for many different preimages x, the reduction running on $f(x)$ queries the instance i).

In this case, we leverage the fact that there are at most $q(n)$ queries made on each solution. Thus no matter which solution is returned from the SOLVE algorithm, it queries at most $q(n)$ different preimages x. In other words the returned solution could be "useful" for the reduction for at most $q(n)$ out of many different challenges.

Claim 4.4.9. *Let* $\mu \in \mathrm{T}$ *be any type and* (R, T_R, C, T_C, p) *be any deterministic* f*-oblivious many-one fully black-box construction of a worst-case hard* TFNP *problem from* `injective-OWF` *of type* μ*. Let* $i \in \{0,1\}^*$ *be any instance of the corresponding* TFNP *problem (defined by* C*). Let*

$$q = \max \left| Q\left(C^f(i,s)\right) \right|,$$

where the maximum is taken over $f \in Inj_\mu$ *and* $s \in \{0,1\}^*$ *such that* $|s| \leq p(|i|)$*. Let* $k \in \mathbb{N}$*,* $Y \subseteq \{0,1\}^*$ *be a set of size* k *and*

$$\mathcal{F}_Y = \left\{ f \mid f \in Inj_\mu \wedge Y \subseteq Im(f) \right\}.$$

Then for any $f \in \mathcal{F}_Y$ *and for any* $s \in \{0,1\}^*$*, such that* $|s| \leq p(|i|)$*:*

$$\Pr_{y \leftarrow Y}\left[f^{-1}(y) \in Q\left(C^f(i,s)\right) \right] \leq \frac{q}{k}.$$

Proof. The probability follows from the assumption that for any $s \in \{0,1\}^*$ such that $|s| \leq p(|s|)$ the algorithm $C^f(i,s)$ makes at most q queries to f (i.e., $\left|Q\left(C^f(i,s)\right)\right| \leq q$). Thus

$$\Pr_{y \leftarrow Y}\left[f^{-1}(y) \in Q\left(C^f(i,s)\right)\right] \leq \frac{\left|Q\left(C^f(i,s)\right)\right|}{|Y|} \leq \frac{q}{k}.$$

□

Now we prove a second bound on the probability that for some instance i the algorithm Solve returns a solution which hits the preimage of some reduction's challenge $f(x)$. This bound is meaningful in the case when the reduction queries i only on a few challenges from $\mathrm{Im}(f_n)$, where n is the security parameter (i.e., $n = |x|$).

Recall that $Y_{i,n}$ is the set of challenges y whose preimages are of length n and on which the reduction queries the instance i, i.e., this bound is used when $Y_{i,n}$ is small. Note that if the returned solution queries the preimage of any y from $Y_{i,n}$, it may "help" the reduction non-trivially (imagine $Y_{i,n}$ containing only one element - then Solve may potentially reveal all preimages of length n over different instances).

We want to bound the probability that Solve stops to "protect" $Y_{i,n}$ before returning the solution (i.e., removes $Y_{i,n}$ from Y_i). We assume that $Y_{i,n}$ is protected before the first iteration of the while loop and removed only due to absence of a "benign" solution. That is we assume $\left|\{0,1\}^{\mu(n)} \setminus Z_i\right| \geq 2^n$ and thus strings of length $\mu(n)$ are not removed from the protected set on line 4 of Algorithm 6. The probability that this assumption is not satisfied is upper bounded separately by Claim 4.4.11.

Note that it is not sufficient to show that there is a solution which does not query the preimage of any $y \in Y_{i,n}$ for a single fixed preimage length n. As even though we have such a solution, there might be another $Y_{i,n'}$, such that its preimages are queried on all solutions. Then Solve would have to stop "protecting" $Y_{i,n'}$ to be able to return a solution. The problem occurs when Solve stops "protecting" $Y_{i,n}$ before $Y_{i,n'}$.

We show that if we stop "protecting" the sets $Y_{i,n}$ in the order given by decreasing density ($|Y_{i,n}|/2^n$) it is unlikely that we would remove a small set $Y_{i,n}$. Especially we bound the probability that there exists a solution which does not query preimage neither for any y from $Y_{i,n}$ nor for any y from $Y_{i,n'}$, where the density of $Y_{i,n'}$ is smaller (i.e., $|Y_{i,n'}|/2^{n'} \leq |Y_{i,n}|/2^n$).

To bound this probability, we consider another function $g \in \mathrm{Inj}_\mu$, which is similar to f, but we "hide" the preimages of all Y_i, resp. Z_i (the set of all "protected" images of different lengths). This means that for any $x \in f^{-1}[Y_i]$ we let $g(x) = y'$, where y' is chosen outside of the $\mathrm{Im}(f) \cup Z_i$ where Z_i is the set of all challenges querying the instance i. By the assumption that for any "protected" length n there are at least 2^n different strings in $\{0,1\}^{\mu(n)} \setminus Z_i$ we can always find such a function g. By correctness there is a solution with respect to g that cannot query the preimage of any y from Y_i (as Y_i is not in the image set). We show that with high probability this solution remains to be a solution and that it also does not query a preimage of any $y \in Y_i$ even when C is run with respect to f.

Claim 4.4.10. *Let $\mu \in \mathrm{T}$ be any type and (R, T_R, C, T_C, p) be any f-oblivious many-one fully black-box construction of a worst-case hard* **TFNP** *problem from* `injective-OWF` *of type μ. Let $i \in \{0,1\}^*$ be any instance of the corresponding* **TFNP** *problem and*

$$q = \max \left| Q\left(C^f(i,s)\right)\right|. \tag{4.4.7}$$

where the maximum is taken over $f \in Inj_\mu$ and $s \in \{0,1\}^*$ such that $|s| \leq p(|i|)$. Let $\delta\colon \mathbb{N} \to [0,1]$ be any function, $Z \subseteq \{0,1\}^*$ be a finite set, such that

$$\forall n \in \mathbb{N}\colon \left|\{0,1\}^{\mu(n)} \setminus Z\right| \geq 2^n, \tag{4.4.8}$$

$Y \subseteq Z$ such that

$$\forall n \in \mathbb{N}\colon \left|\{0,1\}^{\mu(n)} \cap Y\right| \leq \delta(n) 2^n \tag{4.4.9}$$

and $\mathcal{F}_Y$ be the set defined as follows

$$\mathcal{F}_Y = \left\{ f \in Inj_\mu \mid Im(f) \cap Z = Y \right\}. \tag{4.4.10}$$

Let BAD_f denote the event that

$$\forall s \in \{0,1\}^* \text{ such that } |s| \leq p(|i|)\colon$$
$$\text{either } C^f(i,s) \neq 1 \text{ or } Y \cap f\left[Q\left(C^f(i,s)\right)\right] \neq \emptyset.$$

Suppose $\mathcal{F}_Y$ is nonempty, then

$$\Pr_{f \leftarrow \mathcal{F}_Y}[BAD_f] \leq \sup_{n \in \mathbb{N}} \delta(n) q.$$

Intuitively the set Z corresponds to all challenges which query the instance i and the set Y contains only those which are in the image of f. The function δ upper-bounds the *density* of the set Z in the image of f.

Proof. For any $m \in \mathbb{N}$ let $Z_{(m)}$ denote the subset of Z which consists of all strings of length m, i.e., $Z_{(m)} = Z \cap \{0,1\}^m$ and similarly let $Y_{(m)} = Y \cap \{0,1\}^m$. We define the following set

$$\mathcal{H}_Z = \left\{ h \in \mathrm{Inj}_\mu \mid Z \cap \mathrm{Im}(h) = \emptyset \right\}. \tag{4.4.11}$$

Intuitively the set $\mathcal{H}_Z$ is the set of functions "avoiding" the set Z in their image. The set $\mathcal{F}_Y$ represents the set of functions "containing" exactly challenges Y from the set Z in their image (where the density of elements of Y, respectively Z, is bounded for all preimage lengths).

We define the set $X_{f,Z}$ to be the set of preimages of Z with respect to f (i.e., $X_{f,Z} = \{x \mid f(x) \in Z\}$). Now we can bound the probability using Lemma 4.4.8 as follows (we prove the inequality below):

$$\Pr_{f \leftarrow \mathcal{F}_Y}[\mathrm{BAD}_f] \leq \sup_{h \in \mathcal{H}_Z} \Pr_{f \leftarrow \mathcal{F}_Y}\left[\mathrm{BAD}_f \mid f \restriction \overline{X_{f,Z}} = h \restriction \overline{X_{f,Z}}\right] \tag{4.4.12}$$

First note that by the assumptions (see Equation 4.4.8) the set $\mathcal{H}_Z$ is not empty. Unfortunately $\mathcal{H}_Z$ contains uncountably many functions, whereas Lemma 4.4.8 assumes that $\mathcal{H}_Z$ is finite. We claim that we can consider all the functions only up to a certain input length. Since the instance i is fixed, the running time and thus also the length of any f-query made by the algorithm C is bounded by $T_C\left(|i| + p\left(|i|\right)\right)$. Thus the probability of BAD_f depends only on the choice of $f_1, f_2, \ldots, f_{T_C(|i|+p(|i|))}$ since longer inputs are ignored during the computation and make no difference in the probability. For the ease of presentation we slightly abuse the notation and use all functions from Inj_μ instead of restricting ourself to only short input lengths.

To prove the above inequality we show that for any function $f \in \mathcal{F}_Y$ there exists a function $h \in \mathcal{H}_Z$ such that $f \upharpoonright \overline{X_{f,Z}} = h \upharpoonright \overline{X_{f,Z}}$ and that the number of such functions depends only on Y and is independent of the exact matching f. Let $\mathcal{H}_{Z,f}$ be a set defined as follows:

$$\mathcal{H}_{Z,f} = \left\{h \in \mathcal{H}_Z \mid f \upharpoonright \overline{X_{f,Z}} = h \upharpoonright \overline{X_{f,Z}}\right\}.$$

We show that the set $\mathcal{H}_{Z,f}$ is nonempty and finite for any $f \in \mathcal{F}_Y$. Moreover we prove that for any $f, f' \in \mathcal{F}_Y$ the sets $\mathcal{H}_{Z,f}$ and $\mathcal{H}_{Z,f'}$ are of the same size. We compute the size of the set $\mathcal{H}_{Z,f}$ as follows:

$$|\mathcal{H}_{Z,f}| = \left|\left\{h \in \mathcal{H}_Z \mid f \upharpoonright \overline{X_{f,Z}} = h \upharpoonright \overline{X_{f,Z}}\right\}\right| \tag{4.4.13}$$

$$= \prod_{\substack{n\in\mathbb{N},\ m=\mu(n)\\ \text{s.t. } \{0,1\}^m \cap Z \neq \emptyset}} \left(\binom{2^m - 2^n - \left|Z_{(m)}\right| + \left|Y_{(m)}\right|}{\left|Y_{(m)}\right|}\left(\left|Y_{(m)}\right|!\right)\right). \tag{4.4.14}$$

First note that $2^m - 2^n - \left|Z_{(m)}\right| \geq 0$ by the assumptions on the size of $Z_{(m)}$ (see Equation (4.4.8)) and thus the binomial number is well defined standard binomial, i.e., the below integer is at most the above one. By definition, both $f, h \in \mathrm{Inj}_\mu$ (see Equations (4.4.10) and (4.4.11) - definition of $\mathcal{F}_Y$, respectively $\mathcal{H}_Z$), and furthermore $\mathrm{Im}(h) \cap Z = \emptyset$ (see Equation (4.4.11) for definition of $\mathcal{H}_Z$). Equation (4.4.14) follows from the fact that there are exactly $\binom{2^m-2^n-\left|Z_{(m)}\right|+\left|Y_{(m)}\right|}{\left|Y_{(m)}\right|}\left|Y_{(m)}\right|!$ possibilities how to define $h \upharpoonright \left(X_{f,Z} \cap \{0,1\}^n\right)$ as $\mathrm{Im}\left(h \upharpoonright X_{f,Z}\right) \cap \left(Z \cup \mathrm{Im}(f)\right) = \emptyset$ and

$$\left|\left(Z \cup \mathrm{Im}(f)\right) \cap \{0,1\}^m\right| = 2^n + \left|Z_{(m)}\right| - \left|Y_{(m)}\right|,$$

where $m = \mu(n)$. Note that size of $\mathcal{H}_{Z,f}$ is finite, as Z is a finite set and thus the product contains finitely many factors. Observe also that $|\mathcal{H}_{Z,f'}| = |\mathcal{H}_{Z,f}|$ as the size depends only on the type μ and the sizes of the sets Z and Y, but is independent of the exact mapping f.

It suffices to prove that for each $h \in \mathcal{H}_Z$:

$$\Pr_{f \leftarrow \mathcal{F}_Y}\left[\mathrm{BAD}_f \mid f \upharpoonright \overline{X_{f,Z}} = h \upharpoonright \overline{X_{f,Z}}\right] \leq \delta(n)q.$$

We fix any $h \in \mathcal{H}_Z$ and show the upper bound for the fixed function h. By the correctness of the underlying TFNP problem (see Definition 4.3.1 for the exact order of the quantifiers), there exists $s \in \{0,1\}^*$ such that $|s| \leq p(i)$ and $C^h(i,s) = 1$. Let Q_h be the set of queries made on the solution $C^h(i,s)$, i.e.,

$Q_h = Q\left(C^h(i,s)\right)$. Note that if $Q_h \cap X_{f,Z}$ is empty then all queries C makes on input (i,s) are answered equally for h as for f thus s is a solution also with respect to f (i.e., $C^f(i,s) = 1$) and $Q\left(C^f(i,s)\right)$ has empty intersection with $X_{f,Z}$ (as $Q\left(C^f(i,s)\right) = Q_h$). Thus using union bound we get the following probability:

$$\begin{aligned}
\Pr_{f \leftarrow \mathcal{F}_Y}\left[\mathrm{BAD}_f \mid f \restriction \overline{X_{f,Z}} = h \restriction \overline{X_{f,Z}}\right] &\leq \Pr_{f \leftarrow \mathcal{F}_Y}\left[X_{f,Z} \cap Q_h \neq \emptyset\right] \\
&\leq \sum_{n \in \mathbb{N}} \Pr_{f \leftarrow \mathcal{F}_Y}\left[X_{f,Z} \cap Q_h \cap \{0,1\}^n \neq \emptyset\right].
\end{aligned}$$

We bound the probability for a single preimage length n and the corresponding image length $m = \mu(n)$. Let q_n denote the number of queries of length n, i.e., $q_n = |Q_h \cap \{0,1\}^n|$ and K denotes $\min\left(2^n, |Z \cap \{0,1\}^m|\right)$. We may bound the probability as follows:

$$\begin{aligned}
&\Pr_{f \leftarrow \mathcal{F}_Y}\left[X_{f,Z} \cap Q_h \cap \{0,1\}^n \neq \emptyset\right] \\
&\qquad \leq \max_{k \in \{1,2,\ldots,K\}} \Pr_{f \leftarrow \mathcal{F}_Y}\left[X_{f,Z} \cap Q_h \cap \{0,1\}^n \neq \emptyset \mid |X_{f,Z} \cap \{0,1\}^n| = k\right] \\
&\qquad \leq \max_{k \in \{1,2,\ldots,K\}} \frac{|Q_h \cap \{0,1\}^n| \cdot \binom{2^n-1}{k-1}}{\binom{2^n}{k}} \\
&\qquad = \max_{k \in \{1,2,\ldots,K\}} \frac{k}{2^n} q_n
\end{aligned}$$

First inequality follows from the fact that condition $|X_{f,Z} \cap \{0,1\}^n| = k$ over different k creates a partition of the probabilistic space and we may use Lemma 4.4.8. Second inequality is computed as follows:

1. We have $\binom{2^n}{k}$ possibilities how to choose the set $X_{f,Z} \cap \{0,1\}^n$ as a subset of $\{0,1\}^n$ of size k.

2. But there are at most $|Q_h \cap \{0,1\}^n| \cdot \binom{2^n-1}{k-1}$ ways to choose the set in such a way that it has nonempty intersection with Q_h. We have to pick at least one x from $Q_h \cap \{0,1\}^n$ and the remaining $k-1$ elements can be chosen arbitrarily from the remaining $2^n - 1$ elements.

Using the definition of $\mathcal{F}_Y$ (see Equation 4.4.10), respectively Y (see Equation 4.4.9), to bound $\frac{k}{2^n} \leq \delta(n)$ we get the following bound:

$$\Pr_{f \leftarrow \mathcal{F}_Y}\left[X_{f,Z} \cap Q_h \cap \{0,1\}^n \neq \emptyset\right] \leq \sup_{k \in \{1,2,\ldots,K\}} \frac{k}{2^n} q_n \leq \delta(n) q_n.$$

Finally we summarize all the inequalities above:

$$\begin{aligned}\Pr_{f\leftarrow\mathcal{F}_Y}[\mathrm{BAD}_f] &\leq \sup_{h\in\mathcal{H}_Z}\Pr_{f\leftarrow\mathcal{F}_Y}\left[\mathrm{BAD}_f \mid f\restriction\overline{X_{f,Z}} = h\restriction\overline{X_{f,Z}}\right]\\ &\qquad\qquad\qquad\qquad\text{(see Equation (4.4.12))}\\ &\leq \sup_{h\in\mathcal{H}_Z}\sum_{n\in\mathbb{N}}\Pr_{f\leftarrow\mathcal{F}_Y}[X_{f,Z}\cap Q_h\cap\{0,1\}^n\neq\emptyset]\\ &\leq \sum_{n\in\mathbb{N}}\delta(n)q_n\\ &\leq \sup_{n\in\mathbb{N}}\delta(n)\sum_{n\in\mathbb{N}}q_n\\ &\leq \sup_{n\in\mathbb{N}}\delta(n)q.\end{aligned}$$

The last inequality follows by the definition of $q_n = |Q_h\cap\{0,1\}^n|$ and the assumption on the maximal number of queries made on a potential solution (see Equation 4.4.7) as we can bound

$$\sum_{n\in\mathbb{N}}q_n = \sum_{n\in\mathbb{N}}|Q_h\cap\{0,1\}^n| = |Q_h|\leq q.$$

□

We use the following claim, whenever the image of the function f contains only few challenges y we should protect, however there are too many challenges outside the image of f that should be protected. In this case we cannot use Claim 4.4.10 in particular because the assumption on Z (see Equation (4.4.8)) is not satisfied and thus there is no function for which the set of protected challenges of the given length would be empty. For this reason Solve never "protects" challenges of any length for which its input instance i is queried too often (see line 4 of Algorithm 6). We show that for a randomly chosen function from Inj_μ it is unlikely that its image contains only small number of such "protected" challenges. More specifically we show the following claim, where one can interpret the set A as the set of all possible strings of the fixed output length, the set B as all challenges on which the particular instance is queried and the set C as the image of the function $f\leftarrow\mathrm{Inj}_\mu$ with respect to the fixed output length. The assumptions on sizes a, b, c in the following claim are chosen to match the usage of the claim. More specifically from the fact that the function goes from n to at least $n+1$ bits and from the fact that there is no function which image avoids the "protected" challenges (i.e., any choice of C of size c must contain at least one element from B).

Claim 4.4.11. *Let $a, b, c\in\mathbb{N}$ be such that $a\geq 2c$ and $b\geq a-c+1$. Moreover assume $c\geq 16$. Let A be any set of size a and B be any subset of A of size b. Then*

$$\Pr\left[|C\cap B|\leq\sqrt{c}\right]\leq\frac{2}{\sqrt{c}},$$

where the probability is taken over C which is chosen uniformly at random from all subsets of A of size c.

Proof. In this proof all probabilities are taken over C which is chosen uniformly at random from all subsets of A of size c. Without loss of generality we can assume that $b = a - c + 1$ as the probability only decreases with larger b. We count the number of ways to choose a set containing exactly i elements from B and $c - i$ elements from $A \setminus B$. Using $a - b = c - 1$ (and thus also $i = |C \cap B| \geq 1$) we rewrite the probability as follows:

$$\begin{aligned}
\Pr_C\left[|C \cap B| \leq \sqrt{c}\right] &= \frac{\sum_{i=1}^{\sqrt{c}} \binom{b}{i}\binom{a-b}{c-i}}{\sum_{j=1}^{c} \binom{b}{j}\binom{a-b}{c-j}} \\
&= \frac{\sum_{i=1}^{\sqrt{c}} \binom{b}{i}\binom{c-1}{c-i}}{\sum_{j=1}^{c} \binom{b}{j}\binom{c-1}{c-j}} \\
&\leq \frac{\sum_{i=1}^{\sqrt{c}} \binom{b}{i}\binom{c-1}{c-i}}{\sum_{j=\sqrt{c}}^{c-\sqrt{c}} \binom{b}{j}\binom{c-1}{c-j}}.
\end{aligned}$$

Observe that

$$\begin{aligned}
\forall i \in \left\{1, 2, \ldots, \sqrt{c}\right\} &\colon \binom{c-1}{c-i} \leq \binom{c-1}{\sqrt{c}} \text{ and} \\
\forall j \in \left\{\sqrt{c}, \ldots, c - \sqrt{c}\right\} &\colon \binom{c-1}{c-j} \geq \binom{c-1}{\sqrt{c}}.
\end{aligned}$$

Note that by the assumption $b = a - c + 1 \geq c + 1$ we can also bound

$$\begin{aligned}
\forall i \in \left\{1, 2, \ldots, \sqrt{c}\right\} &\colon \binom{b}{i} \leq \binom{b}{\sqrt{c}} \text{ and} \\
\forall j \in \left\{\sqrt{c}, \ldots, c - \sqrt{c}\right\} &\colon \binom{b}{j} \geq \binom{b}{\sqrt{c}}.
\end{aligned}$$

Adding these bounds together and using the assumption that $c \geq 16$ we get the following:

$$\begin{aligned}
\Pr_C\left[|C \cap B| \leq \sqrt{c}\right] &\leq \frac{\sum_{i=1}^{\sqrt{c}} \binom{b}{i}\binom{c-1}{c-i}}{\sum_{j=\sqrt{c}}^{c-\sqrt{c}} \binom{b}{j}\binom{c-1}{c-j}} \\
&\leq \frac{\sqrt{c}\binom{b}{\sqrt{c}}\binom{c-1}{c-\sqrt{c}}}{(c - 2\sqrt{c})\binom{b}{\sqrt{c}}\binom{c-1}{c-\sqrt{c}}} \\
&\leq \frac{2}{\sqrt{c}} && (2\sqrt{c} \leq \tfrac{c}{2} \text{ for } c \geq 16)
\end{aligned}$$

which concludes the proof.

□

Now we are ready to prove Claim 4.4.7. That is we show that if the reduction is too successful it has to return a correct preimage for many challenges y even though their preimages are not queried on the solution obtained from SOLVE. Or equivalently show that a too successful reduction gives us a set G_f which is too large and thus can be later used to get too efficient encoding. Intuitively there is a trade-off based on the density: we use Claims 4.4.10 and 4.4.11 when the density is small and we use Claim 4.4.9 when the density is large enough.

Proof of Claim 4.4.7. Recall that $y \in \operatorname{Im}(f_n)$ is in G_f if it satisfies the following two conditions:

1. $R^{f,\textsc{Solve}}(1^n, y)$ inverts f on y, and
2. there is no indirect query for y (i.e., $f^{-1}(y) \notin Q_f^{\text{indir}}\left(R^{f,\textsc{Solve}}(1^n, y)\right)$).

We denote the event that R fails to invert f on y by $\mathsf{FAIL}_{f,y}$ (if this happens $y \notin \text{INV}_f$ see line 2 of Algorithm 7) and the event that there was an indirect query to the preimage of y as $\mathsf{HIT}_{f,y}$ (and thus y is not added to G_f see line 3 of Algorithm 7).

We bound the probability using union bound as follows:

$$\Pr_{\substack{x \leftarrow \{0,1\}^n,\ h \leftarrow \text{Inj}_\mu \\ f = h \circ \pi}} [f(x) \notin G_f] \leq \Pr_{\substack{x \leftarrow \{0,1\}^n,\ h \leftarrow \text{Inj}_\mu \\ f = h \circ \pi,\ y = f(x)}} [\mathsf{FAIL}_{f,y}] + \Pr_{\substack{x \leftarrow \{0,1\}^n,\ h \leftarrow \text{Inj}_\mu \\ f = h \circ \pi,\ y = f(x)}} [\mathsf{HIT}_{f,y}] .$$

Note that the probability of $\mathsf{FAIL}_{f,y}$ is one of the summands in the statement of the claim as

$$\Pr_{\substack{x \leftarrow \{0,1\}^n,\ h \leftarrow \text{Inj}_\mu \\ f = h \circ \pi,\ y = f(x)}} [\mathsf{FAIL}_{f,y}] = \Pr_{\substack{x \leftarrow \{0,1\}^n,\ h \leftarrow \text{Inj}_\mu \\ f = h \circ \pi}} \left[R^{f,\textsc{Solve}}\left(1^n, f(x)\right) \neq x\right] .$$

Thus we only need to show that

$$\Pr_{\substack{x \leftarrow \{0,1\}^n,\ h \leftarrow \text{Inj}_\mu \\ f = h \circ \pi,\ y = f(x)}} [\mathsf{HIT}_{f,y}] \leq \frac{2\left(q(n)+1\right)}{2^{n/2}}. \qquad (4.4.15)$$

Let us call P_n the set of possible TFNP instances that the reduction might query when running on some challenge y and a security parameter n:

$$P_n = \left\{ i \in \{0,1\}^* \mid \exists x \in \{0,1\}^n,\ \exists f \in \text{Inj}_\mu \colon i \in Q_{\textsc{Solve}}\left(R^{f,\textsc{Solve}}\left(1^n, f(x)\right)\right) \right\}.$$

Observe that the size of P_n is finite as the length of the challenge $f(x)$ and thus also the running time of R and the length of the longest queried instance is limited. Moreover since R is a deterministic many-one reduction the instance i is queried if and only if $\{i\} = Q_{\textsc{Solve}}\left(R^{f,\textsc{Solve}}\left(1^n, f(x)\right)\right)$. Which means that $\mathcal{C} = \left\{ i \in Q_{\textsc{Solve}}\left(R^{f,\textsc{Solve}}\left(1^n, f(x)\right)\right)\right\}_{i \in P_n}$ defines a partition of the probability space and we may use Lemma 4.4.8 to bound:

$$\Pr\left[\mathsf{HIT}_{f,y}\right] \leq \max_{i \in P_n} \Pr\left[\mathsf{HIT}_{f,y} \mid i \in Q_{\textsc{Solve}}\left(R^{f,\textsc{Solve}}\left(1^n, f(x)\right)\right)\right], \qquad (4.4.16)$$

where the probabilites are taken over the choice of $x \leftarrow \{0,1\}^n$, $h \leftarrow \text{Inj}_\mu$ and we define $f = h \circ \pi$, $y = f(x)$.

This allows us to fix any instance $i \in \{0,1\}^*$ and bound the probability that x was indirectly queried on the fixed TFNP instance i. Note that if the challenge $f(x)$ is contained in Y_i (see line 7 of Algorithm 6) when the solution is returned, respectively the set $Y_{i,n}$ is never removed before the algorithm $\textsc{Solve}$ returns, then the returned solution does not query the preimage of the challange $f(x)$. Thus we inspect whether the set $Y_{i,n}$ is still "protected" at the time when the algorithm returns. We use the following notation in the rest of the proof. Let

$t^{\mathrm{end}}_{i,f} = j$ if the algorithm $\textsc{Solve}^f(i)$ returns the solution in the j-th iteration of the while loop (see line 13 of Algorithm 6). Furthermore let $t^{\mathrm{del}}_{i,f,n}$ denote the iteration of the same while loop in which $Y_{i,n}$ is removed from Y_i (see line 20 of Algorithm 6). Note that we set $t^{\mathrm{del}}_{i,f,n} = \infty$ in the case when $Y_{i,n}$ is never removed from Y_i and we set $t^{\mathrm{del}}_{i,f,n} = 0$ if the set is removed already before the while loop, i.e., on line 4 of Algorithm 6.

Let $\delta\colon \mathbb{N} \to [0,1]$ be the function defined as $\delta(n) = \frac{1}{2^{n/2}}$ (the choice is explained at the end of the proof). We can rewrite the probability as follows:

$$\begin{aligned}\Pr\left[\mathsf{HIT}_{f,y}\right] = {}& \Pr\left[\mathsf{HIT}_{f,y} \;\middle|\; t^{\mathrm{end}}_{i,f} \le t^{\mathrm{del}}_{i,f,n}\right] \Pr\left[t^{\mathrm{end}}_{i,f} \le t^{\mathrm{del}}_{i,f,n}\right] + \\ &+ \Pr\left[\mathsf{HIT}_{f,y} \;\middle|\; t^{\mathrm{end}}_{i,f} > t^{\mathrm{del}}_{i,f,n} \wedge \frac{|Y_{i,n}|}{2^n} > \delta(n)\right] \Pr\left[t^{\mathrm{end}}_{i,f} > t^{\mathrm{del}}_{i,f,n} \wedge \frac{|Y_{i,n}|}{2^n} > \delta(n)\right] + \\ &+ \Pr\left[\mathsf{HIT}_{f,y} \;\middle|\; t^{\mathrm{end}}_{i,f} > t^{\mathrm{del}}_{i,f,n} \wedge \frac{|Y_{i,n}|}{2^n} \le \delta(n)\right] \Pr\left[t^{\mathrm{end}}_{i,f} > t^{\mathrm{del}}_{i,f,n} \wedge \frac{|Y_{i,n}|}{2^n} \le \delta(n)\right],\end{aligned} \tag{4.4.17}$$

the probabilities are taken over the choice of $h \leftarrow \mathrm{Inj}_\mu$, $x \leftarrow \{0,1\}^n$ such that $i \in Q_{\textsc{Solve}}\left(R^{f,\textsc{Solve}}(1^n, y)\right)$ holds, where $f = h \circ \pi$, $y = f(x)$ (i.e., the TFNP instance i is fixed and we choose uniformly at random x, h such that i is queried).

Observe that,

$$\Pr\left[\mathsf{HIT}_{f,y} \;\middle|\; t^{\mathrm{end}}_{i,f} \le t^{\mathrm{del}}_{i,f,n}\right] = 0$$

the probability is taken over the choice of $h \leftarrow \mathrm{Inj}_\mu$, $x \leftarrow \{0,1\}^n$ such that $i \in Q_{\textsc{Solve}}\left(R^{f,\textsc{Solve}}(1^n, y)\right)$ holds, where $f = h \circ \pi$, $y = f(x)$. This holds because the returned solution does not contain preimage of any element from Y_i and $y \in Y_i$ during the iteration $t^{\mathrm{end}}_{i,f}$.

Using this and bounding some of the probabilities by 1 we get the following bound:

$$\begin{aligned}\Pr\left[\mathsf{HIT}_{f,y}\right] \le \Pr\left[\mathsf{HIT}_{f,y} \;\middle|\; t^{\mathrm{end}}_{i,f} > t^{\mathrm{del}}_{i,f,n} \wedge \frac{|Y_{i,n}|}{2^n} > \delta(n)\right] + \\ + \Pr\left[t^{\mathrm{end}}_{i,f} > t^{\mathrm{del}}_{i,f,n} \wedge \frac{|Y_{i,n}|}{2^n} \le \delta(n)\right],\end{aligned}$$

the probabilities are taken over the choice of $h \leftarrow \mathrm{Inj}_\mu$, $x \leftarrow \{0,1\}^n$ such that $i \in Q_{\textsc{Solve}}\left(R^{f,\textsc{Solve}}(1^n, y)\right)$ holds, where $f = h \circ \pi$, $y = f(x)$.

We bound the first probability using Claim 4.4.9 for the instance i, $k = |Y_{i,n}| \ge \delta(n)2^n$, the set $Y = Y_{i,n}$ and $q(n)$ as the upper bound on the number of f-queries made by C. Thus we get:

$$\Pr\left[\mathsf{HIT}_{f,y} \;\middle|\; t^{\mathrm{end}}_{i,f} > t^{\mathrm{del}}_{i,f,n} \wedge \frac{|Y_{i,n}|}{2^n} > \delta(n)\right] \le \frac{q(n)}{k} \le \frac{q(n)}{\delta(n)2^n},$$

the probability is taken over the choice of $h \leftarrow \mathrm{Inj}_\mu$, $x \leftarrow \{0,1\}^n$ such that $i \in Q_{\textsc{Solve}}\left(R^{f,\textsc{Solve}}(1^n, y)\right)$ holds, where $f = h \circ \pi$, $y = f(x)$.

We can use Claims 4.4.10 and 4.4.11 to bound the second probability. We want to argue that it is unlikely that $Y_{i,n}$ is deleted before $\textsc{Solve}$ returns and $Y_{i,n}$ has a small density in the image of f. We distinguish two cases and bound the probability by Claim 4.4.11 if $t^{\mathrm{del}}_{i,f,n} = 0$, i.e., the case where the set was never protected by the algorithm Solve, and by Claim 4.4.10 if $t^{\mathrm{del}}_{i,f,n}$ is greater than 0, i.e., the set was deleted because there was no "benign" solution before its removal.

We first bound the probability that $\Pr\left[t^{\mathrm{del}}_{i,f,n} = 0 \wedge \frac{|Y_{i,n}|}{2^n} \leq \delta(n)\right]$. Observe that by our choice of $\delta(n) = \frac{1}{2^{n/2}}$ there are at most $\delta(n)2^n = 2^{n/2}$ strings in $Y_{i,n}$ since the set $Y_{i,n}$ is not dense in $\operatorname{Im}(f_n)$. Also note that there must be at least $2^{\mu(n)} - 2^n + 1$ strings of length $\mu(n)$ on which instance i is queried. This follows from the definition of the set Z_i in the algorithm Solve (see lines 1 and 4 of Algorithm 6). Let us denote the set of all strings of lenth $\mu(n)$ for which instance i is queried by $Z_{i,n}$. Thus we can rewrite the probability

$$\begin{aligned}\Pr\left[t^{\mathrm{del}}_{i,f,n} = 0 \wedge \frac{|Y_{i,n}|}{2^n} \leq \delta(n)\right] &= \Pr\left[|Z_{i,n}| \geq 2^{\mu(n)} - 2^n + 1 \wedge |Y_{i,n}| \leq 2^{n/2}\right] \\ &\leq \Pr\left[|Y_{i,n}| \leq 2^{n/2} \;\middle|\; |Z_{i,n}| \geq 2^{\mu(n)} - 2^n + 1\right] \\ &\leq \Pr\left[|\operatorname{Im}(h) \cap Z_{i,n}| \leq 2^{n/2} \;\middle|\; |Z_{i,n}| \geq 2^{\mu(n)} - 2^n + 1\right]\end{aligned}$$

the probabilities are taken over the choice of $h \leftarrow \mathrm{Inj}_{\mu}$, $x \leftarrow \{0,1\}^n$ such that $i \in Q_{\text{Solve}}\left(R^{f,\text{Solve}}(1^n, y)\right)$ holds, where $f = h \circ \pi$, $y = f(x)$. We use Claim 4.4.11 to show that such a choice of h is unlikely. We set $A = \{0,1\}^{\mu(n)}$ and B to be the set of all challenges of length $\mu(n)$ for which instance i is queried, i.e., $B = Z_{i,n}$. We let the set C represent the image of the function $f \restriction \{0,1\}^n$, thus we set $c = 2^n$. Note that the assumption on $b = |B|$ from Claim 4.4.11 is satisfied since the probability is conditioned on $|Z_{i,n}| \geq 2^{\mu(n)} - 2^n + 1$. Similarly the assumption on $c = |C| \geq 16$ follows directly from the assumptions of this claim. The claim gives us the following bound:

$$\Pr\left[|C \cap B| \leq 2^{n/2}\right] \leq \frac{2}{2^{n/2}}$$

where the probability is over a uniform distribution over all subsets $C \subset A$ of size c. Thus we bound

$$\Pr\left[t^{\mathrm{del}}_{i,f,n} = 0 \wedge \frac{|Y_{i,n}|}{2^n} \leq \delta(n)\right] \leq \frac{2}{2^{n/2}}.$$

From now we assume that $t^{\mathrm{del}}_{i,f,n} > 0$ and in particular there must be at most $2^{\mu(n)} - 2^n$ strings of length $\mu(n)$ on which instance i is queried by the definition of $t^{\mathrm{del}}_{i,f,n}$ and Z_i (see lines 1 and 4 of Algorithm 6). We need to determine the sets Y, Z we use in the Claim 4.4.10. There might be preimage lengths n' such that whenever $Y_{i,n}$ is "protected" $Y_{i,n'}$ is also "protected" (in particular all n' satisfying $|Y_{i,n'}|/2^{n'} \leq |Y_{i,n}|/2^n$). Thus we need to use Claim 4.4.10 for the set Y which is a union of all $Y_{i,n'}$ with small density (i.e., small $|Y_{i,n'}|/2^{n'}$). Let Y^{del}_i be the set Y_i as defined in the algorithm Solve in the iteration $t^{\mathrm{del}}_{i,f,n}$ (i.e., the iteration in which $Y_{i,n}$ is deleted). Similarly as in the algorithm Solve (see line 1 of Algorithm 6) we define the set Z_i to be the set of all strings which query the instance i, i.e.,

$$Z_i = \bigcup_{n=1}^{T_C(|i|+p(|i|))} \left\{y' \in \{0,1\}^{\mu(n)} \mid i \in Q_{\text{Solve}}\left(R^{f,\text{Solve}}(1^n, y')\right)\right\}.$$

Let Z_i^{del} be a subset of Z_i where we restrict the set only to strings of "relevant" lengths (lengths for which strings are still protected in the $t_{i,f,n}^{\mathrm{del}}$ iteration of the algorithm), i.e.,

$$Z_i^{\mathrm{del}} = \left\{ y' \in Z_i \mid Y_i^{\mathrm{del}} \cap \{0,1\}^{|y'|} \neq \emptyset \right\}.$$

In other words Z_i^{del} is a superset of Y_i^{del} which contains also relevant strings (i.e., those on which instance i is queried) which are outside the image of f.

We use Claim 4.4.10 where we set $Z = Z_i^{\mathrm{del}}$ and $Y = Y_i^{\mathrm{del}}$. First we prove that Y and Z satisfy the assumption of Claim 4.4.10. More specifically that Y can be partitioned into finitely many sets $Y_{i,n'}$, where $n' \in \mathbb{N}$, in such a way that

1. $Y = \bigcup_{n' \in \mathbb{N}} Y_{i,n'}$,
2. $\forall n' \in \mathbb{N}\ \forall y \in Y_{i,n'}\colon |y| = \mu(n')$, and
3. $\forall n' \in \mathbb{N}\colon |Y_{i,n'}| \leq \delta(n)/2^{n'}$.

Similarly Z can be partitioned into finitely many sets $Z_{i,n'}$, where $n' \in \mathbb{N}$, in such a way that

1. $Z = \bigcup_{n' \in \mathbb{N}} Z_{i,n'}$,
2. $\forall n' \in \mathbb{N}\ \forall z \in Z_{i,n'}\colon |z| = \mu(n')$, and
3. $\forall n' \in \mathbb{N}\colon \left|\{0,1\}^{\mu(n')} \setminus Z_{i,n'}\right| \geq 2^{n'}$.

Note that for our choice of both Y and Z these conditions are satisfied. The first two conditions for Y are satisfied trivially by setting each $Y_{i,n'}$ to the corresponding set in Solve. The third condition follows from the fact that we are in the iteration in which $Y_{i,n}$ should be removed from Y_i. We are removing the set $Y_{i,n}$ with the highest density and $|Y_{i,n}|/2^n \leq \delta(n)$ thus all other sets $Y_{i,n'} \subseteq Y_i^{\mathrm{del}}$ have density at most $\delta(n)$. Similarly for the set Z we can trivially satisfy the first two conditions by splitting Z_i^{del} into sets $Z_{i,n'} = Z_i^{\mathrm{del}} \cap \{0,1\}^{\mu(n')}$. The last condition follows from the fact that $t_{i,f,n}^{\mathrm{del}} > 0$ and thus we "protect" only strings of lengths which satisfy this inequality (see line 4 of Algorithm 6).

Thus we can use Lemma 4.4.8 and bound the probability as follows

$$\begin{aligned}
&\Pr\left[t_{i,f}^{\mathrm{end}} > t_{i,f,n}^{\mathrm{del}} > 0 \;\middle|\; \frac{|Y_{i,n}|}{2^n} \leq \delta(n) \right] \\
&\quad \leq \max_{Y,Z} \Pr\left[t_{i,f}^{\mathrm{end}} > t_{i,f,n}^{\mathrm{del}} > 0 \;\middle|\; \frac{|Y_{i,n}|}{2^n} \leq \delta(n) \wedge Z_i^{\mathrm{del}} = Z \wedge Y_i^{\mathrm{del}} = Y \right],
\end{aligned}$$

the probabilities are taken over the choice of $h \leftarrow \mathrm{Inj}_\mu$, $x \leftarrow \{0,1\}^n$ such that $i \in Q_{\textsc{Solve}}\left(R^{f,\textsc{Solve}}(1^n, y)\right)$ holds, where $f = h \circ \pi$, $y = f(x)$ and the maximum is over all sets Y, Z such that

$$Y \subseteq Z \subseteq \bigcup_{n'=1}^{T_C(|i|+p(|i|))} \{0,1\}^{\mu(n')}$$

satisfying the conditions described above (the condition on strings length copies that from algorithm Solve, see line 1 of Algorithm 6).

Now we fix any sets Y and Z, such that $Y \subseteq Z \subseteq \bigcup_{n'=1}^{T_C(|i|+p(|i|))} \{0,1\}^{\mu(n')}$ and both satisfy the conditions on existence of partitioning described above. We upper bound the probability using Claim 4.4.10, where we set the instace to i, the upper bound on the number of f-queries made by C to $q(n)$, the density of challenges y querying instance i to $\delta(n)$, the type of the function to μ, the sets of all protected challenges to our fixed set Z and its intersection with the image of the considered functions to Y. Recall the definition of $\mathcal{F}_Y$ from Claim 4.4.10:

$$\mathcal{F}_Y = \left\{ f \in \mathrm{Inj}_\mu \mid \mathrm{Im}(f) \cap Z = Y \right\}, \tag{4.4.18}$$

Observe that by our careful choice of sets Y and Z the set $\mathcal{F}_Y$ is not empty. Let us also recall the definition of BAD_f from Claim 4.4.10 which denotes the event that:

$$\begin{aligned} &\forall s \in \{0,1\}^* \text{ such that } |s| \le p(|i|): \\ &\qquad\qquad \text{either } C^f(i,s) \ne 1 \text{ or } Y \cap f\left[Q\left(C^f(i,s)\right)\right] \ne \emptyset. \end{aligned}$$

Claim 4.4.10 gives us that

$$\Pr_{f \leftarrow \mathcal{F}_Y}[\mathrm{BAD}_f] \le \delta(n)q(n).$$

Observe that for our choices of $i, q(n), \delta(n), \mu, Z$ and Y the event BAD_f corresponds to the situation that there was no "benign" solution before $Y_{i,n}$ was deleted from Y_i, i.e., it bounds the probability $t_{i,f}^{\mathrm{end}} > t_{i,f,n}^{\mathrm{del}} > 0$ and thus we get

$$\Pr\left[t_{i,f}^{\mathrm{end}} > t_{i,f,n}^{\mathrm{del}} > 0 \;\middle|\; \frac{|Y_{i,n}|}{2^n} \le \delta(n) \wedge Z_i^{\mathrm{del}} = Z \wedge Y_i^{\mathrm{del}} = Y \right] \le \delta(n)q(n),$$

the probability is taken over the choice of $h \leftarrow \mathrm{Inj}_\mu$, $x \leftarrow \{0,1\}^n$ such that $i \in Q_{\textsc{Solve}}\left(R^{f,\textsc{Solve}}(1^n, y)\right)$ holds, where $f = h \circ \pi$, $y = f(x)$.

Summarizing the above inequalities we get

$$\begin{aligned} \Pr[\mathsf{HIT}_{f,y}] \le \Pr&\left[t_{i,f}^{\mathrm{end}} > t_{i,f,n}^{\mathrm{del}} \wedge \frac{|Y_{i,n}|}{2^n} \le \delta(n) \right] + \\ &+ \Pr\left[\mathsf{HIT}_{f,y} \;\middle|\; t_{i,f}^{\mathrm{end}} > t_{i,f,n}^{\mathrm{del}} \wedge \frac{|Y_{i,n}|}{2^n} > \delta(n) \right] \\ \le \Pr&\left[t_{i,f,n}^{\mathrm{del}} = 0 \wedge \frac{|Y_{i,n}|}{2^n} \le \delta(n) \right] + \\ &+ \Pr\left[t_{i,f}^{\mathrm{end}} > t_{i,f,n}^{\mathrm{del}} > 0 \wedge \frac{|Y_{i,n}|}{2^n} \le \delta(n) \right] + \\ &+ \Pr\left[\mathsf{HIT}_{f,y} \;\middle|\; t_{i,f}^{\mathrm{end}} > t_{i,f,n}^{\mathrm{del}} \wedge \frac{|Y_{i,n}|}{2^n} > \delta(n) \right] \\ \le &\frac{2}{2^{n/2}} + \frac{q(n)}{\delta(n)2^n} + \delta(n)q(n), \end{aligned}$$

the probabilities are taken over the choice of $h \leftarrow \mathrm{Inj}_\mu$, $x \leftarrow \{0,1\}^n$ such that $i \in Q_{\textsc{Solve}}\left(R^{f,\textsc{Solve}}(1^n, y)\right)$ holds, where $f = h \circ \pi$, $y = f(x)$.

Finally as $\delta(n) = \frac{1}{2^{n/2}}$, we get

$$\begin{aligned}\Pr_{\substack{x\leftarrow\{0,1\}^n,\ h\leftarrow \mathrm{Inj}_\mu\\ f=h\circ\pi,\ y=f(x)}}[\mathsf{HIT}_{f,y}] &\le \frac{2}{2^{n/2}} + \frac{q(n)}{\delta(n)2^n} + \delta(n)q(n)\\ &= \frac{2\,(q(n)+1)}{2^{n/2}}.\end{aligned}$$

Note that δ is chosen in such a way that it minimizes the sum $\frac{q(n)}{\delta(n)2^n} + \delta(n)q(n)$. This concludes the proof of Claim 4.4.7 as we proved that

$$\begin{aligned}\Pr\ [f(x)\notin G_f] &\le \Pr[\mathsf{FAIL}_{f,y}] + \Pr[\mathsf{HIT}_{f,y}]\\ &\le \Pr\left[R^{f,\textsc{Solve}}(1^n,y) \ne f^{-1}(y)\right] + \frac{2\,(q(n)+1)}{2^{n/2}},\end{aligned}$$

where the probabilites are taken over the choice of $x \leftarrow \{0,1\}^n$, $h \leftarrow \mathrm{Inj}_\mu$ and we define $f = h\circ\pi$, $y = f(x)$. □

Finally, we upper bound the expected size of encoding produced by $\textsc{Encode}_n$ (see Algorithm 7) stated in Lemma 4.3.8.

Proof of Lemma 4.3.8. Let $E_h(\pi)$ denote the size of encoding, i.e., $E_h(\pi) = \left|\textsc{Encode}_n^h(\pi)\right|$ and let f be a random variable which corresponds to $h\circ\pi$ throughout the whole proof. First we bound the size of encoding for permutation π in the case that $\textsc{Encode}_n$ returned $\mathcal{M} = (1, |X_f|, Y_f, X_f, \sigma)$, i.e., $|X_f| \ge 2^{0.6n}$. We denote the size of X_f by A and set $N = 2^n$. As stated in Claim 4.4.4 the size of encoding from algorithm $\textsc{Encode}_n$ in case $A \ge 2^{0.6n}$ is

$$\begin{aligned}\mathbb{E}_{\pi\leftarrow\mathrm{Inj}_n^n,\ h\leftarrow\mathrm{Inj}_\mu}\left[E_h(\pi)\mid A\ge 2^{0.6n}\right] &= 1+n+2\left\lceil\log\left(\binom{N}{A}\right)\right\rceil + \lceil\log((N-A)!)\rceil\\ &\le 4+n+\log\left(\binom{N}{A}^2\cdot(N-A)!\right)\\ &= 4+n+\log(N!)+\log\left(\binom{N}{A}\frac{1}{A!}\right).\end{aligned}$$

We bound the last term as follows:

$$\begin{aligned}\log\left(\binom{N}{A}\frac{1}{A!}\right) &\le \log\left(\left(\frac{eN}{A}\right)^A\left(\frac{e}{A}\right)^A\right)\\ &= A\,(n+2\log e-2\log A).\end{aligned}$$

Recall that by assumption $2^{0.6n} \le A$ on the other hand $A = |X_f| \le 2^n$ as $X_f \subseteq \{0,1\}^n$. Now we differentiate by A and get

$$(A\,(n+2\log e-2\log A))' = n+2\log e-2\log A-\frac{2}{\ln 2}.$$

The derivative is negative and the function is decreasing for $A\in[2^{0.6n},2^n]$ and $n\ge 50$. We can bound

$$\begin{aligned}\log\left(\binom{N}{A}\frac{1}{A!}\right) &\le A\,(n+2\log e-2\log A)\\ &\le 2^{0.6n}\left(n+2\log e-2\log 2^{0.6n}\right) \quad \text{(decreasing for } A\in[2^{0.6n},2^n])\\ &\le 2^{0.6n}\,(-0.1n). \quad \text{(using } n\ge 50)\end{aligned}$$

If we combine the above bounds, we get the following bound on the size of the encoding for case $A \geq 2^{0.6n}$:

$$\begin{aligned}
\mathbb{E}_{\pi \leftarrow \mathrm{Inj}_n^n,\ h \leftarrow \mathrm{Inj}_\mu}\left[E_h(\pi) \mid A \geq 2^{0.6n}\right] &= 4 + n + \log(N!) + \log\left(\binom{N}{A}\frac{1}{A!}\right) \\
&\leq 2n + \log(N!) + 2^{0.6n}(-0.1n) \\
&\leq \log(N!) - n2^{0.2n}.
\end{aligned}$$

Now we bound the probability that $A = |X_f| \geq 2^{0.6n}$. By assumption of the lemma we can bound $(2q(n)+2) \leq 2^{0.2n}$ and get:

$$\begin{aligned}
\Pr_{\substack{\pi \leftarrow \mathrm{Inj}_n^n,\ h \leftarrow \mathrm{Inj}_\mu \\ f = h\circ\pi}}\left[|X_f| \geq 2^{0.6n}\right] &\geq \Pr\left[|G_f| \geq (2q(n)+1)\,2^{0.6n}\right] \\
&= \Pr\left[\left|\overline{G_f}\right| \leq 2^n - (2q(n)+1)\,2^{0.6n}\right] \\
&= \Pr\left[\left|\overline{G_f}\right| \leq 2^n - 2^{0.8n}\right]. \qquad (2q(n)+1 \leq 2^{0.2n})
\end{aligned}$$

By Claim 4.4.7 we can bound the expected size of $\overline{G_f}$ as follows:

$$\begin{aligned}
\mathbb{E}_{\substack{\pi \leftarrow \mathrm{Inj}_n^n,\ h \leftarrow \mathrm{Inj}_\mu \\ f = h\circ\pi}}\left[\left|\overline{G_f}\right|\right] &\leq \left(\frac{2\,(q(n)+1)}{2^{n/2}} + (1-\beta(n))\right)2^n \\
&\leq \left(\frac{2\,(q(n)+1)}{2^{n/2}} + \left(1 - 2^{-0.1n}\right)\right)2^n \qquad (\text{as } \beta(n) \geq 2^{-0.1n}) \\
&\leq 2\cdot 2^{0.7n} + 2^n - 2^{0.9n} \qquad (\text{as } 2q(n)+2 \leq 2^{0.2n}) \\
&\leq 2^n - \frac{9}{10}2^{0.9n}. \qquad (\text{as } n \geq 50)
\end{aligned}$$

We can use the Markov inequality to bound

$$\Pr_{\substack{\pi \leftarrow \mathrm{Inj}_n^n,\ h \leftarrow \mathrm{Inj}_\mu \\ f = h\circ\pi}}\left[\left|\overline{G_f}\right| > 2^n - 2^{0.8n}\right] \leq \frac{2^n - \frac{9}{10}2^{0.9n}}{2^n - 2^{0.8n}}.$$

Combining these inequalities we get:

$$\begin{aligned}
\Pr_{\substack{\pi \leftarrow \mathrm{Inj}_n^n,\ h \leftarrow \mathrm{Inj}_\mu \\ f = h\circ\pi}}\left[|X_f| \geq 2^{0.6n}\right] &\geq \Pr\left[\left|\overline{G_f}\right| \leq 2^n - 2^{0.8n}\right] \\
&= 1 - \Pr\left[\left|\overline{G_f}\right| > 2^n - 2^{0.8n}\right] \\
&\geq 1 - \frac{2^n - \frac{9}{10}2^{0.9n}}{2^n - 2^{0.8n}} \\
&\geq \frac{8}{10}2^{-0.1n}. \qquad (n \geq 50)
\end{aligned}$$

We are ready to bound the size of the encoding as follows:

$$\begin{aligned}
\mathbb{E}_{\pi\leftarrow \mathrm{Inj}_n^n,\ h\leftarrow \mathrm{Inj}_\mu}\left[E_h(\pi)\right] &= \Pr_{\pi,h}\left[|X_{\pi\circ h}| < 2^{0.6n}\right](1+\log N!) + \\
&\quad + \Pr_{\pi,h}\left[|X_{\pi\circ h}| \geq 2^{0.6n}\right]\mathbb{E}_{\pi,h}\left[E_h(\pi)\;\middle|\; |X_{\pi\circ h}| \geq 2^{0.6n}\right] \\
&\leq \Pr_{\pi,h}\left[|X_{\pi\circ h}| < 2^{0.6n}\right](1+\log N!) + \\
&\quad + \Pr_{\pi,h}\left[|X_{\pi\circ h}| \geq 2^{0.6n}\right]\left(\log(N!) - n2^{0.2n}\right) \\
&\leq 1 + \log(N!) - \Pr_{\pi,h}\left[|X_{\pi\circ h}| \geq 2^{0.6n}\right] n2^{0.2n} \\
&\leq 1 + \log(N!) - \left(\frac{8}{10}2^{-0.1n}\right) n2^{0.2n} \\
&\leq \log(N!) - \frac{8}{10}n2^{0.1n}.
\end{aligned}$$

Finally by averaging argument there exists a function $h \in \mathrm{Inj}_\mu$ such that $\mathbb{E}_{\pi\leftarrow \mathrm{Inj}_n^n}\left[E_h(\pi)\right] \leq \log(N!) - \frac{8}{10}n2^{0.1n}$. □

4.5 Extensions

In this section, we prove extensions of the result from Section 4.3.2. We first allow the security reduction to submit multiple queries to the oracle Solve as long as the queries are non-adaptive. This extension is a rather small modification of the previous proof and can be found in Section 4.5.1.

Then in Section 4.5.2 we further broaden our result by allowing even randomized non-adaptive reductions. Here non-trivial changes even for oracle Solve are needed. If we would define the sets Z_i, Y_i to be the sets of all challenges that ever query the instance (see lines 1 and 7 of Algorithm 6) and protect them the same way as before, we could run into troubles. This is because we would try to protect also the challenges for which the probability of querying the instance i would be very small. There could be many such challenges which could lead to absence of a "benign" solution and thus the algorithm would weaken the conditions on "benign" solution and stop to protect all challenges for a specific security parameter. But it might help the security reduction a lot if we give it a solution for a challenge y which is queried with high probability. To deal with this problem we stop protecting the challenges y in order of decreasing probability. For more details see Section 4.5.2.

4.5.1 Non-Adaptive Reductions

It is possible to extend our proof from Section 4.3.2 to rule out even *non-adaptive security reductions* which submit multiple queries to the oracle Solve in parallel, though still f-obliviously, as defined in Definition 4.3.3.

Notice that the algorithms Solve, Encode$_n$, Decode$_n$, and SolveSim (see Algorithms 6 to 9) are well defined even for non-adaptive reductions and we can use them without any change. Our analysis showing that Solve always returns a solution (Lemma 4.3.5), that Decode$_n$ correctly decodes the permutation (Lemma 4.3.6), and that the encoding is prefix-free (Lemma 4.3.7) remain unchanged as well. The statement of Lemma 4.3.8 also holds without a change.

The only difference in our proof would be in the proof of Claim 4.4.7 in Equation (4.4.16), where we are upper bounding the probability of an indirect hit. In the non-adaptive reductions, the right side upper bounds only the probability that ℓ-th query to Solve causes an indirect hit for some fixed ℓ (if we would define the set of instances i appropriately to correspond to instances queried as the ℓ-th query). Then we use union bound over ℓ to say that no query causes an indirect hit (similarly as we do in the black-box separation for average-case NP, see Chapter 3). Thus the maximum in Equation (4.4.16) needs to be multiplied by $T_R\left(n + 100n^{\log(n)}\right)$, since this upper bounds the running time of the security reduction R and thus also the number of Solve queries.

Accordingly, the bound on the indirect hit (Equation (4.4.15)) changes to

$$\Pr_{\substack{x \leftarrow \{0,1\}^n,\ h \leftarrow \mathrm{Inj}_\mu \\ f = h \circ \pi,\ y = f(x)}} [\mathsf{HIT}_{f,y}] \leq \frac{T_R\left(n + 100n^{\log(n)}\right) 2\left(q(n)+1\right)}{2^{n/2}}. \tag{4.5.1}$$

This is still $\mathcal{O}\left(n^{\mathrm{polylog}(n)}\right)$. Thus when we propagate this bound into the proof of Lemma 4.3.8 we can still bound the probability of indirect hit by $2^{0.2n}$ for large enough n. Therefore $\mathbb{E}_{\substack{\pi \leftarrow \mathrm{Inj}_n^n,\ h \leftarrow \mathrm{Inj} \\ f = h\circ\pi}}\left[\left|\overline{G_f}\right|\right]$ remains unchanged for large enough n and we can prove the following strengthening of Theorem 4.3.4.

Theorem 4.5.1. *Let $\mu \in \mathrm{T}$ be any type. There is no fully black-box construction (R, T_R, C, T_C, p) of a worst-case hard TFNP problem from injective one-way functions of type μ with a deterministic f-oblivious non-adaptive reduction with success probability at least $2^{-0.1n}$ such that both running times $T_R, T_C \in O\left(\ell^{polylog(\ell)}\right)$, where n is the security parameter and ℓ corresponds to the length of the input of R, resp. C.*

4.5.2 Randomized Reductions

In this section, we describe how to generalize our proof to handle fully black-box constructions of hard TFNP problems from `injective-OWF` with *randomized security reductions.* More precisely we focus on reductions which are randomized non-adaptive but still f-oblivious. We show the impossibility of such constructions too. Written formally we get the following theorem of the full strength:

Theorem 4.5.2. *Let $\mu \in \mathrm{T}$ be any type. There is no fully black-box construction (R, T_R, C, T_C, p) of a worst-case hard TFNP problem from injective one-way functions of type μ with a randomized f-oblivious non-adaptive reduction with success probability at least $2^{-0.1n}$ such that both running times $T_R, T_C \in O\left(\ell^{polylog(\ell)}\right)$, where n is the security parameter and ℓ corresponds to the length of the input of R, resp. C.*

Note that our algorithm Solve (as described in Algorithm 6) is not able to handle randomized reductions. One could imagine that the construction has for each challenge $y \in \{0,1\}^*$ created a specific instance $i_y \in \{0,1\}^*$ which is created in such a way that most solutions query the preimage of y. R when running on a challenge y queries i_y with high probability but to "hide" the real challenge it queries also some instances $i_{y'}$ for $y' \neq y$, each of them with small probability.

As there are many different challenges for which i_y is queried, we cannot apply Claim 4.4.10 to argue the existence of a "benign" solution. Thus SOLVE would probably stop to "protect" all challenges of length $|y|$ and return some solution – which with high probability helps R to invert on y. This would lead to a successful security reduction as a "useful" solution would be returned from SOLVE with high probability over the choice of queried instance.

The crux of the problem is that once solve encounters absence of a "benign" solution it stops to "protect" challenges in "batches" – in particular stops to "protect" all solutions of some length at once. This is not needed and SOLVE can rather give up on "protecting" the challenges one by one, taking the probability that the specific instance i (on which SOLVE runs) is queried into account. We give the precise description of the new algorithm SOLVE in Algorithm 10.

Solve as described in Algorithm 10 works properly: We don't give a full formal proof for this here, but by a similar argument as in Lemma 4.3.5 we can argue that it halts at the latest after $|Y_i| \leq 2^{T_C(|i|+p(|i|))+1}$ iterations of the while loop when there are no challenges present in Y_i. Thus SOLVE may return any solution at this step and at least one solution exists by the correctness of the construction (see Definition 4.3.1).

The only catch is that for every challenge we need to be able to compute the probability that the instance i is queried. But as we are interested only in challenges from $y \in \{0,1\}^{\mu(n)}$, where $n \in \{1, 2, \ldots T_C(|i| + p(|i|))\}$ we can for each such challenge simulate the reduction on $(1^n, y; r_1), (1^n, y; r_2), \ldots, (1^n, y; r_\ell)$ where strings $r_1, r_2, \ldots, r_\ell$ represent the random choices of R. Note that there are only finitely many strings $r_1, r_2, \ldots r_\ell$ we need to take into account as the running time of R is bounded by $T_R(n + |y|) = T_R(n + \mu(n)) \leq T_R\left(n + 100n^{\log n}\right)$. Thus we may consider only strings $r_1, r_2, \ldots r_\ell \in \{0,1\}^{T_R(n+\mu(n))}$ (if the reduction does less than this amount of random choices the remaining random bits are ignored) and we can still compute the probability precisely.

Encoding and decoding of a random permutation: The encoding and decoding of a random permutation $\pi \in \mathrm{Inj}_\mu$ can be again done by algorithms ENCODE$_n$, DECODE$_n$ (see Algorithms 7 and 8) with only small changes. First we need to use as a subroutine a new simulator SOLVESIM (previously described in Algorithm 9) to simulate our new SOLVE (described in Line 8). As all the changes are in the order in which SOLVE stops to "protect" the challenges it suffices to observe that the probabilities are independent on the function f (due to f-obliviousness of the reduction) and thus SOLVESIM can hold the same values as SOLVE does.

The second change to the algorithms is that we need to equip ENCODE$_n$ and DECODE$_n$ with a new source of shared randomness which will be used to simulate the security reduction. For this it suffices to choose one string $r_y \in \{0,1\}^{T_R(n+\mu(n))}$ per each challenge $y \in \mathrm{Im}(f_n) = \mathrm{Im}(h_n)$. Then both ENCODE$_n$ as well as DECODE$_n$ when simulating R on input $(1^n, y)$ use r_y for the random choices made by R. By the fact that $T_R(n + \mu(n))$ upper bounds the running time of R we know that no more random bits are needed. If only $\ell < T_R(n + \mu(n))$ random choices are made by R, only the first ℓ bits of the string r_y are used and the remaining bits are ignored.

Except the changes described above, the proofs of Lemmas 4.3.6 and 4.3.7 can be used for the randomized reductions as well. Thus we have correctness of the $\textsc{Encode}_n$,$\textsc{Decode}_n$ routines and we know that they produce prefix-free encoding. It suffices to upper bound the length of a produced encoding.

Bounding the length of the encoding: The crucial step of the proof is to bound the length of the resulting encoding, that is to show an equivalent of Lemma 4.3.8. We need to show (similarly as in Claim 4.4.7) that the probability that a "useful" solution was returned from $\textsc{Solve}$ is small. That is we want to show that with high probability over the choice of the challenge y there was not an indirect hit (the solutions returned from $\textsc{Solve}$ do not reveal the preimage of y). We include a full proof of Claim 4.5.3 an equivalent of Claim 4.4.7 for randomized reductions.

The rest of the proof can be done similarly as we bound the probability that a random challenge is in the good set by

$$\frac{2\,(q(n)+1)\,T_R\,(n+\mu(n))}{2^{n/2}} + \Pr\left[R^{f,\textsc{Solve}}(1^n, y; r) \neq f^{-1}(y)\right].$$

Since the numerator $2\,(q(n)+1)\,T_R\,(n+\mu(n)) \in O(n^{\text{polylog}(n)}) \subseteq o(2^{-0.2n})$ the proof of Lemma 4.3.8 can be left without change even for the randomized non-adaptive f-oblivious reductions and the proof of Theorem 4.5.2 follows by the same argumentation as in the proof of Theorem 4.3.4.

Claim 4.5.3. *Let $\mu \in \mathrm{T}$ be any type and (R, T_R, C, T_C, p) be any randomized deterministic f-oblivious non-adaptive fully black-box construction of a worst-case hard* `TFNP` *problem from* `injective-OWF` *of type μ. Let $n \geq 16$ be any natural number and let $\pi \in \mathit{Inj}_n^n$ be any permutation. Let $q(n)$ be the maximal number of f-queries made by C on any queried instance (see Definition 4.4.5). Then*

$$\Pr\left[f(x) \notin G_f\right] \leq \frac{2\,(q(n)+1)\,T_R\,(n+\mu(n))}{2^{n/2}} + \Pr\left[R^{f,\textsc{Solve}}(1^n, y; r) \neq f^{-1}(y)\right],$$

where the probabilities are taken over the randomness of $r \leftarrow \{0,1\}^{T_R(n+\mu(n))}$ of the security reduction R, the choice of $x \leftarrow \{0,1\}^n$, $h \leftarrow \mathit{Inj}_\mu$ and both $f = h \circ \pi$ and G_f are as defined in the algorithm $\textsc{Encode}_n$ (see lines 1 and 3 of Algorithm 7).

Note that in the statement of the claim we consider the randomness r of the security reduction to be only the strings of length $\{0,1\}^{T_R(n+\mu(n))}$. As we already mentioned this can be done without loss of generality as R cannot make more than this amount of random coin tosses (by the fact that the running time of R is bounded) and if it does less we can just ignore the last bits as unused random coin tosses.

Also note that the function $q(n)$ (Definition 4.4.5) was defined only for deterministic many-one reduction, i.e., in the definition we consider the exact one choice of the instance i which is queried by security reduction. We need to slightly alter the definition in the case of randomized non-adaptive reduction to choose any instance i which is queried with non zero probability and take the maximum also over the choice of such an instance i.

The proof follows the blueprint of the proof of Claim 4.4.7 the main differences are that we have to take the probabilities of querying a particular instance into account and the changes described in the Section 4.5.1 which allow us to deal with the non-adaptivity of the security reduction. As the changes are non-trivial we include the full proof with the aforementioned changes.

Proof. Recall that $y \in \mathrm{Im}\left(f_n\right)$ is in G_f if it satisfies the following two conditions:

1. $R^{f,\textsc{Solve}}(1^n, y; r)$ inverts f on y, and
2. there is no indirect query for y (i.e., $f^{-1}(y) \notin Q_f^{\mathrm{indir}}\left(R^{f,\textsc{Solve}}(1^n, y; r)\right)$).

We denote the event that R fails to invert f on y by $\mathsf{FAIL}_{f,y}$ (if this happens $y \notin \mathrm{INV}_f$ see line 2 of Algorithm 7) and the event that there was a indirect query to the preimage of y as $\mathsf{HIT}_{f,y}$ (and thus y is not added to G_f see line 3 of Algorithm 7). We bound the probability using union bound as follows:

$$\Pr\left[f(x) \notin G_f\right] \leq \Pr\left[\mathsf{FAIL}_{f,y}\right] + \Pr\left[\mathsf{HIT}_{f,y}\right],$$

where the probability is over the randomness $r \leftarrow \{0,1\}^{T_R(n+\mu(n))}$, $h \leftarrow \mathrm{Inj}_\mu$ and $x \leftarrow \{0,1\}^n$, f denotes the function $h \circ \pi$ and $y = f(x)$.

Note that the probability of $\mathsf{FAIL}_{f,y}$ is one of the summands in the statement of the claim as

$$\Pr\left[\mathsf{FAIL}_{f,y}\right] = \Pr\left[R^{f,\textsc{Solve}}\left(1^n, f(x); r\right) \neq x\right].$$

where the probability is over the randomness $r \leftarrow \{0,1\}^{T_R(n+\mu(n))}$, $h \leftarrow \mathrm{Inj}_\mu$ and $x \leftarrow \{0,1\}^n$, f denotes the function $h \circ \pi$ and $y = f(x)$. Thus we only need to show that

$$\Pr\left[\mathsf{HIT}_{f,y}\right] \leq \frac{2\left(q(n)+1\right)T_R\left(n+\mu(n)\right)}{2^{n/2}}. \tag{4.5.2}$$

where the probability is over the randomness $r \leftarrow \{0,1\}^{T_R(n+\mu(n))}$, $h \leftarrow \mathrm{Inj}_\mu$ and $x \leftarrow \{0,1\}^n$, f denotes the function $h \circ \pi$ and $y = f(x)$.

From now we assume that there is only one query made to Solve. To emphasize this the event $\mathsf{HIT}^{(1)}_{f,y}$ corresponds to the event that there was an indirect query to a challenge y assuming that the security reduction always queries Solve on at most one instance. Observe that if we bound the probability of this event:

$$\Pr\left[\mathsf{HIT}^{(1)}_{f,y}\right] \leq \frac{2\left(q(n)+1\right)}{2^{n/2}}, \tag{4.5.3}$$

then

$$\Pr\left[\mathsf{HIT}_{f,y}\right] \leq \frac{2\left(q(n)+1\right)T_R\left(n+\mu(n)\right)}{2^{n/2}},$$

where the probabilities are in both cases over the choice of $r \leftarrow \{0,1\}^{T_R(n+\mu(n))}$, $x \leftarrow \{0,1\}^n$, $h \leftarrow \mathrm{Inj}_\mu$, f denotes the function $h \circ \pi$ and $y = f(x)$. This follows from the non-adaptivity of the queries because any reduction R for which $\mathsf{HIT}_{f,y}$ has larger probability gives us a 1-query reduction R' where $\mathsf{HIT}^{(1)}_{f,y}$ has larger probability than possible by Equation (4.5.3) as follows: R' on the input $(1^n, y; r)$ chooses uniformly at random one query i from $Q_{\textsc{Solve}}\left(R^{f,\textsc{Solve}}(1^n, y; r)\right)$. As the running time (and thus also the number of queries made by R) is upper bounded by $T_R(n + \mu(n))$, we get

$$\Pr\left[\mathsf{HIT}^{(1)}_{f,y} \text{ occurs for } R'\right] \geq \frac{\Pr\left[\mathsf{HIT}_{f,y} \text{ occurs for } R\right]}{T_R\left(n + \mu(n)\right)},$$

where the probabilities are over the choice of $r \leftarrow \{0,1\}^{T_R(n+\mu(n))}$, $x \leftarrow \{0,1\}^n$, $h \leftarrow \mathrm{Inj}_\mu$, f denotes the function $h \circ \pi$ and $y = f(x)$. Thus we can restrict ourselves to security reductions which submit at most one query to Solve.

Similarly as in the proof of Claim 4.4.7 we now want to fix one instance $i \in \{0,1\}^*$ and bound the probability of $\mathsf{HIT}^{(1)}_{f,y}$ assuming i is the queried instance. Let P_n denote the set of possible TFNP instances that the reduction might query when running on some challenge $y \in \{0,1\}^{\mu(n)}$ and a security parameter n:

$$P_n = \left\{ i \in \{0,1\}^* \;\middle|\; \begin{array}{c} i \in Q_{\textsc{Solve}}\left(R^{f,\textsc{Solve}}\left(1^n, f(x); r\right)\right) \\ \text{for some } x \in \{0,1\}^n, r \in \{0,1\}^{T_R(n+\mu(n))}, f \in \mathrm{Inj}_\mu \end{array} \right\}.$$

Observe that the size of P_n is finite as the length of the challenge $f(x)$ and thus also the running time of R and the length of the longest queried instance are limited. Moreover $\mathcal{C} = \left\{ i \in Q_{\textsc{Solve}}\left(R^{f,\textsc{Solve}}\left(1^n, f(x); r\right)\right)\right\}_{i \in P_n}$ defines a partition of the probability space since we assume that R makes at most one query to Solve. Thus we may use Lemma 4.4.8 to bound:

$$\Pr\left[\mathsf{HIT}^{(1)}_{f,y}\right] \leq \max_{i \in P_n} \Pr\left[\mathsf{HIT}^{(1)}_{f,y} \mid i \in Q_{\textsc{Solve}}\left(R^{f,\textsc{Solve}}\left(1^n, f(x); r\right)\right)\right], \qquad (4.5.4)$$

where the probabilities are in both cases over the choice of $r \leftarrow \{0,1\}^{T_R(n+\mu(n))}$, $x \leftarrow \{0,1\}^n$, $h \leftarrow \mathrm{Inj}_\mu$, f denotes the function $h \circ \pi$ and $y = f(x)$.

This allows us to fix any instance $i \in \{0,1\}^*$ and bound the probability that x was indirectly queried on this fixed instance i. Note that if the challenge $f(x)$ is contained in Y_i (see line 8 of Algorithm 10) when the solution is returned, then the returned solution does not query the preimage of the challange $f(x)$. Thus we inspect whether the challenge y is still in $Y_{i,n} \subseteq Y_i$ and thus still "protected" at the time when the algorithm returns. We use the following notation in the rest of the proof. Let $t^{\mathrm{end}}_{i,f} = j$ if the algorithm $\textsc{Solve}^f(i)$ returns the solution in the j-th iteration of the while loop (see line 14 of Algorithm 10). Furthermore let $t^{\mathrm{del}}_{i,f,n,y}$ denote the iteration of the same while loop in which y is removed from Y_i, respectively $Y_{i,n}$ (see lines 21 and 22 of Algorithm 10). Note that we set $t^{\mathrm{del}}_{i,f,n,y} = \infty$ in the case when y is never removed from $Y_{i,n}$ neither from Y_i and we set $t^{\mathrm{del}}_{i,f,n,y} = 0$ if the set is removed already before the while loop, i.e., on line 5 of Algorithm 10.

We distinguish the following two "bad" events when we deleted a challenge from the protected set, but returning its preimage can potentially "help" the reduction too much:

$\mathsf{BADD}^{(1)}_{i,f,n}$: There is a challenge y from $\{0,1\}^{\mu(n)}$ which is unprotected from the start of the algorithm (i.e., $U_{i,n} \neq \emptyset$, see line 4 of Algorithm 10), but

$$|Z_i| < 2^{n/2}$$

where Z_i is as computed on line 1 of Algorithm 10, i.e., Z_i contains *all* challenges from $\{0,1\}^{\mu(n)}$ which query the instance i.

$\mathsf{BADD}^{(2)}_{i,f,n}$: The algorithm removed a challenge $y \in \{0,1\}^{\mu(n)}$ from $Y_{i,n}$, respectively Y_i, due to absence of a "benign" solution in an iteration of the while loop in which $|Y_{i,n}| \leq 2^{n/2}$.

We bound the probabilities of these two events and then show that if none of them occurs the returned solution does not cause an indirect hit with high probability.

Similarly as in the proof of Claim 4.4.7, we can bound the probability of the first event using Claim 4.4.11 for $A = \{0,1\}^{\mu(n)}$, B being the set of challenges from $\{0,1\}^{\mu(n)}$ which have zero probability of querying the instance i and C being the image of f_n, respectively h_n. As we choose $h \leftarrow \mathrm{Inj}_\mu$, the image f_n is chosen from all subsets of A of size 2^n uniformly at random and we know that $\mu(n) > n$ thus $|A| \geq 2|C|$. The inequality $|B| \geq |A| - |C| - 1$ follows from the fact that Z_i contained more than $2^{\mu(n)} - 2^n$ challenges, otherwise $U_{i,n} = \emptyset$. Thus we can bound the probability of $\mathsf{BADD}^{(1)}_{i,f,n}$ by:

$$\Pr\left[\mathsf{BADD}^{(1)}_{i,f,n}\right] \leq \frac{2}{2^{n/2}}, \tag{4.5.5}$$

where the probability is over the choice of $r \leftarrow \{0,1\}^{T_R(n+\mu(n))}$, $x \leftarrow \{0,1\}^n$, $h \leftarrow \mathrm{Inj}_\mu$ and f denotes the function $h \circ \pi$.

To bound $\mathsf{BADD}^{(2)}_{i,f,n}$ we use Claim 4.4.10 similarly as in the proof Claim 4.4.7. Let Y_i^{del} denote the set Y_i during the iteration in which $\mathsf{BADD}^{(2)}_{i,f,n}$ occurs (for the first time) and Z_i^{del} be the set Z_i after we potentially deleted few challenges from it, i.e., after the deletions made on line 5 of Algorithm 10. We observe that by Lemma 4.4.8 the probability can be bounded by

$$\Pr\left[\mathsf{BADD}^{(2)}_{i,f,n}\right] \leq \max_{Y,Z} \Pr\left[\mathsf{BADD}^{(2)}_{i,f,n} \;\middle|\; Y_i^{\mathrm{del}} = Y \ \wedge\ Z_i = Z\right]$$

where the probability is over the choice of $r \leftarrow \{0,1\}^{T_R(n+\mu(n))}$, $x \leftarrow \{0,1\}^n$, $h \leftarrow \mathrm{Inj}_\mu$ and f denotes the function $h \circ \pi$ and the maximum is over

$$Y \subseteq Z \subseteq \bigcup_{n=1}^{T_R(n+\mu(n))} \{0,1\}^{\mu(n)},$$

such that for every $n' \in \mathbb{N}$: $|Y \cap \{0,1\}^{n'}| \leq 2^{n'}/2^{n/2}$ and $|Z| \leq 2^{\mu(n')} \setminus 2^{n'}$. Note that in the iteration in which $\mathsf{BADD}^{(2)}_{i,f,n}$ occurs the conditions on Z and Y are satisfied as:

$|Z_i| \leq 2^{\mu(n')} \setminus 2^{n'}$: On line 5 of Algorithm 10 we add challenges to Z_i making sure that this condition is satisfied.

$|Y_i^{\mathrm{del}} \cap \{0,1\}^{n'}| \leq 2^{n'}/2^{n/2}$: By the definition of $\mathsf{BADD}^{(2)}_{i,f,n}$ we removed a challenge from $Y_{i,n}$ which contains at most $2^{n/2}$ challenges. Since we choose $Y_{i,n}$ from which we delete the challenge in the order of decreasing density (see lines 21 and 22 of Algorithm 10) each other set must have density at most $2^{n/2}/2^n = 1/2^{n/2}$.

Then the fact that $\mathsf{BADD}^{(2)}_{i,f,n}$ occurred implies that there was no "benign" solution before deleting another challenge from Y even though the density of "protected" challenges was already small (smaller than $1/2^{n/2}$). Thus using Claim 4.4.10 we get:

$$\Pr\left[\mathsf{BADD}^{(2)}_{i,f,n}\right] \le \max_{Y,Z} \Pr\left[\mathsf{BADD}^{(2)}_{i,f,n} \;\middle|\; Y_i^{\mathrm{del}} = Y \ \wedge\ Z_i = Z\right] \tag{4.5.6}$$

$$\le \frac{q(n)}{2^{n/2}} \tag{4.5.7}$$

where the probability is over the choice of $r \leftarrow \{0,1\}^{T_R(n+\mu(n))}$, $x \leftarrow \{0,1\}^n$, $h \leftarrow \mathrm{Inj}_\mu$ and f denotes the function $h \circ \pi$ and the maximum is over

$$Y \subseteq Z \subseteq \bigcup_{n=1}^{T_R(n+\mu(n))} \{0,1\}^{\mu(n)},$$

such that for every $n' \in \mathbb{N}$: $|Y \cap \{0,1\}^{n'}| \le 2^{n'}/2^{n/2}$ and $|Z| \le 2^{\mu(n')} \setminus 2^{n'}$.

If neither $\mathsf{BADD}^{(1)}_{i,f,n}$ nor $\mathsf{BADD}^{(2)}_{i,f,n}$ occurs but we still got a solution which queries the preimage of our challenge $y \in \{0,1\}^{\mu(n)}$ (i.e., causes an indirect query) there must be a set $Y^{\mathrm{del}}_{i,n}$ of at least $2^{n/2}$ different challenges from the image of f_n for which the reduction queries the instance i with non-zero probability and with respect to which a "benign" solution was returned from SOLVE. This is beacuse the solution can query only the challenges which are not "protected" at that point of time. To be unprotected we have to delete the challenge y either from Z_i on line 5 of Algorithm 10 or in Y_i and $Y_{i,n}$ on lines 21 and 22 of Algorithm 10. In the first case the fact that $\mathsf{BADD}^{(1)}_{i,f,n}$ does not occur implies existence of at least $2^{n/2}$ challenges in $\mathrm{Im}(f_n)$ which query i with non-zero probability. In the second case removing the challenge y and keeping less than $2^{n/2}$ solutions "protected" corresponds to an occurence of $\mathsf{BADD}^{(2)}_{i,f,n}$.

Moreover as we have been removing the challenges from $Y_{i,n}$ and thus also from Y_i in the order of minimal probability of querying i, we can bound

$$\Pr\left[\mathsf{HIT}^{(1)}_{f,y} \;\middle|\; \neg\mathsf{BADD}^{(1)}_{i,f,n} \wedge \neg\mathsf{BADD}^{(2)}_{i,f,n}\right] \le \frac{q(n)}{2^{n/2}}$$

where the probability is over the choice of $r \leftarrow \{0,1\}^{T_R(n+\mu(n))}$, $x \leftarrow \{0,1\}^n$, $h \leftarrow \mathrm{Inj}_\mu$, f denotes the function $h \circ \pi$ and $y = f(x)$.

Thus using union bound and combining the above inequality with already proven Equations (4.5.5) and (4.5.7) we get:

$$\begin{aligned}
&\Pr\left[\mathsf{HIT}^{(1)}_{f,y}\right] \\
&\quad \le \Pr\left[\mathsf{BADD}^{(1)}_{i,f,n}\right] + \Pr\left[\mathsf{BADD}^{(2)}_{i,f,n}\right] + \Pr\left[\mathsf{HIT}^{(1)}_{f,y} \;\middle|\; \neg\mathsf{BADD}^{(1)}_{i,f,n} \wedge \neg\mathsf{BADD}^{(2)}_{i,f,n}\right] \\
&\quad \le \frac{2}{2^{n/2}} + \frac{q(n)}{2^{n/2}} + \frac{q(n)}{2^{n/2}} \\
&\quad \le \frac{2(q(n)+1)}{2^{n/2}}
\end{aligned}$$

where the probabilities are over the choice of $r \leftarrow \{0,1\}^{T_R(n+\mu(n))}$, $x \leftarrow \{0,1\}^n$, $h \leftarrow \mathrm{Inj}_\mu$, f denotes the function $h \circ \pi$ and $y = f(x)$. $\square$

Algorithm 10: The oracle SOLVE.

```
Hardwired      : a randomized $f$-oblivious non-adaptive fully black-box
                 construction $(R, T_R, C, T_C, p)$ of a worst-case hard TFNP
                 problem from injective-OWF of type $\mu$
Oracle access: an injective function $f = \{f_n\}_{n\in\mathbb{N}} \in \mathrm{Inj}_\mu$
Input          : an instance $i \in \{0,1\}^*$
Output         : a solution $s \in \{0,1\}^*$ such that $C^f(i,s) = 1$ and $|s| \le p(|i|)$

1  Compute
     $Z_i = \bigcup_{n=1}^{T_C(|i|+p(|i|))} \left\{ y \in \{0,1\}^{\mu(n)} \,\middle|\, i \in Q_{\mathrm{SOLVE}}\left(R^{f,\mathrm{SOLVE}}(1^n, y; r)\right) \text{ for some } r \right\}$
2  for $n \in \{1, \ldots, T_C(|i| + p(|i|))\}$ do
3      if $\left|\{0,1\}^{\mu(n)} \setminus Z_i\right| < 2^n$ then
4          Compute $U_{i,n} \subseteq Z_i \cap \{0,1\}^{\mu(n)}$ containing exactly $2^n - \left|\{0,1\}^{\mu(n)} \setminus Z_i\right|$
             challenges for which probability of querying $i$ is minimal
5          Set $Z_i = Z_i \setminus U_{i,n}$
6      end
7  end
8  Compute $Y_i = Z_i \cap \mathrm{Im}(f)$
9  Compute $N_i = \{n \in \mathbb{N} \mid Y_i \cap \mathrm{Im}(f_n) \neq \emptyset\}$
10 for $n \in N_i$ do
11     Compute $Y_{i,n} = Y_i \cap \mathrm{Im}(f_n)$
12 end
13 Compute $S_{i,f} = \left\{ s \in \{0,1\}^* \mid |s| \le p(|i|) \wedge C^f(i,s) = 1 \right\}$
14 while True do
15     $B_{i,f} = \left\{ s \in S_{i,f} \mid f\left[Q\left(C^f(i,s)\right)\right] \cap Y_i = \emptyset \right\}$
16     if $B_{i,f} \neq \emptyset$ then
17         return lexicographically smallest $s \in B_{i,f}$
18     end
19     Choose $n \in N_i$ such that $\frac{|Y_{i,n}|}{2^n}$ is maximized.
20     Choose $y \in Y_{i,n}$ such that the probability of querying $i$ is minimal for
         challenge $y$.
21     Set $Y_i = Y_i \setminus \{y\}$
22     Set $Y_{i,n} = Y_{i,n} \setminus \{y\}$
23     if $Y_{i,n} = \emptyset$ then
24         Set $N_i = N_i \setminus \{n\}$
25     end
26 end
```

Conclusion

In Chapter 2, we studied the complexity of the `Arrival` problem and shown that the run-profiles are efficiently recognizable which combined with results of Dohrau et al. [2017] demonstrates that `Arrival` $\in$ UP $\cap$ coUP. We have also proven that `Arrival` belongs to the class CLS by presenting a reduction from `Arrival` to `EOML`. As pointed out by Fearnley et al. [2020], using the same reduction, we get a reduction from `Arrival` to `UniqueEOPL`. There are a few other problems in TFNP which are guaranteed to have a unique solution. Such problems are, for instance, `PLCP` and `USO`, which are also reducible to `UniqueEOPL` as shown by Fearnley et al. [2020]. This raises the following open question:

What is the relation between `Arrival` and other TFNP problems with unique solution? Or more specifically, can we reduce `Arrival` to `PLCP` or `USO`?

The fact that `Arrival` is in the class CLS (which is very close to FP, i.e., the search version of P) also raises the following question:

Is there a polynomial algorithm for `Arrival`?

To this end there are some promising results such as the recent results by Gärtner et al. [2021] who give a sub-exponential time algorithm for `Arrival`. Moreover, they have demonstrated existence of a polynomial-time algorithm at least for graphs with a feedback vertex set of constant-size.

In Chapters 3 and 4, we study the possibility of basing worst-case hardness in TFNP on average case hardness in NP (or in UP), respectively, on injective one-way functions. In the first case, we rule out any fully black-box construction of worst-case hard problem in TFNP from an average-case hard NP problem. Unfortunately, we have not been able to show the same result for one-way functions. More specifically, we can rule out only "simple" reductions from injective one-way functions to TFNP. Thus, the natural question is whether it is possible to extend this result:

Is it possible to rule out fully black-box constructions of worst-case hardness in TFNP from (injective) OWF even when the security reduction is not f-oblivious and/or has adaptive queries?

Bibliography

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

David Aldous. Minimization algorithms and random walk on the d-cube. *The Annals of Probability*, 11(2):403–413, 1983.

Paul Beame, Stephen Cook, Jeff Edmonds, Russell Impagliazzo, and Toniann Pitassi. The relative complexity of NP search problems. *Journal of Computer and System Sciences*, 57(1):3–19, 1998.

Nir Bitansky and Akshay Degwekar. On the complexity of collision resistant hash functions: New and old black-box separations. In *TCC (1)*, volume 11891 of *Lecture Notes in Computer Science*, pages 422–450. Springer, 2019.

Nir Bitansky and Idan Gerichter. On the cryptographic hardness of local search. In *11th Innovations in Theoretical Computer Science Conference, ITCS 2020, January 12-14, 2020, Seattle, Washington, USA*, pages 6:1–6:29, 2020.

Nir Bitansky, Omer Paneth, and Alon Rosen. On the cryptographic hardness of finding a Nash equilibrium. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 1480–1498, 2015.

Nir Bitansky, Akshay Degwekar, and Vinod Vaikuntanathan. Structure versus hardness through the obfuscation lens. *SIAM Journal on Computing*, 50(1): 98–144, 2021.

Manuel Blum and Sampath Kannan. Designing programs that check their work. *Journal of the ACM (JACM)*, 42(1):269–291, 1995.

Zvika Brakerski, Jonathan Katz, Gil Segev, and Arkady Yerukhimovich. Limits on the power of zero-knowledge proofs in cryptographic constructions. In *TCC*, volume 6597 of *Lecture Notes in Computer Science*, pages 559–578. Springer, 2011.

Chris Brzuska and Geoffroy Couteau. Towards fine-grained one-way functions from strong average-case hardness. *IACR Cryptol. ePrint Arch*, 2020:1326, 2020.

Harry Buhrman, Lance Fortnow, Michal Koucký, John D Rogers, and Nikolay Vereshchagin. Does the polynomial hierarchy collapse if onto functions are invertible? *Theory of Computing Systems*, 46(1):143–156, 2010.

Josh Buresh-Oppenheim and Tsuyoshi Morioka. Relativized NP search problems and propositional proof systems. In *19th Annual IEEE Conference on Computational Complexity (CCC 2004), 21-24 June 2004, Amherst, MA, USA*, pages 54–67, 2004.

Joshua Buresh-Oppenheim. On the TFNP complexity of factoring. Unpublished, `http://www.cs.toronto.edu/~bureshop/factor.pdf`, 2006.

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

Arka Rai Choudhuri, Pavel Hubáček, Chethan Kamath, Krzysztof Pietrzak, Alon Rosen, and Guy N. Rothblum. Finding a Nash equilibrium is no easier than breaking Fiat-Shamir. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 1103–1114. ACM, 2019a.

Arka Rai Choudhuri, Pavel Hubáček, Chethan Kamath, Krzysztof Pietrzak, Alon Rosen, and Guy N. Rothblum. PPAD-hardness via iterated squaring modulo a composite. *IACR Cryptology ePrint Archive*, 2019:667, 2019b.

Thomas M. Cover and Joy A. Thomas. *Elements of information theory.* John Wiley & Sons, 2012.

Constantinos Daskalakis and Christos H. Papadimitriou. Continuous local search. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, pages 790–804, 2011.

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

Jérôme Dohrau, Bernd Gärtner, Manuel Kohler, Jiří Matoušek, and Emo Welzl. ARRIVAL: A zero-player graph game in NP$\cap$coNP. In Martin Loebl, Jaroslav Nešetřil, and Robin Thomas, editors, *A Journey Through Discrete Mathematics: A Tribute to Jiří Matoušek*, pages 367–374. Springer International Publishing, 2017.

Naomi Ephraim, Cody Freitag, Ilan Komargodski, and Rafael Pass. Continuous verifiable delay functions. In *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part III*, volume 12107 of *Lecture Notes in Computer Science*, pages 125–154, 2020.

John Fearnley, Martin Gairing, Matthias Mnich, and Rahul Savani. Reachability switching games. *CoRR*, abs/1709.08991, 2017. URL `http://arxiv.org/abs/1709.08991`.

John Fearnley, Spencer Gordon, Ruta Mehta, and Rahul Savani. Unique end of potential line. *Journal of Computer and System Sciences*, 114:1–35, 2020.

John Fearnley, Paul Goldberg, Alexandros Hollender, and Rahul Savani. The complexity of gradient descent: CLS = PPAD $\cap$ PLS. In *Proceedings of the ACM Symposium on Theory of Computing.* Association for Computing Machinery, 2021.

Marc Fischlin. Black-box reductions and separations in cryptography. In *AFRICACRYPT*, volume 7374 of *Lecture Notes in Computer Science*, pages 413–422. Springer, 2012.

Sanjam Garg, Omkant Pandey, and Akshayaram Srinivasan. Revisiting the cryptographic hardness of finding a Nash equilibrium. In *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II*, pages 579–604, 2016.

Bernd Gärtner, Thomas Dueholm Hansen, Pavel Hubáček, Karel Král, Hagar Mosaad, and Veronika Slívová. ARRIVAL: Next Stop in CLS. In *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*, volume 107, pages 60:1–60:13. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2018.

Bernd Gärtner, Sebastian Haslebacher, and Hung P Hoang. A subexponential algorithm for ARRIVAL. *arXiv preprint arXiv:2102.06427*, 2021.

Rosario Gennaro and Luca Trevisan. Lower bounds on the efficiency of generic cryptographic constructions. In *FOCS*, pages 305–313. IEEE Computer Society, 2000.

Yael Gertner, Tal Malkin, and Omer Reingold. On the impossibility of basing trapdoor functions on trapdoor predicates. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, pages 126–135. IEEE Computer Society, 2001.

Oded Goldreich and Leonid A Levin. A hard-core predicate for all one-way functions. In *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, pages 25–32, 1989.

Iftach Haitner, Jonathan J. Hoch, Omer Reingold, and Gil Segev. Finding collisions in interactive protocols - tight lower bounds on the round and communication complexities of statistically hiding commitments. *SIAM J. Comput.*, 44 (1):193–242, 2015.

Chun-Yuan Hsiao and Leonid Reyzin. Finding collisions on a public road, or do secure hash functions need secret coins? In *CRYPTO*, volume 3152 of *Lecture Notes in Computer Science*, pages 92–105. Springer, 2004.

Pavel Hubáček, Moni Naor, and Eylon Yogev. The journey from NP to TFNP hardness. In *8th Innovations in Theoretical Computer Science Conference, ITCS 2017, January 9-11, 2017, Berkeley, CA, USA*, pages 60:1–60:21, 2017.

Pavel Hubáček, Chethan Kamath, Karel Král, and Veronika Slívová. On average-case hardness in TFNP from one-way functions. In *Theory of Cryptography Conference*, pages 614–638. Springer, 2020.

Pavel Hubáček and Eylon Yogev. Hardness of continuous local search: Query complexity and cryptographic lower bounds. *SIAM J. Comput.*, 49(6):1128–1172, 2020. doi: 10.1137/17M1118014. URL `https://doi.org/10.1137/17M1118014`.

Russell Impagliazzo and Steven Rudich. Limits on the provable consequences of one-way permutations. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing, May 14-17, 1989, Seattle, Washigton, USA*, pages 44–61. ACM, 1989.

Ruta Jawale and Dakshita Khurana. Lossy correlation intractability and PPAD hardness from sub-exponential LWE. *IACR Cryptol. ePrint Arch.*, 2020:911, 2020.

Ruta Jawale, Yael Tauman Kalai, Dakshita Khurana, and Rachel Zhang. SNARGs for bounded depth computations and PPAD hardness from sub-exponential LWE. *IACR Cryptol. ePrint Arch.*, 2020:980, 2020.

Emil Jeřábek. Integer factoring and modular square roots. *J. Comput. Syst. Sci.*, 82(2):380–394, 2016.

David S. Johnson, Christos H. Papadimitriou, and Mihalis Yannakakis. How easy is local search? *J. Comput. Syst. Sci.*, 37(1):79–100, 1988.

Yael Tauman Kalai and Rachel Zhang. SNARGs for bounded depth computations from sub-exponential LWE. *IACR Cryptol. ePrint Arch.*, 2020:860, 2020.

Yael Tauman Kalai, Omer Paneth, and Lisa Yang. How to delegate computations publicly. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 1115–1124, 2019.

Yael Tauman Kalai, Omer Paneth, and Lisa Yang. Delegation with updatable unambiguous proofs and PPAD-hardness. In *Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part III*, volume 12172 of *Lecture Notes in Computer Science*, pages 652–673. Springer, 2020.

Karthik C. S. Did the train reach its destination: The complexity of finding a witness. *Inf. Process. Lett.*, 121:17–21, 2017.

Ilan Komargodski and Gil Segev. From Minicrypt to Obfustopia via private-key functional encryption. In *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part I*, pages 122–151, 2017.

Leonid A Levin. Average case complete problems. *SIAM Journal on Computing*, 15(1):285–286, 1986.

Alex Lombardi and Vinod Vaikuntanathan. Fiat-Shamir for repeated squaring with applications to PPAD-hardness and VDFs. In *Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part III*, volume 12172 of *Lecture Notes in Computer Science*, pages 632–651. Springer, 2020.

Mohammad Mahmoody and David Xiao. On the power of randomized reductions and the checkability of sat. In *2010 IEEE 25th Annual Conference on Computational Complexity*, pages 64–75. IEEE, 2010.

Takahiro Matsuda and Kanta Matsuura. On black-box separations among injective one-way functions. In *Theory of Cryptography - 8th Theory of Cryptography Conference, TCC 2011, Providence, RI, USA, March 28-30, 2011. Proceedings*, volume 6597 of *Lecture Notes in Computer Science*, pages 597–614. Springer, 2011.

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

Tsuyoshi Morioka. Classification of search problems and their definability in bounded arithmetic. *Electronic Colloquium on Computational Complexity (ECCC)*, (082), 2001.

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

Rafael Pass and Muthuramakrishnan Venkitasubramaniam. Private coins versus public coins in zero-knowledge proof systems. In *TCC*, volume 5978 of *Lecture Notes in Computer Science*, pages 588–605. Springer, 2010.

Rafael Pass and Muthuramakrishnan Venkitasubramaniam. A round-collapse theorem for computationally-sound protocols; or, TFNP is hard (on average) in Pessiland. *CoRR*, abs/1906.10837, 2019.

Omer Reingold, Luca Trevisan, and Salil P. Vadhan. Notions of reducibility between cryptographic primitives. In *Theory of Cryptography, First Theory of Cryptography Conference, TCC 2004, Cambridge, MA, USA, February 19-21, 2004, Proceedings*, pages 1–20, 2004.

Alon Rosen and Gil Segev. Chosen-ciphertext security via correlated products. *SIAM J. Comput.*, 39(7):3058–3088, 2010.

Alon Rosen, Gil Segev, and Ido Shahaf. Can PPAD hardness be based on standard cryptographic assumptions? In *Theory of Cryptography - 15th International Conference, TCC 2017, Baltimore, MD, USA, November 12-15, 2017, Proceedings, Part II*, volume 10678 of *Lecture Notes in Computer Science*, pages 747–776. Springer, 2017.

Steven Rudich. *Limits on the Provable Consequences of One-way Functions.* PhD thesis, EECS Department, University of California, Berkeley, Dec 1988. URL `http://www2.eecs.berkeley.edu/Pubs/TechRpts/1988/6060.html`.

Daniel R. Simon. Finding collisions on a one-way street: Can secure hash functions be based on general assumptions? In *Advances in Cryptology - EUROCRYPT '98, International Conference on the Theory and Application of Cryptographic Techniques, Espoo, Finland, May 31 - June 4, 1998, Proceeding*, volume 1403 of *Lecture Notes in Computer Science*, pages 334–345. Springer, 1998.

Hoeteck Wee. Lower bounds for non-interactive zero-knowledge. In *Theory of Cryptography, 4th Theory of Cryptography Conference, TCC 2007, Amsterdam, The Netherlands, February 21-24, 2007, Proceedings*, volume 4392 of *Lecture Notes in Computer Science*, pages 103–117. Springer, 2007.

List of Figures

List of Tables

List of Publications

1. Bernd Gärtner, Thomas Dueholm Hansen, Pavel Hubáček, Karel Král, Hagar Mosaad, and Veronika Slívová. ARRIVAL: Next Stop in CLS. In *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*, volume 107, pages 60:1–60:13. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2018

2. Pavel Hubáček, Michal Koucký, Karel Král, and Veronika Slívová. Stronger Lower Bounds for Online ORAM. In *Theory of Cryptography Conference*, pages 264–284. Springer, 2019

3. Pavel Hubáček, Chethan Kamath, Karel Král, and Veronika Slívová. On Average-Case Hardness in TFNP from One-Way Functions. In *Theory of Cryptography Conference*, pages 614–638. Springer, 2020

4. Pavel Dvořák, Michal Koucký, Karel Král, and Veronika Slívová. Data Structures Lower Bounds and Popular Conjectures. *arXiv:2102.09294, 2021*, *Under submission*

5. Tereza Klimošová, Josef Malík, Tomáš Masařík, Jana Novotná, Daniël Paulusma, and Veronika Slívová. Colouring $(P_r + P_s)$-free Graphs. *Algorithmica*, 82(7): 1833–1858, 2020