On Average-Case Hardness in TFNP from One-Way Functions*

Pavel Hubáček[1][0000−0002−6850−6222], Chethan Kamath[2], Karel Král[1][0000−0002−6557−9354], and Veronika Slívová[1][0000−0003−4514−9098]

[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) hard TFNP problem 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).

* 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 1568819, and by the Charles University grant SVV-2017-260452. The second author is supported by the IARPA grant IARPA/2019-19-020700009.

Keywords: TFNP · PPAD · Average-case hardness · One-way functions · Black-box separations.

1 Introduction

The complexity class TFNP of *total* search problems [?], i.e., with syntactically guaranteed existence of a solution for all instances, holds a perplexing place in the hierarchy of computational complexity classes. The standard method for arguing computational hardness in TFNP is via clustering these problems into subclasses characterised by the existential argument guaranteeing their totality [?]. 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, [?, ?, ?, ?, ?] 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 [?], 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 [?, ?, ?, ?, ?, ?, ?, ?, ?]. The end of the line in this 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 [?]) 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 (i.e., languages) in NP∩coNP. The existence of a hard-on-average decision problem in NP∩coNP follows from the existence of hard-core predicates for any one-way permutation [?]. However, the existence of injective one-way functions is insufficient for black-box constructions of hard-on-average distributions for languages in NP ∩ coNP even assuming indistinguishability obfuscation [?] (in fact, [?] ruled out even black-box constructions of worst-case hardness in NP∩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 [?, ?] or under derandomization-style assumptions and one-way functions [?]. On the other hand, no analogue of the impossibility result for basing average-case hardness in NP ∩ coNP on (injective) one-way functions [?] is currently known for TFNP. Rosen, Segev, and Shahaf [?] 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. [?] also provided some insight into the structure of hard-on-average distributions in TFNP. They showed that any hard-on-average distribu-

tion 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 [?] 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 [?] 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 oracle-aided circuit C^f_y computing the function $\mathsf{C}^f_y(x) = f(x) \oplus y$. It is not hard to see that the instance of PIGEON C^f_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 (on average) TFNP problem is extremely simple in various aspects:

- The construction is *fully black-box*, i.e., the PIGEON instance can be implemented via an oracle-aided circuit treating the one-way permutation as a black-box and the reduction inverts when given oracle access to an arbitrary solver for PIGEON.
- The reduction is *many-one*, i.e., a single call to a PIGEON-solving oracle suffices for finding the preimage of y.

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

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 $\mathsf{C}^h(x) = h(x) \,\|\, 1$, where $\|$ represents the operation of string concatenation. Since C^h concatenates the value $h(x)$ with 1 for any input x, it never maps to the all-zero string and, therefore, has the same collisions as h. Note that, unlike in the above construction from one-way permutations, the instances resulting from collision-resistant hash functions do not have a unique solution. In fact, there are always at least 2^{n-1} 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. [**?**]. 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 restricted, our results are the first step towards the full-fledged black-box separation of TFNP and (injective) one-way functions. We note that

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

the full-fledged separation would necessarily subsume the known separation of collision-resistant hash functions and injective one-way functions [?], for which, despite the recent progress, there are only non-trivial proofs [?, ?, ?].

1.2 Our Techniques

Our results employ the framework of black-box separations [?, ?, ?]. The approach suggested in [?] 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 [?, ?], 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 [?]. 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. Designing such a SOLVE oracle and then arguing that it does not help inverting injective one-way function, which we carry out using the reconstruction paradigm of Gennaro ond Trevisan [?], constitute the main technical challenges. Before giving an overview of these two steps, we explain the structural insight that is key to our separation, and guided us in the design of the two steps.

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 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 [**?**], e.g., the works which employed the collision-finding oracle of Simon [**?**, **?**, **?**, **?**, **?**]. 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 [**?**], 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 [**?**] who, nevertheless, did not have to 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 [**?**].

[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.

Applying the compression argument. Two important subtleties arise in the proof when we try to turn the reduction into a pair of compression and decompression algorithms, which we explain next. First, the reconstruction paradigm is conventionally applied to random permutations [**?**, **?**], 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 [**?**] 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 [**?**, **?**]) 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). An orthogonal notion of a π-oblivious construction appears in the work of Wee [?]. 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 [?] 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 [?] 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 [?], which captures the computational complexity of finding Nash equilibria in bimatrix games [?, ?]. Other subclasses of TFNP include:

- PPA [?], which captures computational problems related to the *parity argument* like Borsuk-Ulam theorem or fair division [?];
- PLS [?], defined to capture the computational complexity of problems amenable to *local search* and its "continuous" counterpart CLS $\subseteq$ PLS [?] (in fact CLS $\subseteq$ PLS $\cap$ PPAD), 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 [?] and PWPP $\subseteq$ PPP [?], 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 [?] 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 [?] 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 [?], building on the work of Buresh-Oppenheim [?], 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. [?] who reduced hardness in PPAD to indistinguishability obfuscation (and injective OWFs) expanding on Abbot, Kane, and Valiant [?]; Hubáček and Yogev [?] extended this result to CLS $\subseteq$ PLS$\cap$PPAD. The underlying assumption was relaxed further to cryptographic assumptions that are more plausible than indistinguishability obfuscation in [?, ?, ?]. Using similar ideas,

Bitansky and Gerichter [?] presented a construction for hard-on-average distributions in the complexity class PLS in the random oracle model. Building on these results, a flurry of recent works [?, ?, ?, ?, ?] further weakened the assumptions required for proving average-case hardness in CLS to sub-exponential hardness of learning with errors, bringing us closer to proving average-case hardness in CLS under a standard *concrete* cryptographic assumption.

One-way functions and TFNP. Hubáček et al. [?] 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 [?] recently complemented the [?] 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. [?] 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 [?], 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 [?] argument has been exploited to its limits in [?], 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 $\mathsf{NP} \cap \mathsf{coNP}$ in a black-box way [?].

2 Preliminaries

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

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

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

Notation 3 (Injective functions) *We denote by Inj_n^m the set of all injective functions from $\{0,1\}^n$ to $\{0,1\}^m$. For the special case when $n = m$ we get the set of all permutations on $\{0,1\}^n$.*

The set Inj is the set of all functions $f\colon \{0,1\}^* \to \{0,1\}^*$, such that f can be interpreted as a sequence $f = \left\{ f_n \mid f_n \in Inj_n^{m(n)} \right\}_{n\in\mathbb{N}}$ 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 $f \restriction \{0,1\}^n$ (i.e., restriction of f to the domain $\{0,1\}^n$.), where $f \in Inj$.

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

Notation 4 (Function composition) *Let X, Y, Z be any sets such that $X \subseteq Y$ and let $f\colon X \to Y$ and $g\colon Y \to Z$. We define the function $g \circ f\colon Y \to Z$ as:*

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

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

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

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

Corollary 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. The entropy of a uniformly randomly chosen permutation from Inj_n^n is $\log(2^n!)$ as we choose uniformly at random from $2^n!$ distinct permutations. By Proposition 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 (worst-case) hard TFNP problem from an injective one-way function.

Definition 2 (Fully black-box construction of a worst-case hard TFNP problem from iOWF). *A* fully black-box construction of a worst-case 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:*** *For all* $f \in Inj$*,* $n \in \mathbb{N}$*, and* $y, i, s \in \{0,1\}^*$*, the algorithm* $R^f(1^n, y)$ *halts in time* $T_R(|y|)$*, and 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* $|s| \leq p(|i|)$ *and* $C^f(i, s) = 1$*, i.e., for any instance of the* TFNP *problem there exists a solution of polynomial length.*
3. ***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 $n \in \mathbb{N}$*,*

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

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^f_y, x), respectively (C^f_y, x, x'), outputs 1 if and only if $\mathsf{C}^f_y(x) = 0^n$, respectively $\mathsf{C}^f_y(x) = \mathsf{C}^f_y(x')$. The reduction algorithm $R(1^n, y)$ simply queries the TFNP solver Solve with the instance $i = \mathsf{C}^f_y$, i.e., a circuit computing the function $\mathsf{C}^f_y(x) = f(x) \oplus y$, and outputs the solution s returned by Solve for which, by construction, $f(s) = y$.

Below, we provide some additional remarks on important points in the above definition.

Reduction-specific Solve. Let us emphasize the order of quantifiers restricting the security reduction in Definition 2:

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

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

Constructions of worst-case vs. average-case hardness in TFNP. Our Definition 2 considers constructions of a *worst-case* hard TFNP problem – the reduction has access to Solve which is promised to return a solution to *any* instance of the TFNP problem. To capture constructions of *average-case* hardness in TFNP, we would need to extend the construction with an efficiently sampleable distribution D of instances of the TFNP problem and require the reduction to invert when given access to any Solve that returns solutions for instances coming from the specific distribution D (see Definition 5.1 in [**?**]). However, given that we are proving a black-box separation, showing impossibility for worst-case hardness is a stronger result.

The quality of Solve. Note that we consider security reductions which 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 also makes our negative result stronger – it is potentially easier to give a reduction when given access to Solve which is guaranteed to always return a solution.

Direct and indirect queries to f. The security reduction R can learn something about f in various ways. It may query f directly or the information might be deduced from the solution of some queried instance of the TFNP problem returned by 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 5 (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 $Q_{\text{Solve}}(R^{f,\text{Solve}}(1^n,y))$ 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 $Q_f^{dir}(R^{f,\text{Solve}}(1^n,y))$ 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 $Q_f^{indir}(R^{f,\text{Solve}}(1^n,y))$ 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 $i \in Q_{\text{Solve}}(R^{f,\text{Solve}}(1^n,y))$ and $s = \text{Solve}^f(i)$.*

– *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.

Restrictions on the power of the reduction. We consider various 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 $f \in Inj$, for any oracle* Solve *and for all $y \in \{0,1\}^*$, $R^{f,\textsc{Solve}}(1^n, y)$ makes a single query to the oracle* Solve.

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

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

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 3.3, we explain how to extend this result to rule out fully black-box constructions even with a *randomized* f-oblivious *non-adaptive* reduction. For lack of space, we omit the proofs of the technical lemmata and instead, for interested readers, provide pointers to the appropriate part of full version [**?**].

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

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 $C^f(i,s) = 1$

Compute $Y_i = \bigcup_{n=1}^{T_C(|i|+p(|i|))} \left\{y \in \mathrm{Im}(f_n) \mid i \in Q_{\textsc{Solve}}(R^{f,\textsc{Solve}}(1^n, y))\right\}$
Compute $N_i = \{n \in \mathbb{N} \mid Y_i \cap \mathrm{Im}(f_n) \neq \emptyset\}$
for $n \in N_i$ **do**
  Compute $Y_{i,n} = Y_i \cap \mathrm{Im}(f_n)$
end
Compute $S_{i,f} = \left\{s \in \{0,1\}^* \mid |s| \leq p(|i|) \text{ and } C^f(i,s) = 1\right\}$
while *True* **do**
  $B_{i,f} = \{s \in S_{i,f} \mid f[Q(C^f(i,s))] \cap Y_i = \emptyset\}$
  if $B_{i,f} \neq \emptyset$ **then**
    return lexicographically smallest $s \in B_{i,f}$
  end
  Choose $n \in N_i$ such that $\frac{|Y_{i,n}|}{2^n}$ is maximized.
  Set $N_i = N_i \setminus \{n\}$
  Set $Y_i = Y_i \setminus Y_{i,n}$
end

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. That is, we design an oracle O relative to which (injective) one-way functions exist but TFNP is broken (even in the worst case). We follow the two-oracle approach [?], and, therefore, our oracle $O = (f, \textsc{Solve})$ consist of:

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

That a random injective function is one-way is a well-established result (see, e.g., Claim 5.3 in [?] for the more general case of random functions). The bulk of technical work revolves around showing that f remains one-way even in the presence of SOLVE. For any fully black-box construction with a deterministic f-oblivious many-one reduction, we provide an oracle SOLVE which finds a solution for any TFNP instance (i.e., TFNP is easy in the presence of SOLVE) and argue that it does not help the reduction in inverting injective one-way functions. The description of our oracle SOLVE is given in Algorithm 1 and it is explained below.

Oracle SOLVE*:* Let (R, T_R, C, T_C, p) be the construction of a hard TFNP problem from injective one-way function with a deterministic f-oblivious many-one security reduction 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 the input lengths for the preimages x such that the reduction $R^{f,\textsc{Solve}}(1^n, f(x))$ queries the instance i. Solve then splits Y_i into subsets $Y_{i,n}$ using the input lengths of interest in N_i. Finally, Solve computes the set $S_{i,f}$ which is the set of *all* possible solutions for the instance i.

The strategy of Solve is to return a solution from the set of "benign" solutions $B_{i,f}$, which do not induce any query to preimages of the protected challenges in Y_i. If there is any such benign solution then Solve simply halts and returns the lexicographically smallest one. Unfortunately, it might 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 often enough, as any fixed solution induces only a bounded number of queries to f (bounded by T_C). 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 Solve to return a solution revealing preimages of challenges from the auxiliary set $Y_{i,n}$ maximizing $\frac{|Y_{i,n}|}{2^n}$. If the fraction $\frac{|Y_{i,n}|}{2^n}$ is small then Solve is able to find a benign solution which protects the preimages of length n (see [?, Claim 13]). 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

[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, which allows SolveSim to simulate Solve without querying f on too many inputs.

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

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 $\textsc{Decode}_n$
Input : a permutation $\pi \in \mathrm{Inj}_n^n$ on $\{0,1\}^n$
Output : an encoding $\mathcal{M}$ of π

$f = h \circ \pi$, i.e., $f(x) = \begin{cases} h(\pi(x)) & \text{for all } x \text{ of length } n \\ h(x) & \text{otherwise} \end{cases}$
$\mathrm{INV}_f = \left\{ y \in \mathrm{Im}(h_n) \mid R^{\textsc{Solve},f}(1^n, y) = f^{-1}(y) \right\}$
$G_f = \left\{ y \in \mathrm{INV}_f \mid f^{-1}(y) \notin Q_f^{\mathrm{indir}}(R^{f,\textsc{Solve}}(1^n, y)) \right\}$
$Y_f = \emptyset$ 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 $\mathcal{M} = (1, |X_f|, Y_f, X_f, \sigma = f \restriction (\{0,1\}^n \setminus X_f)) \in \{0,1\}^{1+n+\left\lceil \log \binom{2^n}{|Y_f|} \right\rceil + \left\lceil \log \binom{2^n}{|X_f|} \right\rceil + \left\lceil \log\left(|\{0,1\}^n \setminus X_f|!\right) \right\rceil}$
end

solution can induce queries to only a small number of preimages of challenges from the set $Y_{i,n}$ (see [?, Claim 12]).

In order to show formally that Solve does not help in inverting the injective one-way function, we employ an incompressibility argument similar to [?]. 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 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 Solve oracle given access only to a partially defined oracle f (as we are reconstructing f). For the description of the algorithm SolveSim, which simulates the Solve oracle for the purpose of decoding in $\textsc{Decode}_n$, 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 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 – the part of f corresponding to 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 permutation π would use fewer bits. Thus, $\textsc{Encode}_n$ simply outputs the trivial encoding when X_f is too small. The first bit of the encoding distinguishes between 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 explicit 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

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 $\pi \in \mathrm{Inj}_n^n$

Parse $\mathcal{M} = (b, \mathcal{M}')$, where $b \in \{0,1\}$
if $b = 0$ **then**
  Decode π from $\mathcal{M}'$
  return π
end
Parse $\mathcal{M}' = (|X_f|, Y_f, X_f, \sigma)$
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
while $Y_f \neq \emptyset$ **do**
  Pick lexicographically smallest $y \in Y_f$
  Let $f''(x) = \begin{cases} y & \text{for all } x \in \mathrm{Dom}(h) \setminus \mathrm{Dom}(f') \\ f'(x) & \text{otherwise} \end{cases}$
  $x = R^{f'', \textsc{SolveSim}(h, f', \cdot)}(1^n, y)$
  Let $f'(x) = y$
  Set $Y_f = Y_f \setminus \{y\}$
end
return $\pi = (h^{-1} \circ f') \restriction \{0,1\}^n$

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 $\mathrm{Im}(f) \setminus Y_f$. 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 $\mathrm{Im}(\sigma) = \mathrm{Im}(f) \setminus Y_f = \mathrm{Im}(h \circ \pi) \setminus Y_f = \mathrm{Im}(h) \setminus Y_f$.

$\textsc{Decode}_n$ 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'.

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 SolveSim) with respect to only one h at a time.

The computation of the algorithm SolveSim (Algorithm 4) is similar to the computation of Solve (Algorithm 1). First, SolveSim computes the sets Y_i. There is one big difference between Solve and SolveSim. As 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 Solve algorithm.

- We use h whenever we need to determine the image of f_n for some n. As $\forall n \in \mathbb{N}$: $\mathrm{Im}(h_n) = \mathrm{Im}(f_n)$ using $\mathrm{Im}(h_n)$ instead of $\mathrm{Im}(f_n)$ makes no difference to the computation.
- The second place where h is used instead of f is when SolveSim computes the set Y_i. Specifically, when determining images y for which the security reduction R queries the given instance i, the algorithm SolveSim computes the same Y_i as if it used f by the f-obliviousness of the security reduction.
- In all other places, SolveSim uses the partial knowledge of f (i.e., the partial function f'). This causes a real difference in the computation. In particular, the set $S_{i,f'}$ (as computed by SolveSim) may differ a lot from $S_{i,f}$ (as computed by Solve) as some solutions from $S_{i,f}$ potentially query some unknown parts of f. Thus, the set $S_{i,f'}$ computed by SolveSim is just a subset of the whole $S_{i,f}$. The set $S_{i,f'}$ contains only the solutions SolveSim is "aware of" (f' is defined for all queries and thus SolveSim may verify the solution). The rest of the computation is practically the same, except that SolveSim is restricted just to the set of solutions $S_{i,f'}$. The main trick is that we make sure that 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. For ease of presentation and understanding, we divide the proof into four lemmata, Lemma 1 through 4. For lack of space, their proofs are provided in the full version [**?**]. Lemma 1 shows that given an instance i of the TFNP problem represented by the algorithm C^f, our 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 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 $|s| \leq p(|i|)$ and $C^f(i,s) = 1$.*

To argue that Solve does not help in inverting injective functions, we analyze the joint properties of the algorithms $\textsc{Encode}_n$ and $\textsc{Decode}_n$. First, we show

Algorithm 4: The algorithm SolveSim.

Hardwired : a fully black-box construction (R, T_R, C, T_C, p) of a hard TFNP problem from iOWF
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 $C^{f'}(i,s) = 1$

Compute $Y_i = \bigcup_{n=1}^{T_C(|i|+p(|i|))} \left\{ y \in \mathrm{Im}(h_n) \;\middle|\; i \in Q_{\textsc{Solve}}(R^{h,\textsc{Solve}}(1^n, y)) \right\}$
Compute $N_i = \{n \in \mathbb{N} \mid Y_i \cap \mathrm{Im}(h_n) \neq \emptyset\}$
for $n \in N_i$ **do**
  Compute $Y_{i,n} = Y_i \cap \mathrm{Im}(h_n)$
end
Compute
$$S_{i,f'} = \left\{ s \in \{0,1\}^* \;\middle|\; |s| \leq p(|i|) \text{ and } Q(C^{f'}(i,s)) \subseteq \mathrm{Dom}(f') \text{ and } C^{f'}(i,s) = 1 \right\}$$
while *True* **do**
  $B_{i,f'} = \{s \in S_{i,f'} \mid f[Q(C^{f'}(i,s))] \cap Y_i = \emptyset\}$ // "benign" solutions
  if $B_{i,f'} \neq \emptyset$ **then**
    return lexicographically smallest $s \in B_{i,f'}$
  end
  Choose $n \in N_i$ such that $\frac{|Y_{i,n}|}{2^n}$ is maximized.
  Set $N_i = N_i \setminus \{n\}$
  Set $Y_i = Y_i \setminus Y_{i,n}$
end

that $\textsc{Decode}_n$ always returns the correct permutation encoded by $\textsc{Encode}_n$ (see [?, Section 4.2] for the formal proof).

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

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

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

We crutialy rely on f-obliviousness of the reduction for the proof of Lemma 2. It is the property which allows us to simulate the algorithm Solve during the decoding phase, as SolveSim needs to be able to compute the same set Y_i as Solve does. Moreover, SolveSim cannot query f on all preimages as Solve does when computing Y_i. Due to f-obliviousness of the reduction, we may substitute f by h in the computation of Y_i in SolveSim as the resulting set depends only on the image of the function given to R as an oracle (and $\mathrm{Im}(f) = \mathrm{Im}(h)$).

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

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.,*

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

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. 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

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

We claim (see [?, Claim 10]) 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 all quasi-polynomial (and, hence, also for efficient) algorithms R, C. We use this fact again in the proof of the main theorem (Theorem 1), and refer the readers to [?, Section 4.4] for the precise statement and its proof.

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

Proof (of Theorem 1). Suppose to the contrary that there is such a reduction (R, T_R, C, T_C, p). By Lemma 1, the algorithm $\textsc{Solve}$ (Algorithm 1) returns a valid solution with probability one. Thus, the reduction R must invert f with high probability when given access to any oracle $f \in \mathrm{Inj}$ and our oracle $\textsc{Solve}$, 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. As already pointed out, the quasi-polynomial bounds on running times $T_C, T_R \in O(n^{\mathrm{polylog}(n)})$ imply that $q(n) \in o(2^{0.2n})$ (see [?, Claim 10]). 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 of algorithms $\textsc{Encode}_n^h$

and $\textsc{Decode}_n^h$ 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 on the expected length of any prefix-free encoding of a random permutation on $\{0,1\}^n$ given by Corollary 1. □

3.3 Extensions

In this section, we state some extensions to the result in the previous section. We refrain from providing the details and refer the readers to [?, Section 5].

First, 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 $\textsc{Solve}$ in parallel, though still f-obliviously, as defined in Definition 3. The description of the algorithms $\textsc{Solve}$, $\textsc{Encode}_n$, $\textsc{Decode}_n$, and $\textsc{SolveSim}$ remain unchanged, but we require a slightly different analysis tailored for non-adaptive reductions. We refer the readers to [?, Section 5.2] for the details.

Second, we can extend our results to randomised reductions with additional changes to our algorithm $\textsc{Solve}$. 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. Roughly speaking, the $\textsc{Solve}$ for randomised reductions is an extension of the $\textsc{Solve}$ given in Algorithm 1 taking this probability into account. $\textsc{SolveSim}$ is designed accordingly and thanks to f-obliviousness we are still able to show that the simulation is faithful. The rest of the changes involve modifying the existing argument taking into account the changes to $\textsc{Solve}$ and $\textsc{SolveSim}$. We refer the readers to [?, Section 5.3] for the details.

4 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 $\textsc{Solve}$ 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 [?], for which, despite the recent progress, there are only non-trivial proofs [?, ?, ?].

Acknowledgements

We wish to thank the anonymous reviewers for their comments which helped us to clarify the presentations of our results.