On Average-Case Hardness in TFNP from One-Way Functions[⋆]

Pavel Hubáček[1], Chethan Kamath[2], Karel Král[1], and Veronika Slívová[1]

[1] Charles University
{hubacek,kralka,slivova}@iuuk.mff.cuni.cz
[2] Northeastern University
ckamath@protonmail.com

Abstract. The complexity class TFNP consists of all NP *search* problems that are *total* in the sense that a solution is guaranteed to exist for all instances. Over the years, this class has proved to illuminate surprising connections among several diverse subfields of mathematics like combinatorics, computational topology, and algorithmic game theory. More recently, we are starting to better understand its interplay with cryptography.

We know that certain cryptographic primitives (e.g. one-way permutations, collision-resistant hash functions, or indistinguishability obfuscation) imply average-case hardness in TFNP and its important subclasses. However, its relationship with the most basic cryptographic primitive – i.e., one-way functions (OWFs) – still remains unresolved. Under an additional complexity theoretic assumption, OWFs imply hardness in TFNP (Hubáček, Naor, and Yogev, ITCS 2017). It is also known that average-case hardness in most structured subclasses of TFNP does not imply any form of cryptographic hardness in a black-box way (Rosen, Segev, and Shahaf, TCC 2017) and, thus, one-way functions might be sufficient. Specifically, no negative result which would rule out basing average-case hardness in TFNP *solely* on OWFs is currently known. In this work, we further explore the interplay between TFNP and OWFs and give the first negative results.

As our main result, we show that there cannot exist constructions of average-case (and, in fact, even worst-case) hardness in TFNP from OWFs with a certain type of *simple* black-box security reductions. The class of reductions we rule out is, however, rich enough to capture many of the currently known cryptographic hardness results for TFNP. Our results are established using the framework of black-box separations (Impagliazzo and Rudich, STOC 1989) and involve a novel application of the reconstruction paradigm (Gennaro and Trevisan, FOCS 2000).

1 Introduction

The complexity class TFNP of *total* search problems [MP91], i.e., with syntactically guaranteed existence of a solution for all instances, holds a perplexing place in the hierarchy of computational complexity classes. Given that the decision variant of these problems is trivial, the standard method for arguing computational hardness in TFNP is via clustering these problems into subclasses characterised by the existential argument guaranteeing their totality [Pap94]. This approach was particularly successful in illuminating the connections between search problems in seemingly distant domains such as combinatorics, computational topology, and algorithmic game theory (see, for example, [DGP09,KPR$^+$13,FG18,FG19,GH19] and the references therein). However, all results of this type ultimately leave open the possibility of the existence of polynomial time algorithms for all of TFNP.

An orthogonal line of work, which can be traced to Papadimitriou [Pap94], shows the existence of hard problems in subclasses of TFNP under cryptographic assumptions. Such conditional lower bounds for structured subclasses of TFNP were recently given under increasingly more plausible cryptographic assumptions [Jeř16,BPR15,GPS16,HY17,KS17,CHK$^+$19a,CHK$^+$19b,EFKP20,BG20]. The end of the line in this

[⋆] This research was supported in part by the Grant Agency of the Czech Republic under the grant agreement no. 19-27871X, by the Charles University projects PRIMUS/17/SCI/9, UNCE/SCI/004, and GAUK 251309, and by the Charles University grant SVV-2017-260452. The second author is supported by the IARPA grant IARPA/2019-19-020700009.

sequence of results would correspond to a "dream theorem" establishing average-case hardness in one of the lower classes in the TFNP hierarchy (e.g. CLS [DP11]) under some weak general cryptographic assumptions (e.g. the existence of one-way functions).

An informative parallel for the limits of this methodology can be drawn by considering average-case hardness of *decision* problems in $\mathsf{NP} \cap \mathsf{coNP}$, i.e., the decision analogue of TFNP. The existence of a hard-on-average decision problem in $\mathsf{NP} \cap \mathsf{coNP}$ follows from the existence of hard-core predicates for any one-way permutation [GL89]. However, the existence of injective one-way functions is insufficient for black-box constructions of hard-on-average distributions for languages in $\mathsf{NP} \cap \mathsf{coNP}$ even assuming indistinguishability obfuscation [BDV17] (in fact, [BDV17] ruled out even black-box constructions of worst-case hardness in $\mathsf{NP} \cap \mathsf{coNP}$ using these cryptographic primitives).

For total *search* problems, the existence of hard-on-average TFNP distributions is straightforward either from one-way permutations or collision-resistant hash functions. Moreover, there exist constructions of hard-on-average TFNP distributions either assuming indistinguishability obfuscation and one-way functions [BPR15,GPS16] or under derandomization-style assumptions and one-way functions [HNY17]. On the other hand, no analogue of the impossibility result for basing average-case hardness in $\mathsf{NP} \cap \mathsf{coNP}$ on (injective) one-way functions [BDV17] is currently known for TFNP. Rosen, Segev, and Shahaf [RSS17] showed that most of the known structured subclasses of TFNP do not imply (in a black-box way) any form of cryptographic hardness; thus, it is plausible that hard-on-average distributions in TFNP can be based *solely* on the existence of one-way functions.

Rosen et al. [RSS17] also provided some insight into the structure of hard-on-average distributions in TFNP. They showed that any hard-on-average distribution of a TFNP problem from any primitive which exists relative to a random injective trapdoor function oracle (e.g. one-way functions, injective trapdoor functions, or collision-resistant hash functions) must result in instances with a nearly exponential number of solutions. Even though the [RSS17] result restricts the structure of hard-on-average distributions in TFNP constructed from these cryptographic primitives, it certainly does not rule out their existence. Indeed, a collision-resistant hash function constitutes a hard-on-average TFNP distribution, albeit with an exponential number of solutions.

Motivated by the significant gap between negative and positive results, we revisit the problem of existence of average-case hardness in TFNP under weak general cryptographic assumptions and address the following question:

Are (injective) one-way functions sufficiently structured to imply hard-on-average total search problems?

Towards answering this question, we provide negative results and show that *simple* fully black-box constructions of hard-on-average TFNP distributions from injective one-way functions do not exist.

1.1 Our Results

We recall the details of the construction of a hard-on-average distribution in TFNP from one-way permutations to highlight the restrictions on the type of reductions considered in our results.

Consider the total search problem PIGEON, in which you are given a length-preserving n-bit function represented by a Boolean circuit C and are asked to find either a preimage of the all-zero string (i.e., $x \in \{0,1\}^n : \mathsf{C}(x) = 0^n$) or a non-trivial collision (i.e., $x \neq x' \in \{0,1\}^n : \mathsf{C}(x) = \mathsf{C}(x')$). PIGEON is complete for a subclass of TFNP known as PPP, and Papadimitriou [Pap94] gave the following construction of a hard PIGEON problem from one-way permutations. 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 circuit C_y computing the function $\mathsf{C}_y(x) = f(x) \oplus y$. It is not hard to see that the instance of PIGEON C_y 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 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 C_y specifying the Pigeon instance depends only on the challenge y and does not depend on the one-way permutation f in the sense that C_y itself can be specified without querying f.
– The reduction is *deterministic*, i.e., it simply passes y to specify the Pigeon instance.

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 $\mathsf{C}(x) = h(x) \,\|\, 1$, where $\|$ represents the operation of string concatenation. Since C 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} nontrivial 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. [RSS17]. 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 3.2 for the formal statement of Theorem 1).

Theorem 1 (Main Theorem - Informal). *There is no efficient fully black-box construction of a worst-case hard* TFNP *problem from injective one-way functions with a randomized f-oblivious non-adaptive reduction.*

Thus, we actually rule out a larger class of fully black-box constructions with reductions to injective one-way functions than the *deterministic* f-oblivious *many-one* reductions from the motivating examples of *average-case hardness* of Pigeon from one-way permutations, respectively collision-resistant hash functions. We rule out even constructions of *worst-case hard* TFNP problems using *randomized* f-oblivious non-adaptive[3] reductions. The formal definitions of fully black-box constructions with f-oblivious non-adaptive reductions are given in Section 3.1 (see Definition 2 and Definition 3).

Even though somewhat restricted, our results are the first step towards the full-fledged black-box separation of TFNP and (injective) one-way functions. We note that such result would necessarily subsume the known separation of collision-resistant hash functions and injective one-way functions [Sim98], for which, despite the recent progress, there are only non-trivial proofs [MM11,HHRS15,BD19].

1.2 Our Techniques

Our results employ the framework of black-box separations [IR89,RTV04,Fis12]. The approach suggested in [IR89] 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 [RTV04,MM11], this approach actually rules out a larger class of constructions (so-called "relativized" constructions), and to rule out just fully black-box constructions it suffices to use the so-called two-oracle technique [HR04]. 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 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 rule out the existence of fully black-box constructions of hard-on-average TFNP problems from injective one-way functions, one then has to argue that access to such a "breaker" oracle Solve for TFNP does not help any reduction R in inverting injective one-way functions.

[3] Reductions which can ask *multiple* queries *in parallel* to the TFNP oracle.

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.[4]

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 of Gennaro and Trevisan [GT00] 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 primitive P when given access to *any* oracle SOLVE (see Definition 2). 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 respect from the previous works which relied on the reconstruction paradigm of Gennaro and Trevisan [GT00], e.g., the works which employed the collision-finding oracle of Simon [Sim98,HHRS15,PV10,RS10,BKSY11]. We note that the possibility of designing a breaker oracle which depends on the fully black-box construction was exploited already by Gertner, Malkin, and Reingold [GMR01], 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 [HR04] who, nevertheless, did not leverage this observation in their proofs.

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 outputs 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 [GT00].

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 re-

[4] 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. Indeed, the hard instances C_y^f of PIGEON from one-way permutations described in Section 1.1 have a unique solution on which C_y^f always queries a preimage of y under any f.

construction paradigm is conventionally applied to random permutations [GT00,HHRS15], 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 [MM11] 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 (e.g., the collision finding oracle of Simon [Sim98,Bae14]) 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 in section 1.1, f-oblivious security reductions appear also in the cryptographic literature – see for example the standard security reduction in the Goldreich-Levin theorem establishing the existence of hard-core predicate for any one-way function (note that this particular security reduction is also non-adaptive). A somewhat orthogonal notion of π-oblivious construction appears in the work of Wee [Wee07]. However, it is the implementation of the constructed primitive which is "oblivious" to the one-way permutation π in his work.

1.3 Related Work

TFNP *and its subclasses.* The systematic study of total search problems was initiated by Megiddo and Papadimitriou [MP91] with the definition of complexity class TFNP. They observed that a "semantic" class such as TFNP is unlikely to have complete problems, unless NP = coNP. As a resolution, Papadimitriou [Pap94] defined "syntactic" subclasses of TFNP with the goal of clustering search problems based on the various *existence theorems* used to argue their totality. Perhaps the best known such class is PPAD [Pap94], which captures the computational complexity of finding Nash equilibria in bimatrix games [DGP09,CDT09]. Other subclasses of TFNP include:

– PPA [Pap94], which captures computational problems related to the *parity argument* like Borsuk-Ulam theorem or fair division [FG19];
– PLS [JPY88], defined to capture the computational complexity of problems amenable to *local search* and its "continuous" counterpart CLS $\subseteq$ PLS [DP11], which captures finding the computational complexity of

finding (approximate) local optima of continuous functions and contains interesting problems from game theory (e.g., solving the simple stochastic games of Condon or Shapley); and
– PPP [Pap94] and PWPP $\subseteq$ PPP [Jeř16], motivated by the *pigeonhole principle* and contain important problems related to finding collisions in functions.

The relative complexity of some of these classes was studied in [BCE+98] as it was shown using (worst-case) oracle separations that many of these classes are distinct.

Cryptographic hardness in TFNP*.* Hardness from standard cryptographic primitives was long known for the "higher" classes in TFNP like PPP and PPA. We have already mentioned that one-way permutations imply average-case hardness in PPP [Pap94] and existence of collision-resistant hashing (e.g. hardness of integer factoring or discrete-logarithm problem in prime-order groups) implies average-case hardness in PPP (as well as in PWPP). In addition, Jeřábek [Jeř16], building on the work of Buresh-Oppenheim [Bur06], showed that PPA 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. [BPR15] who reduced hardness in PPAD to indistinguishability obfuscation (and injective OWFs); Hubáček and Yogev [HY17] extended this result to CLS. In a series of recent works [CHK+19a,CHK+19b,EFKP20], the underlying assumption has been relaxed further to cryptographic assumptions that are more plausible than indistinguishability obfuscation. Using similar ideas, Bitansky and Gerichter [BG20] presented a construction for hard-on-average distributions in the complexity class PLS in the random oracle model.

One-way functions and TFNP*.* Hubáček et al. [HNY17] showed that average-case hardness in NP (which is implied by OWFs) implies average-case hardness in TFNP under complexity theoretical assumptions related to derandomization. Pass and Venkitasubramaniam [PV19] recently complemented the [HNY17] result by showing that when OWFs *do not exist*, average-case hardness in NP implies average-case hardness in TFNP. However, a definitive relationship between OWFs and TFNP has remained elusive. This prompted Rosen et al. [RSS17] to explore impossibility of reducing TFNP hardness to OWFs. They gave a partial answer by showing that there do not exist hard-on-average distributions of TFNP instances over $\{0,1\}^n$ with $2^{n^{o(1)}}$ solutions from any primitive which exists relative to a random injective trapdoor function oracle (e.g. one-way functions). Their main observation was that the argument in [Rud88], 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 [Rud88] argument has been exploited to its limits in [RSS17], 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 $\mathsf{NP} \cap \mathsf{coNP}$, 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 but injective OWFs, even with indistinguishability obfuscation, (and, therefore, public-key encryption) cannot imply hardness in TFNP in a black-box way.

2 Preliminaries

Here we present the used notation. Unless stated otherwise, all logarithms are base two. We use $\overline{X}$ to denote the set $\{0,1\}^* \setminus X$. The rather non-standard notation used is our notation of function composition.

Notation 2 (Functions) *Let $X, Y \subseteq \{0,1\}^*$, $f\colon X \to Y$ be a function and $X' \subseteq X$ be a set. Then*

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 $\{f(x) \mid x \in X\} \subseteq Y$, and*
4. *$f[X']$ denotes the* image of the restriction of f on X', *i.e., the set $Im(f \restriction X')$.*

Notation 3 (Injective functions) *We denote 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 $f = \left\{f_n \mid f_n \in Inj_n^{m(n)}\right\}_{n\in\mathbb{N}}$ of injective functions, where $m\colon \mathbb{N} \to \mathbb{N}$ is an injective function such that for all $n \in \mathbb{N}$: $m(n) > n$ and $m(n) \leq 100n^{\log n}$.

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

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 $f \upharpoonright \{0,1\}^n$ (i.e., restriction of f to the domain $\{0,1\}^n$.), where $f \in Inj$.

Note that the set Inj as defined above is reasonably restricted. 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 function − with stretch c, where c is a fixed constant.

Notation 4 (Lexicographically smaller) *We use $x <_{lex} y$ or $y >_{lex} x$ to denote the fact that a string x is lexicographically strictly smaller than the string y.*

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. We redefine the standard notation as follows:

Notation 5 (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 then 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 use some basic information-theoretic results about prefix-free codes.

Definition 1 (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 is $0 \leq j < \min(|c_1|, |c_2|)$ such that $(c_1)_j \neq (c_2)_j$.*

Proposition 1 (Theorem 5.3.1 in [CT12]). *The expected length L of any prefix-free binary code for a random variable X is greater than or equal to the entropy $H(X)$, i.e., $L \geq H(X)$.*

Corollary 1. *To encode a uniformly random permutation $\pi \in Inj_n^n$ using prefix-free encoding it takes at least $\log(2^n!)$ bits in expectation.*

Proof. 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 1, we get that the expected size of the encoding is at least $\log(2^n!)$. □

3 Separating TFNP and Injective One-Way Functions

3.1 Fully Black-Box Construction of Hard TFNP Problem from iOWF

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

Definition 2 (Fully black-box construction of hard TFNP problem from iOWF). *A* fully black-box construction of a hard TFNP problem from injective one-way function *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:** There exists a function $T_R\colon \mathbb{N} \to \mathbb{N}$ such that for all $f \in Inj, n \in \mathbb{N}$, and $y \in \{0,1\}^*$, the algorithm $R^f(1^n, y)$ halts in time $T_R(|y|)$.*
   There exists a function $T_C\colon \mathbb{N} \to \mathbb{N}$ such that for all $f \in Inj$, and $i, s \in \{0,1\}^*$, the algorithm $C^f(i, s)$ halts in time $T_C(|i| + |s|)$.
2. ***Correctness:** For all $f \in Inj$ and for all $i \in \{0,1\}^*$, there exists $s \in \{0,1\}^*$ such that $|s| \le p(|i|)$ and $C^f(i, s) = 1$, i.e., for any instance of the* TFNP *problem there exists a solution of polynomial length.*
3. ***Fully black-box proof of security:** There exists a polynomial p' such that for all $f \in Inj$ and any oracle-aided algorithm* Solve*, if*

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

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

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

Definition 2 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, breaks the security of any injective one-way function. For example in the case of the hard Pigeon problem from one-way permutations discussed in Section 1.1, C would be an algorithm which on input (C, x), respectively (C, x, x'), outputs 1 if and only if $\mathsf{C}(x) = 0^n$, respectively $\mathsf{C}(x) = \mathsf{C}(x')$. The reduction algorithm $R(1^n, y)$ simply queries the TFNP solver Solve with the instance $i = \mathsf{C}_y$, i.e., a circuit computing the function $\mathsf{C}_y(x) = f(x) \oplus y$, and outputs the solution s returned by Solve for which, by construction, $f(s) = y$.

Let us emphasize the order of quantifiers restricting the security reduction from TFNP to iOWF:

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

As the reduction must invert given access to any Solve, the oracle Solve may even depend on the behaviour of the reduction R. In particular, Solve can simulate the security reduction R on an arbitrary input and we leverage this crucial observation towards our negative result.

We could easily relax the definition of fully black-box construction by restricting the quantifier for all f to the form: for all f of some type, e.g., restricting ourselves only to injective functions where $f_n\colon \{0,1\}^n \to \{0,1\}^{n+1}$ for all input lengths n. All our results hold for the relaxed version as well.

Note that we consider security reductions which invert 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 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 consider some restricted classes of security reductions as defined below.

Definition 3 (Deterministic/randomized, many-one/non-adaptive, f-oblivious reductions). *Let (R, T_R, C, T_C, p) be a fully black-box construction of a hard* TFNP *problem from injective one-way functions.*

We distinguish deterministic / randomized reductions. 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 security reduction R is many-one *if for all $f \in iOWF$, for any oracle* Solve *and for all $y \in \{0,1\}^*$, $R^{f,\textsc{Solve}}(1^n, y)$ makes at most one query to the oracle* Solve.

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

*The security reduction R is f-*oblivious *if for all $y \in \{0,1\}^*$, for any oracle* Solve *and any pair of functions $f, f' \in iOWF$, $Q_{\textsc{Solve}}(R^{f,\textsc{Solve}}(1^n, y)) = Q_{\textsc{Solve}}(R^{f',\textsc{Solve}}(1^n, y))$ (i.e., queries to the oracle* Solve *depend only on the input y and are independent of the oracle 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 Solve. We introduce the following notation in order to distinguish where queries originate which allows us to reason about the view the security reduction has over the function f in our proof of Theorem 1.

Notation 6 (Query sets Q) *We distinguish the following sets of queries to oracles depending on where these queries originated and which oracle is queried.*

- *Let $Q(C^f(i, s))$ 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 $Q_{\textsc{Solve}}(R^{f,\textsc{Solve}}(1^n, y))$ 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 $Q_f^{dir}(R^{f,\textsc{Solve}}(1^n, y))$ 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 $Q_f^{indir}(R^{f,\textsc{Solve}}(1^n, y))$ 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 $i \in Q_{\textsc{Solve}}(R^{f,\textsc{Solve}}(1^n, y))$ and $s = \textsc{Solve}^f(i)$.*
- *Let $Q_f(R^{f,\textsc{Solve}}(1^n, y)) = Q_f^{dir}(R^{f,\textsc{Solve}}(1^n, y)) \cup Q_f^{indir}(R^{f,\textsc{Solve}}(1^n, y))$.*

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.

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 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 1. *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 with deterministic f-oblivious many-one reduction with success probability at least $2^{-0.1n}$ such that both running times $T_R, T_C \in O(n^{polylog(n)})$.*

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 1 uses, on a high level, a similar template as other black-box separations in the literature. 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 1 and explained below.

Algorithm 1: The oracle Solve.

```
Hardwired     : a fully black-box construction $(R, T_R, C, T_C, p)$ of a hard TFNP problem from iOWF
Oracle access: an injective function $f = \{f_n\}_{n\in\mathbb{N}} \in \mathrm{Inj}$
Input         : an instance $i \in \{0,1\}^*$
Output        : a solution $s \in \{0,1\}^*$ such that $C^f(i,s) = 1$
1  Compute $Y_i = \bigcup_{n=1}^{T_C(|i|+p(|i|))} \{y \in \mathrm{Im}(f_n) \mid i \in Q_{\textsc{Solve}}(R^{f,\textsc{Solve}}(1^n, y))\}$
2  Compute $N_i = \{n \in \mathbb{N} \mid Y_i \cap \mathrm{Im}(f_n) \neq \emptyset\}$
3  for $n \in N_i$ do
4      Compute $Y_{i,n} = Y_i \cap \mathrm{Im}(f_n)$
5  end
6  Compute $S_{i,f} = \{s \in \{0,1\}^* \mid |s| \leq p(|i|)$ and $C^f(i,s) = 1\}$
7  while True do
8      $B_{i,f} = \{s \in S_{i,f} \mid f[Q(C^f(i,s))] \cap Y_i = \emptyset\}$
9      if $B_{i,f} \neq \emptyset$ then
10         return lexicographically smallest $s \in B_{i,f}$
11     end
12     Choose $n \in N_i$ such that $\frac{|Y_{i,n}|}{2^n}$ is maximized.
13     Set $N_i = N_i \setminus \{n\}$
14     Set $Y_i = Y_i \setminus Y_{i,n}$
15 end
```

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 which is hardwired in the oracle 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 Y_i of all challenges y on which the reduction would query the instance i.[5] The challenges in Y_i become "protected" and Solve will attempt to provide a solution which does not reveal a preimage of any $y \in Y_i$ (i.e., s such that $C^f(i,s)$ does not make an f-query on a preimage of any $y \in Y_i$).

Note that we could run into a potential technical issue when defining Y_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 $\mathrm{Im}(f_n)$ for $n \leq T_C(|i| + p(|i|))$.

On lines 2–6, Solve computes the following auxiliary sets N_i, $Y_{i,n}$, and $S_{i,f}$. The set N_i contains all input lengths for the preimages x such that $R^{f,\textsc{Solve}}(1^n, f(x))$ 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 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 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

[5] Here we crucially rely on f-obliviousness of the reduction algorithm R which ensures that Y_i depends only on the image of f.

Algorithm 2: The algorithm $\textsc{Encode}_n$.

```
Hardwired     : a fully black-box construction $(R, T_R, C, T_C, p)$ of a hard $\mathsf{TFNP}$ problem from iOWF
Common Input: an injective function $h \in \mathrm{Inj}$ shared with $\textsc{Decode}_n$
Input         : a permutation $\pi \in \mathrm{Inj}_n^n$ on $\{0,1\}^n$
Output        : an encoding $\mathcal{M}$ of $\pi$
$f = h \circ \pi$, i.e., $f(x) = \begin{cases} h(\pi(x)) & \text{for all } x \text{ of length } n \\ h(x) & \text{otherwise} \end{cases}$
$\mathrm{INV}_f = \{y \in \mathrm{Im}(h_n) \mid R^{\textsc{Solve},f}(1^n, y) = f^{-1}(y)\}$
$G_f = \{y \in \mathrm{INV}_f \mid f^{-1}(y) \notin Q_f^{\mathrm{indir}}(R^{f,\textsc{Solve}}(1^n, y))\}$
$Y_f = \emptyset$ and $X_f = \emptyset$
while $G_f \neq \emptyset$ do
    Pick lexicographically smallest $y \in G_f$
    $G_f = G_f \setminus \left(f[Q_f(R^{f,\textsc{Solve}}(1^n, y))] \cup \{y\}\right)$
    $Y_f = Y_f \cup \{y\}$ and $X_f = X_f \cup \{f^{-1}(y)\}$
end
if $|X_f| < 2^{0.6n}$ then
    return $\mathcal{M} = (0, \pi)$
end
else
    return $\mathcal{M} = (1, |X_f|, Y_f, X_f, \sigma = f \restriction (\{0,1\}^n \setminus X_f)) \in \{0,1\}^{1+n+\left\lceil \log \binom{2^n}{|Y_f|} \right\rceil + \left\lceil \log \binom{2^n}{|X_f|} \right\rceil + \lceil \log(|\{0,1\}^n \setminus X_f|!) \rceil}$
end
```

often enough, as any fixed solution induces only a bounded number of queries to f. Therefore, if the set of challenges on which R queries i is dense enough w.r.t. some image length then, with overwhelming probability, an arbitrary solution will be benign for the random challenge y given to the reduction. Thus, we allow $\textsc{Solve}$ 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 $\textsc{Solve}$ is able to find a benign solution which protects the preimages of length n (see Claim 14). 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 only a small number of preimages of challenges from the set $Y_{i,n}$ (see Claim 13).

In order to show formally that $\textsc{Solve}$ does not help in inverting the injective one-way function, we employ an incompressibility argument similar to [GT00]. Specifically, we present algorithms $\textsc{Encode}_n$ (given in Algorithm 2) and $\textsc{Decode}_n$ (given in Algorithm 3) 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 $\textsc{Encode}_n$, we have access to the whole permutation and we can effectively provide the reduction with access to $\textsc{Solve}$. However, to be able to use the reduction also in the $\textsc{Decode}_n$, we have to be able to simulate access to our $\textsc{Solve}$ oracle given access only to a partially defined oracle f (as we are reconstructing f). For the description of the algorithm $\textsc{SolveSim}$, which simulates the $\textsc{Solve}$ oracle for the purpose of decoding in $\textsc{Decode}_n$, see Algorithm 4.

$\textsc{Encode}_n$ *algorithm:* The algorithm $\textsc{Encode}_n$ (Algorithm 2) 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 $\textsc{Encode}_n$ (as well as $\textsc{Decode}_n$) with an injective function h. $\textsc{Encode}_n$ then computes the function f as a composition of the functions $h \circ \pi$ and uses the

Algorithm 3: The algorithm $\textsc{Decode}_n$.

```
Hardwired      : a fully black-box construction $(R, T_R, C, T_C, p)$ of a hard TFNP problem from iOWF
Common Input: an injective function $h \in \mathrm{Inj}$ shared with $\textsc{Encode}_n$
Input          : an encoding $\mathcal{M}$
Output         : a permutation $\pi \in \mathrm{Inj}_n^n$
Hardwired      : $f$-oblivious many-one fully black-box construction $(R, T_R, C, T_C, p)$
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 $\mathcal{M}' = (|X_f|, Y_f, X_f, \sigma)$
7  Set partial function $f' = \begin{cases} \sigma & \text{for inputs of length } n \\ h & \text{otherwise} \end{cases}$   // $f'$ is defined only outside $X_f$
8  while $Y_f \neq \emptyset$ do
9      Pick lexicographically smallest $y \in Y_f$
10     Let $f''(x) = \begin{cases} y & \text{for all } x \in \mathrm{Dom}(h) \setminus \mathrm{Dom}(f') \\ f'(x) & \text{otherwise} \end{cases}$
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 $\pi = (h^{-1} \circ f') \restriction \{0,1\}^n$
```

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, $\textsc{Encode}_n$ 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 $\textsc{Encode}_n$ computes the set G_f, which is the set of "good" challenges y, on which the reduction successfully inverts even though $\textsc{Solve}$ 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 as the part of f for the challenges in G_f will be algorithmically reconstructed by $\textsc{Decode}_n$ using the security reduction R.

Specifically, $\textsc{Encode}_n$ computes Y_f, the subset of G_f for which the preimages will be algorithmically reconstructed, as follows: $\textsc{Encode}_n$ 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 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 (Lemma 4). 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 function 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 the two cases.

$\textsc{Decode}_n$ *algorithm:* The encoding returned by $\textsc{Encode}_n$ is uniquely decodable by $\textsc{Decode}_n$ given in Algorithm 3 (see Section 4.2). When the output of $\textsc{Encode}_n$ starts with 0, the rest of the encoding is an encoding of π and we are immediately done with its reconstruction. If the output starts with bit 1, 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 2^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 $\operatorname{Im}(f) \setminus Y_f$. 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 $\operatorname{Im}(\sigma) = \operatorname{Im}(f) \setminus Y_f = \operatorname{Im}(h \circ \pi) \setminus Y_f = \operatorname{Im}(h) \setminus Y_f$.

$\textsc{Decode}_n$ then computes the remaining preimages one by one in lexicographic order 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 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'.

$\textsc{SolveSim}$ *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 4) is similar to the computation of $\textsc{Solve}$ (Algorithm 1). First $\textsc{SolveSim}$ computes the sets Y_i. There is one big difference between $\textsc{Solve}$ and $\textsc{SolveSim}$. As $\textsc{SolveSim}$ does not have access to the whole 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 $\forall n \in \mathbb{N}\colon \operatorname{Im}(h_n) = \operatorname{Im}(f_n)$ using $\operatorname{Im}(h_n)$ instead of $\operatorname{Im}(f_n)$ makes no difference to the computation.
– The second place where h is used instead of f is when $\textsc{SolveSim}$ computes the set Y_i, especially when determining images y for which the security reduction R queries the given instance i. $\textsc{SolveSim}$ computes the same Y_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. Especially 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 $S_{i,f}$. $S_{i,f'}$ contains only the solutions $\textsc{SolveSim}$ is "aware of" (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 will 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 1. Lemma 1 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 such that $C^f(i,s) = 1$ (formal proof is given in Section 4.1). Thus any distribution of instances of the TFNP problem is easy in the presence of $\textsc{Solve}$.

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

To argue that $\textsc{Solve}$ does not help in inverting injective functions, we analyze the joint properties of the algorithms $\textsc{Encode}_n$ and $\textsc{Decode}_n$. First, we show that $\textsc{Decode}_n$ always returns the correct permutation encoded by $\textsc{Encode}_n$ (see Section 4.2 for the formal proof).

```
Algorithm 4: The algorithm SolveSim.
Input      : A function $h \in \mathrm{Inj}$, partial injective function $f' \in \mathrm{Inj}$, and an instance $i \in \{0,1\}^*$
Output     : Solution, i.e., $s \in \{0,1\}^*$ such that $C^{f'}(i,s) = 1$
Hardwired  : $f$-oblivious many-one fully black-box construction $(R, T_R, C, T_C, p)$
1  Compute $Y_i = \bigcup_{n=1}^{T_C(|i|+p(|i|))} \{y \in \mathrm{Im}(h_n) \mid i \in Q_{\textsc{Solve}}(R^{h,\textsc{Solve}}(1^n, y))\}$
2  Compute $N_i = \{n \in \mathbb{N} \mid Y_i \cap \mathrm{Im}(h_n) \neq \emptyset\}$
3  for $n \in N_i$ do
4      Compute $Y_{i,n} = Y_i \cap \mathrm{Im}(h_n)$
5  end
6  Compute $S_{i,f'} = \left\{s \in \{0,1\}^* \;\middle|\; |s| \leq p(|i|) \text{ and } Q(C^{f'}(i,s)) \subseteq \mathrm{Dom}(f') \text{ and } C^{f'}(i,s) = 1\right\}$
7  while True do
8      $B_{i,f'} = \{s \in S_{i,f'} \mid f[Q(C^{f'}(i,s))] \cap Y_i = \emptyset\}$ // "benign" solutions
9      if $B_{i,f'} \neq \emptyset$ then
10         return lexicographically smallest $s \in B_{i,f'}$
11     end
12     Choose $n \in N_i$ such that $\frac{|Y_{i,n}|}{2^n}$ is maximized.
13     Set $N_i = N_i \setminus \{n\}$
14     Set $Y_i = Y_i \setminus Y_{i,n}$
15 end
```

Lemma 2. *For all $n \in \mathbb{N}$, $\pi \in \mathit{Inj}_n^n$, and $h \in \mathit{Inj}$,*

$$\textsc{Decode}_n^h(\textsc{Encode}_n^h(\pi)) = \pi,$$

where $\textsc{Encode}_n$, *respectively* $\textsc{Decode}_n$, *is given in Algorithm 2, respectively Algorithm 3.*

Second, we show that the encoding output by $\textsc{Encode}_n$ is prefix-free (see Section 4.3 for the formal proof).

Lemma 3. *Let $h \in \mathit{Inj}$ be any injective function and $n \in \mathbb{N}$, then the encoding given by the algorithm* $\textsc{Encode}_n$ *(Algorithm 2) is prefix-free, i.e.,*

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

Finally, we bound the expected size of the encoding given by $\textsc{Encode}_n$ (see Section 4.4) which contradicts the information-theoretic bound implied by Corollary 1.

Lemma 4. *Let (R, T_R, C, T_C, p) be a fully black-box construction of a hard* TFNP *problem from an injective one-way function. Assume $n \in \mathbb{N}$ is large enough so that $n \geq 50$ and $2q(n) + 1 \leq 2^{0.2n}$, where $q(n)$ is the maximal number of f-queries made by C on the queried instance (see Definition 4). Let the success probability of R be $\beta \geq 2^{-0.1n}$, i.e., for any f we have*

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

Then

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

In Claim 11, we show that the upper bound $2q(n) + 1 \leq 2^{0.2n}$ used in the statement of the lemma is without loss of generality for large enough n and for the efficient algorithms R, C. We use this fact in the proof of the main theorem (Theorem 1), but for the ease of presentation we state the claim only in Section 4.4.

Equipped with the above lemmata, we can prove Theorem 1.

Proof (of Theorem 1). For contradiction suppose there is such a reduction (R, T_R, C, T_C, p). Then by Lemma 1 the algorithm Solve (Line 1) returns a valid solution with probability one. Thus the reduction R given access to any oracle $f \in \mathrm{Inj}$ and the oracle Solve (Line 1) must invert f with high probability, i.e.,

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

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

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

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

where $q(n)$ is the maximal number of f-queries made by C on the queried instance (see Definition 4). Note that in Claim 11 we show that the bounds on running times $T_C, T_R \in O(n^{\mathrm{polylog}(n)})$ imply that $q(n) \in o(2^{0.2n})$. Thus for large enough n the upper bound $2q(n) + 1 \leq 2^{0.2n}$ holds without loss of generality.

For any $h \in \mathrm{Inj}$ we can use the algorithm $\textsc{Encode}_n^h$ (Algorithm 2) to encode a given permutation $\pi \in \mathrm{Inj}_n^n$. Decodability of the encoding follows from Lemma 2. Moreover by Lemma 3 the encoding is a prefix-free code. By Lemma 4 there is a function $h \in \mathrm{Inj}$ such that the pair $\textsc{Encode}_n^h, \textsc{Decode}_n^h$ defines an encoding of $\pi \leftarrow \mathrm{Inj}_n^n$ with expected length at most $\log(2^n!) - \frac{8}{10} n 2^{0.1n}$. This contradicts the information-theoretic bound (Corollary 1). ◻

4 Proofs of the Supporting Lemmata

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 4.2 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 4.3.

Section 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.

4.1 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 1. *For any instance $i \in \{0,1\}^*$ and any $f \in$ Inj 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 $Y_i, N_i, Y_{i,n}$ 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 1, 2 and 14, 13). 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 2 - Correctness) the algorithm Solve always halts.

Finally when the algorithm returns (only line 10 of Algorithm 1), 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 6 of Algorithm 1). ◻

4.2 Encoding Is Uniquely Decodable

In this section, we show that the encoding given by $\textsc{Encode}_n$ (Algorithm 2) is uniquely decodable by $\textsc{Decode}_n$ (Algorithm 3), i.e., we prove the following lemma.

Lemma 2. *For any $n \in \mathbb{N}$, any $\pi \in Inj_n^n$ and for any $h \in Inj$*

$$\textsc{Decode}_n^h(\textsc{Encode}_n^h(\pi)) = \pi,$$

where $\textsc{Encode}_n$, $\textsc{Decode}_n$ are as described in Algorithms 2 and 3.

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 $\textsc{Decode}_n$ 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 7).

Claim 7. *Let $n \in \mathbb{N}$ be any number, $h \in Inj$ be any function, $\pi \in Inj_n^n$ be any permutation and let $f = h \circ \pi$. Let $\mathcal{M}$ be the output of $\textsc{Encode}_n^h(\pi)$ (see Algorithm 2) 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 3). Let

$$f' = f \restriction (\{x \in X_f \mid f(x) \text{ is the } k\text{-th lexicographically smallest in } Y_f \text{ for } k < j\} \cup \overline{X_f})$$

and

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

Then

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

Note that if $\textsc{Decode}_n$ computed the first $j-1$ preimages then f', f'' from Algorithm 3 correspond to their definition given in Claim 7. This is formalized in the proof of Claim 9.

Proof (of Claim 7). Let x' be any query from $Q_f^{\mathrm{dir}}(R^{f,\textsc{Solve}}(1^n, y))$. 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.

$x' \in X_f \wedge f(x') <_{\mathbf{lex}} y$: Then $f'(x')$ is defined in the j-th iteration, because $\textsc{Decode}_n$ (Algorithm 3) computes preimage of $f(x')$ before y. Thus $f''(x') = f'(x') = f(x')$, where the last inequality 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'', $f''(x') = y = f(x')$.

$x' \in X_f \wedge f(x') >_{\mathbf{lex}} y$: As $f(x') >_{\mathrm{lex}} y$, the $\textsc{Encode}_n$ (Algorithm 2) processes y before $f(x')$. Then if x' is in $Q_f^{\mathrm{dir}}(R^{f,\textsc{Solve}(f,\cdot)}(1^n, y))$, $\textsc{Encode}_n$ removes $f(x')$ from G_f (line 7, Algorithm 2). Thus $f(x')$ is never added into Y_f which is a contradiction.

□

Claim 7 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 $\textsc{Solve}$ (Algorithm 1) and $\textsc{SolveSim}$ (Algorithm 4) answer the TFNP query with the same solution whenever the algorithm $\textsc{Decode}_n$ queries $\textsc{SolveSim}$ during the simulation of the security reduction R.

The main observation is that the algorithm $\textsc{SolveSim}$ is aware of the solution returned by $\textsc{Solve}$ (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 $\textsc{Solve}$ gives the same result in algorithm $\textsc{SolveSim}$, which is possibly aware only about few solutions of the instance. Here we utilize the fact that the algorithm $\textsc{Solve}$ always returns lexicographically smallest solution.

Claim 8. *Let $n \in \mathbb{N}$ be any number, $h \in Inj$ be any function, $\pi \in Inj_n^n$ be any permutation and let $f = h \circ \pi$. Let $\mathcal{M}$ be the output of $\textsc{Encode}_n^h(\pi)$ (see Algorithm 2) and assume that $\mathcal{M} = (1, |X_f|, Y_f, X_f, \sigma)$. Let $j \in \mathbb{N}$ be such that $1 \le j \le |X_f|$. 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*

$$f' = f \restriction (\{x \in X_f \mid f(x) \text{ is the } k\text{-th lexicographically smallest in } Y_f \text{ for } k < j\} \cup \overline{X_f}).$$

Then the query to $\textsc{Solve}$ is answered correctly, i.e.,

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

Proof. We fix $i \in Q_{\textsc{Solve}}(R^{f,\textsc{Solve}}(1^n, y))$. 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 $\textsc{Solve}$ and $\textsc{SolveSim}$. 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 algorithm $\textsc{Solve}$ (Line 1, line 6).

$S_{\textsc{SolveSim},f'}$ Let $S_{\textsc{SolveSim},f'}$ denote the set $S_{f',i}$ as computed by algorithm $\textsc{SolveSim}$ (Algorithm 4, line 6).

$B_{\textsc{Solve},f,k}$ 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 $\textsc{Solve}$ (Algorithm 1, line 8).

$B_{\textsc{SolveSim},f',k}$ 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 $\textsc{SolveSim}$ (Algorithm 4, line 8).

Moreover by f-obliviousness the sets $N_i, Y_i, Y_{i,n}$ depend only on the image of f which is identical to the image of h and h is known to $\textsc{SolveSim}$. Thus all sets used in algorithms $\textsc{Solve}$ and $\textsc{SolveSim}$ 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 thus can be computed by both $\textsc{Solve}$ and $\textsc{SolveSim}$.

First recall that $y \in Y_f$ thus by the definition of Y_f, respectively the definition of $G_f \supseteq Y_f$ (Algorithm 2, lines 3, 4, 6, and 8) we get that $f^{-1}(y) \notin Q(C^f(i,s))$ as images of all $x \in Q_f(R^{f,\textsc{Solve}}(1^n, y)) \supseteq Q(C^f(i,s))$ are removed from G_f (Algorithm 2, line 7). This means that for any $x \in Q(C^f(i,s))$ 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

$$s_{\textsc{Solve}} \in S_{\textsc{SolveSim},f'}. \tag{1}$$

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 equation holds:

$$S_{\textsc{SolveSim},f'} \subseteq S_{\textsc{Solve},f}. \tag{2}$$

During the entire computation both $\textsc{Solve}$ and $\textsc{SolveSim}$ 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 the equation 2 we have that

$$\forall k \in \mathbb{N} \quad B_{\textsc{SolveSim},f',k} = B_{\textsc{Solve},f,k} \cap S_{\textsc{SolveSim},f'}. \tag{3}$$

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:

$$\{s_{\textsc{Solve}}, s_{\textsc{SolveSim}}\} \subseteq B_{\textsc{Solve},f,k} \cap B_{\textsc{SolveSim},f',k} \tag{4}$$

We distinguish the following cases:

$B_{\textsc{Solve},f,k} = \emptyset$ By the assumption $B_{\textsc{SolveSim},f',k} \neq \emptyset$. Thus $\textsc{SolveSim}$ 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 the equation 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 $\textsc{Solve}$ algorithm returns in the k-th iteration of the while loop and thus $s_{\textsc{Solve}} \in B_{\textsc{Solve},f,k}$. By the equation 1 $s_{\textsc{Solve}}$ is contained in $S_{\textsc{SolveSim},f'}$ too. Thus by Equation 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 $\textsc{Solve}$ and $\textsc{SolveSim}$ 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 both algorithms $\textsc{Solve}$ and $\textsc{SolveSim}$ return the same string $s_{\textsc{Solve}} = s_{\textsc{SolveSim}}$. □

We combine the Claims 7 and 8 to show that the function f' is defined consistently with f during the whole computation.

Claim 9. *Let $n \in \mathbb{N}$ be any number, $h \in Inj$ be any function, $\pi \in Inj_n^n$ be any permutation and let $f = h \circ \pi$. Let $\mathcal{M}$ be the output of $\textsc{Encode}_n^h(\pi)$ (see Algorithm 2) 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 $\textsc{Decode}_n^h(\mathcal{M})$ after j iterations of the while loop (line 8, Algorithm 3). Then*

$$f' \subseteq f$$

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

Proof. We proceed by induction on the number of iterations of the while loop of the algorithm $\textsc{Decode}_n$ (Algorithm 3). Before the first iteration, we set $f' = \sigma$ for inputs of length n (see line 11 of Algorithm 2). Recall that $\sigma = f \restriction (\{0,1\}^n \setminus X_f)$ on inputs of length n. For inputs of length different from n we set f' according to h, which is the same as f as $f(x) = h(x)$ for all inputs x of length different from n. Thus, we immediately get $f' \subseteq f$.

In each iteration of the while loop, the algorithm $\textsc{Decode}_n$ (Algorithm 3) 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 3) 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 $R^{f,\textsc{Solve}(f,.)}(1^n, y) = f^{-1}(y)$, as $y \in Y_f$ are taken from the set INV_f (see line 2, Algorithm 2). 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 7 and 8 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 $y \in Y_f$. By induction hypothesis it is defined correctly (gives the same output as f on all $x \in \mathrm{Dom}(f')$). By Claim 7 all f-queries are answered correctly, i.e., for any

$$x' \in Q_f^{\mathrm{dir}}(R^{f,\textsc{Solve}(f,.)}(1^n, y)) \text{ we have } f''(x') = f(x').$$

To prove that $\textsc{Solve}$ query is answered correctly observe that by f-obliviousness of the reduction the same TFNP instances are queried, i.e.,

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

By Claim 8 the query to SolveSim is answered identically as it would be answered by Solve. Thus $R^{f'',\textsc{SolveSim}(h,f',.)}(1^n, y) = R^{f,\textsc{Solve}(f,.)}(1^n, y)$ and the preimage of y is set correctly. ◻

Finally we use Claim 9 to prove that $\textsc{Decode}_n$ correctly decodes the message of $\textsc{Encode}_n$ and outputs the right permutation.

Proof (of Lemma 2). We assume that the first bit of the output of $\textsc{Encode}_n$ (Algorithm 2) is 1, otherwise a full description of π has been sent to $\textsc{Decode}_n$ (Algorithm 3) and the lemma holds trivially. Let $f = h \circ \pi$ similarly as in the $\textsc{Encode}_n$ algorithm (Algorithm 2). We have to show that $\textsc{Decode}_n$ (Algorithm 3) on line 15 holds function f' such that $f' = f$. We use Claim 9 to get that $f' \subseteq f$. Now observe that on line 15 of Algorithm 3 the function f' is defined for all inputs. Thus $f' = f$. Finally the $\textsc{Decode}_n$ algorithm returns a function $(h^{-1} \circ f) \restriction \{0,1\}^n = (h^{-1} \circ h \circ \pi) \restriction \{0,1\}^n = \pi$. ◻

4.3 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.

Lemma 3. *Let $h \in Inj$ be any injective function and $n \in \mathbb{N}$, then the encoding given by the algorithm* $\textsc{Encode}_n$ *(Algorithm 2) is prefix-free, i.e.,*

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

We use the Claim 10 which computes the size of an encoding of a permutation π as returned by $\textsc{Encode}_n$ (Algorithm 2).

Claim 10. *The size of the encoding computed by* $\textsc{Encode}_n$ *(Algorithm 2) is*

1. $1 + \lceil \log(2^n!) \rceil$ *bits in case* $\textsc{Encode}_n$ *(Algorithm 2) 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* $\textsc{Encode}_n$ *(Algorithm 2) 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 $\left\lceil \binom{n}{k} \right\rceil$ 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 $\lceil n! \rceil$ 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 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 $|X_f|$ and 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 domain and the image of σ, as those can be computed (domain from the set X_f and the image from the sets Y_f and $\mathrm{Im}(h_n) = \mathrm{Im}(f_n)$. 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 10 to show that the encoding is prefix-free.

Proof (of Lemma 3). Let $\pi, \pi' \in \mathrm{Inj}_n^n$ be any permutations such that $\pi \neq \pi'$. Let $E_h(\pi) = |\textsc{Encode}_n^h(\pi)|$ and $E_h(\pi') = |\textsc{Encode}_n^h(\pi')|$. Note that by Lemma 2, $\textsc{Encode}_n^h(\pi) \neq \textsc{Encode}_n^h(\pi')$, otherwise we could not uniquely decode the permutation. First observe that if both $\textsc{Encode}_n^h(\pi), \textsc{Encode}_n^h(\pi')$ start with bit 0, then

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

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

We have described the encoding formally in the proof of Claim 10. It suffices to prove the lemma for the case when both encodings $\textsc{Encode}_n^h(\pi)$ and $\textsc{Encode}_n^h(\pi')$ start with bit 1. 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 $\textsc{Encode}_n^h(\pi)$ and $\textsc{Encode}_n^h(\pi')$ differ on the first $n+1$ bits and $\textsc{Encode}_n^h(\pi)$ is not a prefix of $\textsc{Encode}_n^h(\pi')$. 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 $\textsc{Encode}_n^h(\pi) \neq \textsc{Encode}_n^h(\pi')$, we get $\textsc{Encode}_n^h(\pi)$ is not a prefix of $\textsc{Encode}_n^h(\pi')$, which concludes the proof of the lemma. □

4.4 Bounding the Size of the Encoding

The most important part of the whole proof is of course bounding the expected size of our encoding. Claim 12 bounds the probability that a random y is in the good set (Algorithm 2, 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.

Definition 4. *Let (R, T_R, C, T_C, p) be any f-oblivious many-one fully black-box reduction from iOWFto* TFNP. *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.,*

$$q(n) = \sup_{\substack{f \in Inj,\ x \in \{0,1\}^n \\ y = f(x)}} \left\{ |Q(C^f(i,s))| \;\middle|\; i \in Q_{\textsc{Solve}}(R^{f,\textsc{Solve}}(1^n, y)), s \in \{0,1\}^*, |s| \le p(|i|) \right\}.$$

Note that the queries considered above are indirect (i.e., they originated in C and we do not consider f-queries made directly from R).

Lemma 4. *Let (R, T_R, C, T_C, p) be any deterministic f-oblivious fully black-box many-one reduction from iOWFto* TFNP. *Assume $n \in \mathbb{N}$ is large enough so that $n \ge 50$ and $2q(n)+1 \le 2^{0.2n}$ where $q(n)$ is the maximal number of f-queries made by C on the queried instance (see Definition 4). Let the success probability of R be $\beta \ge 2^{-0.1n}$, i.e., for any $f \in Inj$ we have*

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

Then

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

In Claim 11, 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 is without loss of generality.

Claim 11. *Let (R, T_R, C, T_C, p) be any deterministic f-oblivious many-one fully black-box reduction from iOWFto* TFNP. *Let T_C be an upper bound on the running time of the algorithm C and T_R be an upper bound on the running time of the algorithm R. We can bound $q(n)$, i.e., the maximal number of f-queries made by C on any queried instance (see Definition 4), by*

$$q(n) \leq T_C(2p(T_R(101n^{\log n}))). \tag{5}$$

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

$$q(n) \in o(2^{0.2n}). \tag{6}$$

Proof. 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 $n + 100n^{\log n} \leq 101n^{\log n}$ (see Notation 3) and thus can query the TFNP instance i of length at most $T_R(101n^{\log n})$. 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 $p(T_R(101n^{\log n})) + T_R(101n^{\log n}) \leq 2p(T_R(101n^{\log n}))$. Finally, we can bound the number of queries of C by its running time by $T_C(2p(T_R(101n^{\log n})))$. This concludes the proof of inequality 5.

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

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

which proves the inequality 6. □

We prove the bound on the expected length of the encoding (stated in Lemma 4) with expectation taken not only over the choice of π but also over the choice of $h \leftarrow \text{Inj}$ (the expectation could be also understood to be taken over the choice of $f \leftarrow \text{Inj}$ where $f = h \circ \pi$). Then we use an averaging argument to show that there exists $h \in \text{Inj}$ for which the encoding is short enough.

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. the reduction does not invert and
2. the Solve solution queries the preimage.

We bound the probability of both these events and then use them to bound the overall probability of y not being "good".

For bounding the probability of 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 from the image of $\{0,1\}^n$ for which the reduction queries the instance i. In this case the bound in Claim 13 will be used to show that the solution does not "help" the reduction too much.
2. Or there exist only few challenges y of the same length n, such that i is queried. In this case we want to bound the probability that $Y_{i,n}$ (i.e., the "protected" images of f_n as in Algorithm 1) 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. We bound this probability in Claim 14.

Claim 12. *Let (R, T_R, C, T_C, p) be any deterministic f-oblivious many-one fully black-box reduction from iOWFto* TFNP. *Let $n \in \mathbb{N}$ be any natural number and let $\pi \in Inj_n^n$ be any permutation. Let $q(n)$ be the maximal number of f-queries made by C on the queried instance (see Definition 4). Then*

$$\Pr_{x \leftarrow \{0,1\}^n, h \leftarrow Inj}[f(x) \notin G_f] \leq \frac{2q(n)}{2^{n/2}} + \Pr_{x \leftarrow \{0,1\}^n, h \leftarrow Inj}[R^{f,\textsc{Solve}}(1^n, y) \neq f^{-1}(y)],$$

where both $f = h \circ \pi$ and G_f are as defined in the algorithm $\textsc{Encode}_n$ *(see line 2 Algorithm 2).*

Claim 13 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 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 13. *Let (R, T_R, C, T_C, p) be any deterministic f-oblivious many-one fully black-box reduction. Let $i \in \{0,1\}^*$ be any instance of the corresponding* TFNP *problem (defined by C). Let*

$$q = \max_{\substack{f \in Inj, s \in \{0,1\}^* \\ s.t.\ |s| \le p(|i|)}} |Q(C^f(i,s))|.$$

Let $k \in \mathbb{N}$ and let $Y \subseteq \{0,1\}^*$ be a set of size k and

$$\mathcal{F}_Y = \{f \mid f \in Inj \text{ and } Y \subseteq Im(f)\}.$$

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

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

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

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

□

Now we bound 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 does not query i on many different challenges $f(x')$, where x' is of the same length as x.

Recall that $Y_{i,n}$ is the set of challenges y of length n 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). Thus we want to show that there is a solution which does not query preimage of any y from $Y_{i,n}$. The problem is that it is not sufficient to show it for a single fixed preimage length n. As even though we have such a solution, there might be smaller $Y_{i,n'}$, 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 if Solve stops "protecting" $Y_{i,n}$ before $Y_{i,n'}$.

We show that if we stop "protecting" the sets $Y_{i,n}$ in order given by decreasing density ($\frac{|Y_{i,n}|}{2^n}$) this is not the case. 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., $\frac{|Y_{i,n'}|}{2^{n'}} \le \frac{|Y_{i,n}|}{2^n}$).

To bound this probability, we consider another function $g \in \text{Inj}$, which is similar to f, but we "hide" the preimages of all Y_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 $\text{Im}(f)$. 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 go back to f and show that with high probability this solution remains to be a solution and that it also does not query a preimage of any $y \in Y_i$.

Claim 14. *Let (R, T_R, C, T_C, p) be any f-oblivious many-one black-box reduction. Let $i \in \{0,1\}^*$ be any instance of the corresponding* TFNP *problem and*

$$q = \max_{\substack{f \in Inj, s \in \{0,1\}^* \\ s.t.\ |s| \le p(|i|)}} |Q(C^f(i,s))|.$$

Let $\alpha \colon \mathbb{N} \to [0,1]$ be any function, $Y \subseteq \{0,1\}^*$ be any finite set, and $\mu \in \mathrm{T}$ be any type. Let $\mathcal{F}_Y$ be the set defined as follows

$$\mathcal{F}_Y = \{f \in Inj \mid Y \subseteq Im(f), \tau(f) = \mu, \text{ and } \forall n \in \mathbb{N}\colon\ |Y \cap Im(f_n)| \le \alpha(n)2^n\}\,, \tag{7}$$

where $f_n = f \restriction \{0,1\}^n$. Let BAD_f denote the event that

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

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

$$\Pr_{f \leftarrow \mathcal{F}_Y}[BAD_f] \le \alpha(n)q.$$

Proof. We define the following set

$$\mathcal{H}_Y = \{h \in \mathrm{Inj} \mid \tau(h) = \mu \text{ and } Y \cap \mathrm{Im}(h) = \emptyset\}\,. \tag{8}$$

Let $X_{f,Y}$ be the set of preimages of Y in f (i.e., $X_{f,Y} = f^{-1}[Y]$).

Now we can bound the probability as follows (we prove the inequality below):

$$\Pr_{f \leftarrow \mathcal{F}_Y}[\mathrm{BAD}_f] \le \sup_{h \in \mathcal{H}_Y} \Pr_{f \leftarrow \mathcal{F}_Y}[\mathrm{BAD}_f \mid f \restriction \overline{X_{f,Y}} = h \restriction \overline{X_{f,Y}}] \tag{9}$$

Let $\mathcal{H}_{Y,f}$ be defined as follows:

$$\mathcal{H}_{Y,f} = \{h \in \mathcal{H}_Y \mid f \restriction \overline{X_{f,Y}} = h \restriction \overline{X_{f,Y}}\}.$$

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

$$|\mathcal{H}_{Y,f}| = \left|\{h \in \mathcal{H}_Y \mid f \restriction \overline{X_{f,Y}} = h \restriction \overline{X_{f,Y}}\}\right| = \tag{10}$$

$$= \prod_{\substack{n \in \mathbb{N},\ m = \mu(n) \\ \text{s.t. } \{0,1\}^m \cap Y \ne \emptyset}} \left(\binom{2^m - 2^n}{|Y \cap \{0,1\}^m|} (|Y \cap \{0,1\}^m|!)\right). \tag{11}$$

By definition, both $f, h \in \mathrm{Inj}$ (see equation 7 - definition of $\mathcal{F}_Y$ and equation 8 - definition of $\mathcal{H}_Y$), and furthermore $\mathrm{Im}(h) \cap Y = \emptyset$ (see equation 8 for definition of $\mathcal{H}_Y$). The equation 11 follows from the fact that there are $\binom{2^m-2^n}{|Y \cap \{0,1\}^m|}(|Y \cap \{0,1\}^m|)!$ possibilities how to define $h \restriction (X_f \cap \{0,1\}^n)$. Note that size of $\mathcal{H}_{Y,f}$ is finite, as Y is finite set and thus the product contains finitely many factors. Observe also that $|\mathcal{H}_{Y,f'}| = |\mathcal{H}_{Y,f}|$ as the size depends only on the type μ and set Y, but is not dependent on the exact mapping f.

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

$$\Pr_{f \leftarrow \mathcal{F}_Y}[\mathrm{BAD}_f \mid f \restriction \overline{X_{f,Y}} = h \restriction \overline{X_{f,Y}}] \le \alpha(n)q.$$

By the correctness of the underlying TFNP problem (see Definition 2 for the exact order of the quantifiers), there exists $s \in \{0,1\}^*$ such that $|s| \le p(i)$ and $C^h(i,s) = 1$. Let Q be the set of queries made on the solution $C^h(i,s)$, i.e., $Q = Q(C^h(i,s))$. Note that if $Q \cap X_{f,Y}$ is empty then all queries C makes on input i, s are answered equally for h as for f thus s is solution also with respect to f (i.e., $C^f(i,s) = 1$) and $Q(C^f(i,s))$ has empty intersection with $X_{f,Y}$ (as $Q(C^f(i,s)) = Q$). Thus we may rewrite the probability as follows:

$$\begin{aligned}\Pr_{f\leftarrow\mathcal{F}_Y}[\mathrm{BAD}_f \mid f\restriction\overline{X_{f,Y}} = h\restriction\overline{X_{f,Y}}] &\le \Pr_{f\leftarrow\mathcal{F}_Y}[X_{f,Y}\cap Q \neq \emptyset] \\ &\le \sum_{n\in\mathbb{N}} \Pr_{f\leftarrow\mathcal{F}_Y}[X_{f,Y}\cap Q\cap\{0,1\}^n \neq \emptyset]. \qquad \text{(by union bound)}\end{aligned}$$

We bound the probability for a single n and the corresponding output length $m = |f(0^n)|$. Let q_n denote the number of queries of length n, i.e., $q_n = |Q\cap\{0,1\}^n|$. We may bound the probability as follows:

$$\begin{aligned}\Pr_{f\leftarrow\mathcal{F}_Y}[X_{f,Y}\cap Q\cap\{0,1\}^n \neq \emptyset] &\le \sup_{k\in\mathbb{N}} \Pr_{\substack{f\leftarrow\mathcal{F}_Y\\ |X_{f,Y}\cap\{0,1\}^n|=k}}[X_{f,Y}\cap Q\cap\{0,1\}^n \neq \emptyset] &(12)\\ &\le \sup_{k\in\mathbb{N}} \frac{|Q\cap\{0,1\}^n|\cdot\binom{2^n-1}{k-1}}{\binom{2^n}{k}} &(13)\\ &= \sup_{k\in\mathbb{N}} \frac{k}{2^n}q_n &(14)\end{aligned}$$

The inequality 12 follows from the fact that condition $|X_{f,Y}\cap\{0,1\}^n| = k$ over different k creates a partition of the probabilistic space. The inequality 13 is computed as follows:

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

Now we use the definition of $\mathcal{F}_Y$ (see Equation 7) to bound $\frac{k}{2^n}$ and we get the following bound:

$$\Pr_{f\leftarrow\mathcal{F}_Y}[X_{f,Y}\cap Q\cap\{0,1\}^n \neq \emptyset] \le \sup_{k\in\mathbb{N}} \frac{k}{2^n}q_n \le \alpha(n)q_n. \tag{15}$$

Finally we summarize all the inequalities above:

$$\begin{aligned}\Pr_{f\leftarrow\mathcal{F}_Y}[\mathrm{BAD}_f] &\le \sup_{h\in\mathcal{H}_Y} \Pr_{f\leftarrow\mathcal{F}_Y}[\mathrm{BAD}_f \mid f\restriction\overline{X_{f,Y}} = h\restriction\overline{X_{f,Y}}] \qquad \text{(see inequality 9)}\\ &\le \sup_{h\in\mathcal{H}_Y}\sum_{n\in\mathbb{N}} \Pr_{f\leftarrow\mathcal{F}_Y}[X_{f,Y}\cap Q\cap\{0,1\}^n \neq \emptyset]\\ &\le \sup_{h\in\mathcal{H}_Y}\sum_{n\in\mathbb{N}} \alpha(n)q_n.\end{aligned}$$

Using the definition of $q_n = |Q\cap\{0,1\}^n|$ we can bound $\sum_{n\in\mathbb{N}} q_n = \sum_{n\in\mathbb{N}}|Q\cap\{0,1\}^n| = |Q| \le q$, by assumption. This concludes the proof as the sum $\sum_{n\in\mathbb{N}} \alpha(n)q_n$ can be upper bounded by $\alpha(n)q$ and thus $\Pr_{f\leftarrow\mathcal{F}_Y}[\mathrm{BAD}_f] \le \alpha(n)q$.

□

We are ready to lower bound the probability that we can compute x from $f(x)$ using the security reduction R and f defined on values different from x, i.e., we upper bound the following probability

$$\Pr_{\substack{x\leftarrow\{0,1\}^n,\ h\leftarrow\mathrm{Inj}\\ f=h\circ\pi}}[f(x)\notin G_f],$$

where $\pi\in\mathrm{Inj}_n^n$ is a permutation and G_f is as defined in the algorithm $\textsc{Encode}_n$ (see line 3 of Algorithm 2).

Proof (of Claim 12). Recall that $y\in\mathrm{Im}(f_n)$ 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., $f^{-1}(y) \notin Q_f^{\text{indir}}(R^{f,\textsc{Solve}}(1^n, y))$.

Thus, we bound the probability using union bound as follows:

$$\Pr_{\substack{x \leftarrow \{0,1\}^n,\ h \leftarrow \text{Inj} \\ f = h \circ \pi}} [f(x) \notin G_f] \leq \Pr_{\substack{x \leftarrow \{0,1\}^n,\ h \leftarrow \text{Inj} \\ f = h \circ \pi}} [x \notin \text{INV}_f] + \Pr_{\substack{x \leftarrow \{0,1\}^n,\ h \leftarrow \text{Inj} \\ f = h \circ \pi}} [x \in Q_f^{\text{indir}}(R^{f,\textsc{Solve}}(1^n, f(x)))].$$

By definition of INV_f (see line 2 of Algorithm 2)

$$\Pr_{\substack{x \leftarrow \{0,1\}^n,\ h \leftarrow \text{Inj} \\ f = h \circ \pi}} [x \notin \text{INV}_f] = \Pr_{\substack{x \leftarrow \{0,1\}^n,\ h \leftarrow \text{Inj} \\ f = h \circ \pi}} [R^{f,\textsc{Solve}}(1^n, f(x)) \neq x].$$

It suffices to show that

$$\Pr_{\substack{x \leftarrow \{0,1\}^n,\ h \leftarrow \text{Inj} \\ f = h \circ \pi}} [x \in Q_f^{\text{indir}}(R^{f,\textsc{Solve}}(1^n, f(x)))] \leq \frac{2q(n)}{2^{n/2}}. \tag{16}$$

As the TFNP instance queried defines a partition of the probability space we may bound:

$$\begin{aligned} &\Pr_{\substack{x \leftarrow \{0,1\}^n,\ h \leftarrow \text{Inj} \\ f = h \circ \pi}} [x \in Q_f^{\text{indir}}(R^{f,\textsc{Solve}}(1^n, f(x)))] && (17) \\ &\qquad \leq \sup_{i \in \{0,1\}^*} \Pr_{\substack{x \leftarrow \{0,1\}^n,\ h \leftarrow \text{Inj} \\ f = h \circ \pi}} [x \in Q_f^{\text{indir}}(R^{f,\textsc{Solve}}(1^n, f(x))) \mid i \in Q_{\textsc{Solve}}(R^{f,\textsc{Solve}}(1^n, f(x))]. && (18) \end{aligned}$$

This allows us to fix any instance $i \in \{0,1\}^*$ and bound the probability that x was indirectly queried only for the case when TFNP instance i was queried.

We use the following notation in the rest of the proof. Let $t_{i,f}^{\text{end}} = j$ if the algorithm $\textsc{Solve}^f(i)$ returns the solution in the j-th iteration of the while loop (see line 7 of Algorithm 1). Furthermore let $t_{i,f,n}^{\text{del}}$ denote the iteration of the same while loop in which $Y_{i,n}$ is removed from Y_i (see line 14 of Algorithm 1). Note that we set $t_{i,f,n}^{\text{del}} = \infty$ in the case when $Y_{i,n}$ is never removed from Y_i.

Let $\alpha \colon \mathbb{N} \to [0,1]$ be a function chosen later in the proof. We can rewrite the probability as follows:

$$\begin{aligned} &\Pr[x \in Q_f^{\text{indir}}(R^{f,\textsc{Solve}}(1^n, y))] = && (19) \\ &\quad = \Pr\left[x \in Q_f^{\text{indir}}(R^{f,\textsc{Solve}}(1^n, y)) \;\middle|\; t_{i,f}^{\text{end}} \leq t_{i,f,n}^{\text{del}}\right] \Pr\left[t_{i,f}^{\text{end}} \leq t_{i,f,n}^{\text{del}}\right] && (20) \\ &\quad + \Pr\left[x \in Q_f^{\text{indir}}(R^{f,\textsc{Solve}}(1^n, y)) \;\middle|\; t_{i,f}^{\text{end}} > t_{i,f,n}^{\text{del}} \text{ and } \frac{|Y_{i,n}|}{2^n} \leq \alpha(n)\right] \Pr\left[t_{i,f}^{\text{end}} > t_{i,f,n}^{\text{del}} \text{ and } \frac{|Y_{i,n}|}{2^n} \leq \alpha(n)\right] && (21) \\ &\quad + \Pr\left[x \in Q_f^{\text{indir}}(R^{f,\textsc{Solve}}(1^n, y)) \;\middle|\; t_{i,f}^{\text{end}} > t_{i,f,n}^{\text{del}} \text{ and } \frac{|Y_{i,n}|}{2^n} > \alpha(n)\right] \Pr\left[t_{i,f}^{\text{end}} > t_{i,f,n}^{\text{del}} \text{ and } \frac{|Y_{i,n}|}{2^n} > \alpha(n)\right], && (22) \end{aligned}$$

where the probability is over the choice of $h \leftarrow \text{Inj}$, $x \leftarrow \{0,1\}^n$ such that when we set $f = h \circ \pi$, $y = f(x)$ then $i \in Q_{\textsc{Solve}}(R^{f,\textsc{Solve}}(1^n, y))$ holds (the TFNP instance i is fixed and we chose uniformly at random x, h such that i is queried).

Observe that,

$$\Pr\left[x \in Q_f^{\text{indir}}(R^{f,\textsc{Solve}}(1^n, y)) \;\middle|\; t_{i,f}^{\text{end}} \leq t_{i,f,n}^{\text{del}}\right] = 0$$

where the probability is over the choice of $h \leftarrow \text{Inj}$, $x \leftarrow \{0,1\}^n$ such that when we set $f = h \circ \pi$, $y = f(x)$ then $i \in Q_{\textsc{Solve}}(R^{f,\textsc{Solve}}(1^n, y))$ holds. This holds because the returned solution does not contain preimage of any element from Y_i by definition and $y \in Y_i$ during the iteration $t_{i,f}^{\text{end}}$.

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

$$\begin{aligned}\Pr[x \in & Q_f^{\text{indir}}(R^{f,\textsc{Solve}}(1^n, y))] \\ & \leq \Pr\left[t_{i,f}^{\text{end}} > t_{i,f,n}^{\text{del}} \text{ and } \frac{|Y_{i,n}|}{2^n} \leq \alpha(n)\right] + \Pr\left[x \in Q_f^{\text{indir}}(R^{f,\textsc{Solve}}(1^n, y)) \;\middle|\; t_{i,f}^{\text{end}} > t_{i,f,n}^{\text{del}} \text{ and } \frac{|Y_{i,n}|}{2^n} > \alpha(n)\right],\end{aligned}$$

where the probability is over the choice of $h \leftarrow \text{Inj}$, $x \leftarrow \{0,1\}^n$ such that when we set $f = h \circ \pi$, $y = f(x)$ then $i \in Q_{\textsc{Solve}}(R^{f,\textsc{Solve}}(1^n, y))$ holds.

We bound the second probability using Claim 13 for instance i, $k = |Y_{i,n}| \geq \alpha(n)2^n$, the set $Y = Y_{i,n}$ and $q(n)$ as the upper bound on the number of f-queries made by C. This bounds the probability by

$$\Pr\left[x \in Q_f^{\text{indir}}(R^{f,\textsc{Solve}}(1^n, y)) \;\middle|\; t_{i,f}^{\text{end}} > t_{i,f,n}^{\text{del}} \text{ and } \frac{|Y_{i,n}|}{2^n} > \alpha(n)\right] \leq \frac{q(n)}{k} \leq \frac{q(n)}{\alpha(n)2^n},$$

where the probability is over the choice of $h \leftarrow \text{Inj}$, $x \leftarrow \{0,1\}^n$ such that when we set $f = h \circ \pi$, $y = f(x)$ then $i \in Q_{\textsc{Solve}}(R^{f,\textsc{Solve}}(1^n, y))$ holds.

We can use Claim 14 to bound the first probability. We actually bound the probability that $Y_{i,n}$ is not deleted before solve returns supposing that $Y_{i,n}$ has small density, i.e.,

$$\Pr\left[t_{i,f}^{\text{end}} > t_{i,f,n}^{\text{del}} \text{ and } \frac{|Y_{i,n}|}{2^n} \leq \alpha(n)\right] \leq \Pr\left[t_{i,f}^{\text{end}} > t_{i,f,n}^{\text{del}} \;\middle|\; \frac{|Y_{i,n}|}{2^n} \leq \alpha(n)\right],$$

where the probability is over the choice of $h \leftarrow \text{Inj}$, $x \leftarrow \{0,1\}^n$ such that when we set $f = h \circ \pi$, $y = f(x)$ then $i \in Q_{\textsc{Solve}}(R^{f,\textsc{Solve}}(1^n, y))$ holds.

We want to argue that $Y_{i,n}$ is not deleted before $\textsc{Solve}$ returns. 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 14 for the set Y which is a union of $Y_{i,n'}$ with small density (i.e., small $|Y_{i,n'}|/2^{n'}$). As the density is computed with respect to the length of the preimage (not the length of the image) it is practical to consider probability just over functions of the same type. For these reasons we upper bound the probability as follows:

$$\Pr\left[t_{i,f}^{\text{end}} > t_{i,f,n}^{\text{del}} \;\middle|\; \frac{|Y_{i,n}|}{2^n} \leq \alpha(n)\right] \leq \sup_{\mu \in \mathrm{T}} \Pr\left[t_{i,f}^{\text{end}} > t_{i,f,n}^{\text{del}} \;\middle|\; \frac{|Y_{i,n}|}{2^n} \leq \alpha(n) \text{ and } \tau(h) = \mu\right],$$

where the probability is over the choice of $h \leftarrow \text{Inj}$, $x \leftarrow \{0,1\}^n$ such that when we set $f = h \circ \pi$, $y = f(x)$ then $i \in Q_{\textsc{Solve}}(R^{f,\textsc{Solve}}(1^n, y))$ holds. Now we fix some type μ and continue just with functions of the type μ.

We need to determine the set Y we use in the Claim 14. Let Y_i^{del} be the set Y_i as defined in the algorithm $\textsc{Solve}$ in the iteration in which $Y_{i,n}$ is deleted. We want $Y \subseteq \{0,1\}^*$ to match the set Y_i^{del}. More precisely we want Y such that it can be partitioned into 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. $\forall n' \in \mathbb{N}\colon |Y_{i,n'}| \leq \alpha(n)/2^{n'}$.

We can bound the probability as follows

$$\Pr\left[t_{i,f}^{\text{end}} > t_{i,f,n}^{\text{del}} \;\middle|\; \frac{|Y_{i,n}|}{2^n} \leq \alpha(n) \text{ and } \tau(h) = \mu\right] \leq \sup_{Y} \Pr\left[t_{i,f}^{\text{end}} > t_{i,f,n}^{\text{del}} \;\middle|\; \frac{|Y_{i,n}|}{2^n} \leq \alpha(n), \tau(h) = \mu, \text{ and } Y_i^{\text{del}} = Y\right],$$

where the probability is over the choice of $h \leftarrow \text{Inj}$, $x \leftarrow \{0,1\}^n$ such that when we set $f = h \circ \pi$, $y = f(x)$ then $i \in Q_{\textsc{Solve}}(R^{f,\textsc{Solve}}(1^n, y))$ holds and the supremum is over all sets Y satisfying the conditions above.

Observe that the set Y_i^{del} satisfies all conditions on Y (i.e., we can choose Y in such a way that it matches Y_i^{del}). The first two conditions are satisfied trivially by setting each $Y_{i,n'}$ to the corresponding set in $\textsc{Solve}$.

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 $|Y_{i,n}|/2^n \leq \alpha(n)$ thus all other sets $Y_{i,n'} \subseteq Y_i^{\text{del}}$ have density at most $\alpha(n)$.

Now we fix any set Y satisfying the conditions above and we upper bound the probability using Claim 14 for instance i, $q(n)$ being an upper bound on the number of f-queries made by C, $\alpha(n)$ being the density of challenges y and our fixed set Y.

Recall the definition of $\mathcal{F}_Y$ from Claim 14:

$$\mathcal{F}_Y = \{f \in \text{Inj} \mid Y \subseteq \text{Im}(f) \text{ and } \forall n \in \mathbb{N}\colon\ |Y \cap \text{Im}(f_n)| \leq \alpha(n)2^n\}, \tag{23}$$

Observe that $\mathcal{F}_Y$ is not empty. Claim 14 gives us that

$$\Pr_{f \leftarrow \mathcal{F}_Y}[\forall s \in \{0,1\}^* \text{ such that } |s| \leq p(|i|)\colon \text{ either } C^f(i,s) \neq 1 \text{ or } Y \cap f[Q(C^f(i,s))] \neq \emptyset] \leq \alpha(n)q(n).$$

Observe that by the definition of $Y = Y_i^{\text{del}}$ and $\mathcal{F}_Y$ (see Claim 14), this bounds the probability that there was a suitable solution for returning before $Y_{i,n}$ was deleted from Y_i, i.e., it bounds the probability $t_{i,f}^{\text{end}} > t_{i,f,n}^{\text{del}}$ and thus we get

$$\Pr\left[t_{i,f}^{\text{end}} > t_{i,f,n}^{\text{del}} \;\middle|\; \frac{|Y_{i,n}|}{2^n} \leq \alpha(n) \text{ and } \tau(h) = \mu \text{ and } Y_i^{\text{del}} = Y\right] \leq \alpha(n)q(n),$$

where the probability is over the choice of $h \leftarrow \text{Inj}$, $x \leftarrow \{0,1\}^n$ such that when we set $f = h \circ \pi$, $y = f(x)$ then $i \in Q_{\text{SOLVE}}(R^{f,\text{SOLVE}}(1^n, y))$ holds.

Summarizing the above inequalities we get

$$\begin{aligned}\Pr[x \in & Q_f^{\text{indir}}(R^{f,\text{SOLVE}}(1^n,y))] \\ & \leq \Pr\left[t_{i,f}^{\text{end}} > t_{i,f,n}^{\text{del}} \text{ and } \frac{|Y_{i,n}|}{2^n} \leq \alpha(n)\right] + \Pr\left[x \in Q_f^{\text{indir}}(R^{f,\text{SOLVE}}(1^n,y)) \;\middle|\; t_{i,f}^{\text{end}} > t_{i,f,n}^{\text{del}} \text{ and } \frac{|Y_{i,n}|}{2^n} > \alpha(n)\right] \\ & \leq \frac{q(n)}{\alpha(n)2^n} + \alpha(n)q(n),\end{aligned}$$

where the probability is over the choice of $h \leftarrow \text{Inj}$, $x \leftarrow \{0,1\}^n$ such that when we set $f = h \circ \pi$, $y = f(x)$ then $i \in Q_{\text{SOLVE}}(R^{f,\text{SOLVE}}(1^n, y))$ holds.

Finally we set $\alpha(n) = \frac{1}{2^{n/2}}$ to get

$$\Pr_{\substack{x \leftarrow \{0,1\}^n,\ h \leftarrow \text{Inj} \\ f = h \circ \pi}}[x \in Q_f^{\text{indir}}(R^{f,\text{SOLVE}}(1^n, f(x)))] \leq \frac{q(n)}{\alpha(n)2^n} + \alpha(n)q(n) \leq \frac{2q(n)}{2^{n/2}}.$$

This concludes the proof of Claim 12 as we proved that

$$\begin{aligned}\Pr_{\substack{x \leftarrow \{0,1\}^n,\ h \leftarrow \text{Inj} \\ f = h \circ \pi}}[f(x) \notin G_f] & \leq \Pr[x \notin \text{INV}_f] + \Pr[x \in Q_f^{\text{indir}}(R^{f,\text{SOLVE}}(1^n, f(x)))] \\ & \leq \Pr_{x \leftarrow \{0,1\}^n, h \leftarrow \text{Inj}}[R^{f,\text{SOLVE}}(1^n, y) \neq f^{-1}(y)] + \frac{2q(n)}{2^{n/2}}.\end{aligned}$$

□

Finally, we upper bound the expected size of encoding produced by ENCODE_n (see Algorithm 2) stated in Lemma 4.

Proof (of Lemma 4). We denote $E_h(\pi)$ the size of encoding, i.e., $E_h(\pi) = |\text{ENCODE}_n^h(\pi)|$ and set $f = h \circ \pi$. First we bound the size of encoding for permutation π in the case that ENCODE_n returned $\mathcal{M} =$

$(1, |X_f|, Y_f, X_f, \sigma)$, i.e., $|X_f| \geq 2^{0.6n}$. We denote the size of X_f by A and set $N = 2^n$. As stated in Claim 10 the size of encoding from algorithm $\textsc{Encode}_n$ in case $A \geq 2^{0.6n}$ is

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

We bound the last term as follows:

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

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

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

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

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

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

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

Now we bound the probability that $|X_f| \geq 2^{0.6n}$. By assumption of the lemma we can bound $(2q(n)+1) \leq 2^{0.2n}$ and we get

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

By Claim 12 we can bound the expected size of $\overline{G_f}$ as follows:

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

We can use the Markov inequality to bound

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

Combining these inequalities we get:

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

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

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

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

5 Extensions

5.1 Non-Adaptive Reductions

It is possible to extend our proof from Section 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 3.

Notice that the algorithms Solve, Encode$_n$, Decode$_n$, and SolveSim 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 1), that Decode$_n$ correctly decodes the permutation (Lemma 2), and that the encoding is prefix-free (Lemma 3) remain unchanged as well. The statement of Lemma 4 also holds without a change.

The only change in our proof would be in the proof of Claim 12 in Equation 17, where we are upper bounding the probability of an indirect hit. To deal with non-adaptive reductions, we would have to use union bound to bound the probability that there would be a queried instance causing an indirect hit. Thus the supremum in Equation 18 will be multiplied by $T_R(n + 100n^{\log(n)})$ (the upper bound on the number of Solve queries).

Accordingly, the bound on the indirect hit (Equation 16) changes to

$$\Pr_{\substack{x\in\{0,1\}^n,\ h\in\mathrm{Inj}\\ f=h\circ\pi}}[x \in Q_f^{\mathrm{indir}}(R^{f,\textsc{Solve}}(1^n, f(x)))] \leq \frac{T_R(n + 100n^{\log(n)}) \cdot 2q(n)}{2^{n/2}}. \qquad (24)$$

```
Algorithm 5: The oracle Solve (for a randomized security reduction).
    Hardwired     : a fully black-box construction $(R, T_R, C, T_C, p)$ of a hard TFNP problem from iOWF
    Oracle access: an injective function $f = \{f_n\}_{n\in\mathbb{N}} \in \mathrm{Inj}$
    Input         : an instance $i \in \{0,1\}^*$
    Output        : a solution $s \in \{0,1\}^*$ such that $C^f(i,s) = 1$
 1  Compute $Y_i = \bigcup_{n=1}^{T_C(|i|+p(|i|))} \left\{ y \in \mathrm{Im}(f_n) \mid \Pr_{r\leftarrow\{0,1\}^*}[i \in Q_{\textsc{Solve}}(R^{f,\textsc{Solve}}(1^n, y; r))] > 0 \right\}$
 2  for $y \in Y_i$ do
 3      Let $n = |f^{-1}(y)|$
 4      Set // compute probability reduction runs on $y$ conditioned on $i$ has been queried
 5
            $$p_i(y) = \frac{\Pr_{r\leftarrow\{0,1\}^*}\left[i \in Q_{\textsc{Solve}}(R^{f,\textsc{Solve}}(1^n, y; r))\right]}{\Pr_{r\leftarrow\{0,1\}^*, y'\leftarrow Y_i, |y'|=|y|}\left[i \in Q_{\textsc{Solve}}(R^{f,\textsc{Solve}}(1^n, y'; r))\right]}$$
 6  end
 7  Compute $S_{i,f} = \left\{ s \in \{0,1\}^* \mid |s| \le p(|i|) \text{ and } C^f(i,s) = 1 \right\}$
 8  while True do
 9      $B_{i,f} = \{ s \in S_{i,f} \mid f[Q(C^f(i,s))] \cap Y_i = \emptyset \}$
10      if $B_{i,f} \neq \emptyset$ then
11          return lexicographically smallest $s \in B_{i,f}$
12      end
13      Choose $y \in Y_i$ such that $p_i(y)$ is minimized.
14      Set $Y_i = Y_i \setminus \{y\}$
15  end
```

This is still $\mathcal{O}(n^{\mathrm{polylog}(n)})$. This propagates in the proof of Lemma 4 when bounding

$$\mathbb{E}_{\substack{\pi\leftarrow \mathrm{Inj}_n^n,\ h\leftarrow \mathrm{Inj} \\ f=h\circ\pi}}[|\overline{G_f}|]$$

in the same way as we bound the term by $2^{0.2n}$. Therefore, we can prove the following strengthening of Theorem 1.

Theorem 15. *There is no fully black-box construction (R, T_R, C, T_C, p) of a hard* TFNP *problem from injective one-way functions with deterministic f-oblivious non-adaptive reduction with success probability at least $2^{-0.1n}$ such that both running times $T_R, T_C \in O(n^{polylog(n)})$.*

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 iOWFwith *randomized security reductions.*

First, we need to change our algorithm Solve. One could imagine that the construction has some instance i created for a concrete challenge y, on which R queries i with high probability. But R might also query the instance i for many other challenges y' (on each of them with small probability) to hide the real challenge y. Thus we need to take the probability of querying the instance i into account.

Our new Solve works as shown in Algorithm 5. Note that it is an extension of the previous algorithm (i.e., if the security reduction ignores its random bits, Solve removes strings from Y_i in the same order but Algorithm 5 removes challenges one by one like instead of in "batches" $Y_{i,n}$ like Algorithm 1). The proof that the new Solve returns a solution is fairly similar to the proof of Lemma 1. Note that due to f-obliviousness we are still able to simulate this algorithm (all added computation depends only on the $\mathrm{Im}(f)$ and is independent of the concrete mapping f).

We also need to change the algorithms Encode$_n$ and Decode$_n$ as these algorithms need to share the randomness used for the reduction. Thus we equip them with a set of strings r_y for $y \in \mathrm{Im}(h_n)$, where every

r_y will be used for running the security reduction on challenge y. Note that we can bound the length of each r_y by $T_R(n + 100n^{\log n})$. This is because we can bound the input on which R is run by $n + 100n^{\log n}$ and thus bound the running time of R by $T_R(n + 100n^{\log n})$. If r_y is longer the reduction is not able to use it.

We want encoding to be uniquely decodable and prefix-free. We show this not just for any choice of $h \in \mathrm{Inj}$ but also for any choice of the random bits r_y, where $y \in \mathrm{Im}(h_n)$. The proofs follows the same blueprint as the proofs of Lemmas 2 and 3.

The main difference is in the part when we are bounding the size of the encoding (Section 4.4). We need to change Claims 13 and 14 to reflect the fact that we are removing challenges y from the set Y_i based on the probabilities $p_i(y)$.

Both statements of Claims 13 and 14 use a finite set of challenges Y. In our new version of these claims we partition the set $Y = \cup_{n\in\mathbb{N}} Y_n$ where each $Y_n \subseteq \{0,1\}^{\mu(n)}$ (where μ is a fixed function type) and for each Y_n we have a distribution $\mathcal{D}_{Y,n}$.

Here we describe an alternative to Claim 13. Instead of the set Y from the claim we consider Y with the distributions $\mathcal{D}_{Y,n}$, note that for this we need to consider just functions of some fixed type μ (thus $\mathcal{F}_Y$ is restricted to functions of type μ). We take $Y' = \left\{y \in Y \mid \Pr_{z\leftarrow\mathcal{D}_Y}[z = y] \le \gamma\left(|f^{-1}(y)|\right)\right\}$. By union bound ($k$-th query was on a preimage of some $y \in Y$) we can show that for any $f \in \mathrm{Inj}$ and any $s \in \{0,1\}^*, |s| \le p(|i|)$:

$$\forall n \in \mathbb{N}\colon \Pr_{y\leftarrow\mathcal{D}_{Y,n}}[y \in Y' \text{ and } f^{-1}(y) \in Q(C^f(i,s))] \le q\gamma(n)\,.$$

In the alternative to Claim 14 we also use any finite set Y. Now we assume that for each $n \in \mathbb{N}$ and for each $y \in Y_n$ $\Pr_{z\leftarrow\mathcal{D}_{Y,n}}[z = y] \ge \gamma(n)$. Recall the definition of $\mathcal{F}_y$ (Claim 14):

$$\mathcal{F}_Y = \{f \in \mathrm{Inj} \mid Y \subseteq \mathrm{Im}(f), \tau(f) = \mu, \text{ and } \forall n \in \mathbb{N}\colon\ |Y \cap \mathrm{Im}(f_n)| \le \alpha(n)2^n\}\,. \tag{25}$$

Observe that for each n we have $|Y \cap \mathrm{Im}(f_n)| = |Y_n| \le 1/\gamma(n)$ and thus we may use Claim 14 for α defined as $\forall n \in \mathbb{N}\colon \alpha(n) = \frac{1}{\gamma(n)2^n}$ to get that

$$\Pr_{f\leftarrow\mathcal{F}_Y}[\forall s \in \{0,1\}^* \text{ such that } |s| \le p(|i|) \text{ either } C^f(i,s) \ne 1 \text{ or } Y \cap f[Q(C^f(i,s))] \ne \emptyset] \le \frac{q}{\gamma(n)2^n}.$$

To put things together like in Claim 12 we again follow the flow of the original proof. We need to redefine the deletion time $t^{\mathrm{del}}_{i,f,n}$ when the set $Y_{i,n}$ was deleted from Y_i. We instead work with the time $t^{\mathrm{del}}_{i,f,y}$ when the single challenge y was deleted for each $y \in Y_i$.

The bound Equation 19 is substituted by the following:

$$\begin{aligned}
&\Pr[x \in Q_f^{\mathrm{indir}}(R^{f,\textsc{Solve}}(1^n, y))] = \\
&\quad= \Pr\left[x \in Q_f^{\mathrm{indir}}(R^{f,\textsc{Solve}}(1^n, y)) \;\middle|\; t^{\mathrm{end}}_{i,f} \le t^{\mathrm{del}}_{i,f,y}\right] \Pr\left[t^{\mathrm{end}}_{i,f} \le t^{\mathrm{del}}_{i,f,y}\right] \\
&\quad+ \Pr\left[x \in Q_f^{\mathrm{indir}}(R^{f,\textsc{Solve}}(1^n, y)) \;\middle|\; t^{\mathrm{end}}_{i,f} > t^{\mathrm{del}}_{i,f,y} \text{ and } p_i(y) \ge \gamma(n)\right] \Pr\left[t^{\mathrm{end}}_{i,f} > t^{\mathrm{del}}_{i,f,y} \text{ and } p_i(y) \ge \gamma(n)\right] \\
&\quad+ \Pr\left[x \in Q_f^{\mathrm{indir}}(R^{f,\textsc{Solve}}(1^n, y)) \;\middle|\; t^{\mathrm{end}}_{i,f} > t^{\mathrm{del}}_{i,f,y} \text{ and } p_i(y) < \gamma(n)\right] \Pr\left[t^{\mathrm{end}}_{i,f} > t^{\mathrm{del}}_{i,f,y} \text{ and } p_i(y) < \gamma(n)\right],
\end{aligned}$$

Now similarly as in the previous proof we show that the first summand is zero, the second can be bounded using our alternative of Claim 14 and the third by the alternative of Claim 13 (where we additionally need to fix the type of the function similarly as we did when we were using Claim 14 in the original proof).

After the averaging argument which chooses the function h we average again to get suitable randomness for the security reduction (both are independent of π).

This can be combined with the previous changes to get the full strength theorem:

Theorem 16. *There is no fully black-box construction (R, T_R, C, T_C, p) of a hard* TFNP *problem from injective one-way functions with randomized f-oblivious non-adaptive reduction with success probability at least $2^{-0.1n}$ such that both running times $T_R, T_C \in O(n^{polylog(n)})$.*

6 Conclusions

In this work, we have shown that there are intrinsic barriers preventing *simple* fully black-box constructions of hard TFNP problems from injective one-way functions. The main technical contribution of our work is the technique of designing a "TFNP-breaker" oracle SOLVE which depends on the reduction.

The natural direction towards extending our results would be attempting to lift the restriction to f-oblivious and non-adaptive reductions. One reason for why this might be challenging is that a black-box separation of TFNP and injective one-way functions would subsume the separation of collision-resistant hash functions and one-way functions which already has a non-trivial proof.

References

Bae14. Paul Baecher. Simon's circuit. *IACR Cryptol. ePrint Arch.*, 2014:476, 2014.

BCE$^+$98. 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.

BD19. 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.

BDV17. Nir Bitansky, Akshay Degwekar, and Vinod Vaikuntanathan. Structure vs. hardness through the obfuscation lens. In *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part I*, pages 696–723, 2017.

BG20. 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.

BKSY11. 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.

BPR15. 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.

Bur06. Joshua Buresh-Oppenheim. On the TFNP complexity of factoring. Unpublished, http://www.cs.toronto.edu/~bureshop/factor.pdf, 2006.

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

CHK$^+$19a. 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, 2019.

CHK$^+$19b. 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, 2019.

CT12. Thomas M. Cover and Joy A. Thomas. *Elements of information theory.* John Wiley & Sons, 2012.

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

DP11. 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.

EFKP20. Naomi Ephraim, Cody Freitag, Ilan Komargodski, and Rafael Pass. Continuous verifiable delay functions. In Anne Canteaut and Yuval Ishai, editors, *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.

FG18. Aris Filos-Ratsikas and Paul W. Goldberg. Consensus halving is PPA-complete. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 51–64, 2018.

FG19. Aris Filos-Ratsikas and Paul W. Goldberg. The complexity of splitting necklaces and bisecting ham sandwiches. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 638–649, 2019.

Fis12. Marc Fischlin. Black-box reductions and separations in cryptography. In *AFRICACRYPT*, volume 7374 of *Lecture Notes in Computer Science*, pages 413–422. Springer, 2012.

GH19. Paul W. Goldberg and Alexandros Hollender. The hairy ball problem is PPAD-complete. In *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, pages 65:1–65:14, 2019.

GL89. Oded Goldreich and Leonid A. Levin. A hard-core predicate for all one-way functions. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing, May 14-17, 1989, Seattle, Washigton, USA*, pages 25–32, 1989.

GMR01. 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.

GPS16. 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.

GT00. Rosario Gennaro and Luca Trevisan. Lower bounds on the efficiency of generic cryptographic constructions. In *FOCS*, pages 305–313. IEEE Computer Society, 2000.

HHRS15. 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.

HNY17. 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.

HR04. 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.

HY17. Pavel Hubáček and Eylon Yogev. Hardness of continuous local search: Query complexity and cryptographic lower bounds. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 1352–1371, 2017.

IR89. R. Impagliazzo and S. Rudich. Limits on the provable consequences of one-way permutations. In *Proceedings of the Twenty-first Annual ACM Symposium on Theory of Computing*, STOC '89, pages 44–61, New York, NY, USA, 1989. ACM.

Jeř16. Emil Jeřábek. Integer factoring and modular square roots. *J. Comput. Syst. Sci.*, 82(2):380–394, 2016.

JPY88. David S. Johnson, Christos H. Papadimitriou, and Mihalis Yannakakis. How easy is local search? *Journal of Computer and System Sciences*, 37(1):79 – 100, 1988.

KPR$^+$13. Shiva Kintali, Laura J. Poplawski, Rajmohan Rajaraman, Ravi Sundaram, and Shang-Hua Teng. Reducibility among fractional stability problems. *SIAM J. Comput.*, 42(6):2063–2113, 2013.

KS17. 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.

MM11. Takahiro Matsuda and Kanta Matsuura. On black-box separations among injective one-way functions. In *TCC*, volume 6597 of *Lecture Notes in Computer Science*, pages 597–614. Springer, 2011.

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

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

PV10. 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.

PV19. 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.

RS10. Alon Rosen and Gil Segev. Chosen-ciphertext security via correlated products. *SIAM J. Comput.*, 39(7):3058–3088, 2010.

RSS17. 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*, pages 747–776, 2017.

RTV04. 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.

Rud88. Steven Rudich. *Limits on the Provable Consequences of One-way Functions*. PhD thesis, EECS Department, University of California, Berkeley, Dec 1988.

Sim98. Daniel R. Simon. Finding collisions on a one-way street: Can secure hash functions be based on general assumptions? In Kaisa Nyberg, editor, *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.

Wee07. Hoeteck Wee. Lower bounds for non-interactive zero-knowledge. In *TCC*, volume 4392 of *Lecture Notes in Computer Science*, pages 103–117. Springer, 2007.