PPAD-Hardness via Iterated Squaring Modulo a Composite

Arka Rai Choudhuri* Pavel Hubáček† Chethan Kamath‡

Krzysztof Pietrzak§ Alon Rosen¶ Guy N. Rothblum‖

June 5, 2019

Abstract

We show that, relative to a random oracle, solving the End-of-Line problem (which is PPAD-complete) is no easier than computing the function

$$f(N, x, T) = x^{2^T} \bmod N,$$

where N is an n-bit RSA modulus, $x \in \mathbb{Z}_N^*$ and $T \in \mathbb{N}$. It was conjectured by Rivest, Shamir and Wagner, that, unless the factorization of N is known, the fastest algorithm for computing f consists of $\Omega(T)$ iterated squaring operations mod N. Under a milder assumption, namely that computing f takes $n^{\omega(1)}$ time for some (possibly exponentially) large T, our construction of End-of-Line cannot be solved in $\mathsf{poly}(n)$ time.

We prove our result by reducing f to (a variant of) the Sink-of-Verifiable-Line problem, which is known to imply PPAD (and in fact CLS) hardness. The main building block of our reduction is a recently discovered interactive public-coin proof by Pietrzak for certifying $y = f(N, x, T)$, which can be made non-interactive using (an analogue of) the Fiat-Shamir heuristic. The value y can be computed together with the proof in time $\mathsf{poly}(n) \cdot T$, and the proof can be verified in time $\mathsf{poly}(n) \cdot \log T$. The key technical challenge in our setting is to provide a means by which the solution y together with a proof can be computed in small incremental steps, while the correctness of each intermediate state of this computation can still be verified in time $\mathsf{poly}(n, \log T)$.

*Johns Hopkins University, Baltimore, USA. Email: achoud@cs.jhu.edu. Supported in part by a DARPA/ARL Safeware Grant W911NF-15-C-0213, and a subaward from NSF CNS-1414023.

†Charles University, Prague, Czech Republic. Email: hubacek@iuuk.mff.cuni.cz. Supported by the project 17-09142S of GA ČR, Charles University project UNCE/SCI/004, and Charles University project PRIMUS/17/SCI/9. This work was done under financial support of the Neuron Fund for the support of science.

‡IST Austria, Klosterneuburg, Austria. Email: ckamath@ist.ac.at. Supported by the European Research Council, ERC consolidator grant (682815-TOCNeT).

§IST Austria, Klosterneuburg, Austria. Email: pietrzak@ist.ac.at. Supported by the European Research Council, ERC consolidator grant (682815-TOCNeT).

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

‖Weizmann Institute of Science, Rehovot, Israel. Email: rothblum@alum.mit.edu. This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 819702).

Contents

1 Introduction

The complexity class PPAD, defined by Papadimitriou [35], consists of all total search problems that are polynomial-time reducible to the End-of-Line problem: given a source in a directed graph where every vertex has both in-degree and out-degree at most one, find a sink or another source. The End-of-Line problem can be solved in linear time when the graph is given explicitly, but there is no known algorithm solving it in polynomial time when the input is an implicit representation of the graph describing the successor and predecessor of every vertex.

The class PPAD became a subject of intensive study due to its relation to the problem Nash, of finding a Nash equilibrium in a normal-form game. Papadimitriou showed that Nash is reducible to End-of-Line, and thus belongs to PPAD. A reduction in the opposite direction was later established (even for bimatrix games) in a sequence of works by Daskalakis, Goldberg and Papadimitriou [14], and Chen, Deng and Teng [11].

Currently, no PPAD-complete problem is known to admit a sub-exponential-time worst-case algorithm. This, together with the increasingly large number of reductions amongst PPAD complete problems, supports the belief that they are not solvable in polynomial time.Still, even if we do believe that no PPAD complete problem is solvable in polynomial time in the worst-case, it is of great interest to rule out the possibility that these problems admit efficient heuristics that perform well on the average.

1.1 Cryptographic hardness in PPAD

A natural approach for arguing average-case PPAD hardness is to reduce from problems that originate from cryptography. Such an approach was already advocated in Papadimitriou's original paper [35], but up until recently, not much progress has been made in this direction. This has changed as a result of developments in the study of program obfuscation [3, 20].

As shown by Bitansky, Paneth and Rosen [4] (building on [1]), the task of breaking sub-exponentially secure *indistinguishability obfuscation* (iO) is reducible to solving the End-of-Line problem. This gave the first extrinsic evidence of PPAD hardness and provided a plausible method to sample potentially hard-on-average End-of-Line instances.

The Bitansky et al. technique was extended by Hubáček and Yogev [24], who established hardness in CLS, a subclass of PPAD, under the same assumptions. Both results were subsequently strengthened. First, by Garg, Pandey and Srinivasan [21], who reduced from breaking iO with polynomial (instead of sub-exponential) hardness (or alternatively compact public-key functional encryption) and one-way permutations. Second, by Komargodski and Segev [31], who reduced from breaking quasi-polynomially secure private-key functional encryption and sub-exponentially-secure injective one-way functions.

In one way or the other, all of the above assumptions are closely related to iO, whose attainability is not implausible but nevertheless still lies within the domain of speculation. Given that many candidate iO schemes have been broken, and that surviving ones are yet to undergo extensive evaluation by the cryptographic community, it is desirable to base PPAD hardness on alternative assumptions. A step towards this direction was undertaken in a recent work by Choudhuri et al. [12] who showed that PPAD-hardness can be based on the assumption that the Fiat-Shamir transform is (unambiguously) sound for the sumcheck protocol of Lund et al. [32].

The approach of basing average-case TFNP-hardness on relatively established cryptographic assumptions has also been successfully applied in the context of complexity classes other than PPAD. For instance, Papadimitriou [35] showed that one-way permutations imply average-case hardness in PPP, and Jeřábek [25] showed that the undirected version of End-of-Line, which is complete for the class PPA, is no easier than FACTORING. It is also known (folklore) that any assumption that implies the existence of collision-resistant hashing (e.g. hardness of Factoring, SIS or DLP) implies PWPP-hardness. It is currently not known whether any of these

results can be extended to PPAD.

1.2 Our Results

Hubáček and Yogev [24] showed that hardness of a structured promise problem called Sink-of-Verifiable-Line, implies hardness for CLS (and thus also PPAD). Choudhuri et al. [12] defined a closely related problem called relaxed-Sink-of-Verifiable-Line (rSVL), and adapted the [24] result so it also applies to rSVL (Lemma 1). Our main result is the construction of a hard-on average distribution of the rSVL problem based on the hardness of computing the function

$$f(N, x, T) = x^{2^T} \bmod N,$$

where N is the product of two random $n/2$-bit safe primes (where p is a safe prime if $(p-1)/2$ is also a prime), $x \in \mathbb{Z}_N^*$ and $T \in \mathbb{N}$. Computing f was suggested as a hard problem by Rivest, Shamir and Wagner (RSW) [37], who conjectured that for any T, computing $f(N, x, T)$ either requires $\Omega(T)$ *sequential* time or total computation sufficient to factor N.

Our hardness assumption is even milder, as it is sufficient for us that f cannot be computed in time $\mathsf{poly}(n)$ for some (potentially exponentially large) T (see Assumption 2 in §3.2). In other words, unlike RSW: (1) we do not assume any sequentiality, and (2) we allow an exponential gap between T and the total computation required to compute $x^{2^T} \bmod N$.

Our reduction also requires assuming access to a random function (in other words, it is relative to a *random oracle*), which is used in the context of transforming a $\log T$-round public-coin interactive proof into a non-interactive one, analogous to the Fiat-Shamir heuristic [18] (which applies to constant-round public-coin interactive proof systems).

Our main result is stated below. It is an informal restatement of the technical Theorem 1 from §5.3.

Theorem 1 (main, informal)**.** *For a security parameter $n \in \mathbb{N}$, let N be the product of two random $n/2$-bit safe primes and $x \in \mathbb{Z}_N^*$ be sampled uniformly at random. If there exists $T = T(n) \leq 2^n$ such that no $\mathsf{poly}(n)$-time algorithm, on input (N, x, T), can compute $x^{2^T} \bmod N$ except with negligible probability then, relative to a random oracle, there exists a family of hard rSVL instances.*

To summarize, this paper demonstrates new ways for sampling hard-on-average PPAD/CLS instances, based on assumptions of seemingly different nature than those required by prior work (i.e., number-theoretic, in contrast to ones related to obfuscation), and possibly opens up new paths for placing Factoring (or closely related problems such as breaking RSA) in PPAD.

1.3 Techniques and Ideas

The structure of our reduction is similar to that of Bitansky et al. [4] who showed how to use a cryptographic problem that is assumed to be hard in order to sample an instance of Sink-of-Verifiable-Line (SVL), a structured promise problem that is reducible to End-of-Line.

An instance of the Sink-of-Verifiable-Line problem consists of an implicit representation of a directed graph with 2^m vertices such that every vertex has out-degree one and it is possible to efficiently test whether a given vertex v lies i successive steps from vertex v_0. The goal is to find the vertex that lies L steps from v_0. The graph is implicitly represented by two $\mathsf{poly}(m)$-size circuits: a successor circuit S that assigns to every vertex v its successor $\mathsf{S}(v)$, and a verification circuit V that, given v and i, is promised to certify whether $v = \mathsf{S}^i(v_0)$.

Consider the following natural approach for reducing the computation of $f(N, x, T)$ to an instance of SVL with length parameter $L = T$ and vertices of size n (i.e. $m = n$). The graph's source v_0 is a random $x \in \mathbb{Z}_N^*$, and the successor circuit S is the squaring modulo N function, yielding the graph:

$$x \to x^2 \to x^{2^2} \to x^{2^3} \cdots \to x^{2^T} \ (\mathrm{mod} N). \tag{1}$$

Notice that, assuming $x^{2^T} \bmod N$ cannot be computed in time $\mathsf{poly}(n)$ for a sufficiently large T (which can be even exponential in n – see Assumption 2 in §3.2), it is hard for any polynomial-time algorithm to find the node that is T steps from the source x.

In order to complete the reduction to SVL, we need to provide an efficient V that certifies that a vertex $v = y$ is obtained by invoking S for i successive times on x. This is where Pietrzak's proof system for certifying $y = f(N, x, T)$ comes into play [36].

Pietrzak's proof system. Pietrzak's protocol allows a prover to convince a verifier that a tuple $(N, x, T = 2^t, y)$ satisfies the relation $y = x^{2^T} \bmod N$ using $t = \log T$ rounds of interaction. It does not require either prover or verifier to know the factorization of N.

The protocol is recursive in the time parameter T. In the first step, the prover sends the midpoint $\mu = x^{2^{T/2}} \bmod N$ as a commitment to the verifier. If

$$x^{2^{T/2}} = \mu \bmod N \quad \text{and} \quad \mu^{2^{T/2}} = y \bmod N$$

both hold, then so does the original claim. This reduces the task of proving a statement for parameter T to proving two statements for parameter $T/2$. Next, using a random challenge r, the verifier and prover merge these two statements into a single statement by computing $x' := x^r \cdot \mu \bmod N$ and $y' := \mu^r \cdot y \bmod N$ and setting $y' = x'^{2^{T/2}} \bmod N$ as the new statement.

One can show that if the statement (N, x, T, y) is wrong, then with overwhelming probability over the choice of r so is the new statement $(N, x', T/2, y')$. The procedure is repeated t times, halving the time parameter T each time, until we arrive at a claim for $T = 1$ at which point the verifier can efficiently check the correctness itself by performing a single squaring.

The protocol, being public-coin, can be made non-interactive using an analogue of the Fiat-Shamir transformation. For this, the verifier's messages (i.e., the r's) are computed by applying a hash function H to the prover's messages. The non-interactive proof on challenge (N, x, T) is of the form $(N, x, T, y, \mu_1, \ldots, \mu_t)$, and we denote it by $\pi^T_{x \to y}$.

We point out the following three crucial properties of the protocol where n, if you recall, denotes the size of N in binary representation:

> Property 1: Given (N, x, ℓ, y), computing $\pi^\ell_{x \to y}$ requires $\ell + \mathsf{poly}(n)$ multiplications in $\mathbb{Z}^*_N$ and $\mathsf{poly}(n)$ space (if one is not given y, an additional ℓ multiplication are used to first compute $y = x^{2^\ell}$, but we'll always be in a setting where either $\ell = 1$ or y is known).
>
> Property 2: The size of a proof $\pi^\ell_{x \to y}$ is $\mathsf{poly}(n, \log \ell)$ bits.
>
> Property 3: Given two proofs $\pi^\ell_{x \to y}, \pi^\ell_{y \to z}$ as "advice", computing the proof $\pi^{2\ell}_{x \to z}$ can be efficiently reduced to computing a proof $\pi^\ell_{x' \to y'}$.

The reduction. As mentioned above, our goal is to use Pietrzak's protocol in order to efficiently implement a verification circuit V that, given $(v = y, i)$ verifies that $y = x^{2^i} \bmod N$, i.e., that y indeed lies at the i-th position on the line described in (1). A first attempt would be to augment the vertex labels $x_i = x^{2^i} \bmod N$ in (1) with a corresponding proof, i.e., consider the line

$$\pi^0_{x_0 \to x_0} \to \pi^1_{x_0 \to x_1} \to \pi^2_{x_0 \to x_2} \to \cdots \to \pi^T_{x_0 \to x_T},$$

where the circuit V simply runs the efficient proof verification algorithm of Pietrzak's protocol.

This change renders the line efficiently verifiable. However, it is now not at all clear how to implement the successor circuit S efficiently. The labels now comprise of proofs, and S is consequently required to efficiently "update" a proof $\pi^i_{x_0 \to x_i}$ to $\pi^{i+1}_{x_0 \to x_{i+1}}$. To overcome this issue, we use the ability to "merge" proofs, in the sense that given proofs $\pi^\ell_{x \to y}, \pi^\ell_{y \to z}$ one can efficiently compute a single proof $\pi^{2\ell}_{x \to z}$.

Given the ability to merge proofs, we can construct a valid SVL instance by considering a line where going from the i-th vertex to the $i+1$-th vertex we augment the label (now consisting of multiple "partial" proofs) with a proof for the single step $\pi^1_{x_i \to x_{i+1}}$, and then merge the latest proofs as long as they are for the same time parameter (i.e., if the last two proofs are of the form $\pi^\ell_{a\to b}, \pi^\ell_{b\to c}$ merge them into $\pi^{2\ell}_{a\to c}$). This results in a line of the form

$$\pi^0_{x_0\to x_0} \to \pi^1_{x_0\to x_1} \to \pi^2_{x_0\to x_2} \to \pi^2_{x_0\to x_2}, \pi^1_{x_2\to x_3} \to \pi^4_{x_0\to x_4} \to \pi^4_{x_0\to x_4}, \pi^1_{x_4\to x_5} \to \cdots,$$

where crucially the number of proofs contained in each label always remains below $\log T$.

Strictly speaking, Pietrzak's proof system does not support efficient merging of proofs as outlined above. However, it does support somewhat efficient merging as in Property 3. Our key observation is that this somewhat-efficient merging is already sufficient to construct a valid SVL instance where *both* the successor circuit S and verification circuit V run in $\mathsf{poly}(n)$ time.

Suppose that we could construct an SVL instance where starting with a label x, after $L(\ell)$ invocations of S (for $L(\cdot)$ to be defined) we arrive at a label that contains a proof $\pi^\ell_{x\to y}$ establishing $y = x^{2^\ell} \bmod N$. Then we can get an SVL instance where starting with some label x we arrive at a proof $\pi^{2\ell}_{x\to z}$ making $3 \cdot L(\ell)$ invocations of S. The idea is to first compute $\pi^\ell_{x\to y}$ in $L(\ell)$ steps, then $\pi^\ell_{y\to z}$ in another $L(\ell)$ steps (while keeping the first proof $\pi^\ell_{x\to y}$ around in the label), and finally using another $L(\ell)$ steps to merge those two proofs into $\pi^{2\ell}_{x\to z}$ using Property 3. The recursive algorithm outlined above satisfies $L(2\ell) = 3 \cdot L(\ell)$ steps, and as $L(1) = 1$, solving this recursion we get $L(\ell) = \ell^{\log 3}$. Thus, $x^{2^T} \bmod N$ is reached after $L(T) = T^{\log 3}$ invocations of S.

One important detail that we have glossed over in the above description is that, given that the proof system does not have perfect soundness, we need to deal with the existence of accepting proofs for incorrect statements. We handle this, like in [12], by working with a relaxed variant of the Sink-of-Verifiable-Line problem which can also be reduced to End-of-Line and is thus sufficient for establishing average-case hardness in PPAD (in fact, even in CLS). We show in §5 how exactly the above ideas are used to sample instances of the relaxed-Sink-of-Verifiable-Line problem.

Comparison with [12]. The result in this paper can be viewed as an alternative instantiation of the ideas in [12] with are two main differences concerning the underlying assumptions. First, the hardness assumption in [12] — i.e., the (worst-case) hardness of #P — is weaker than the concrete number theoretic Assumption 2. However, secondly, the interactive protocol underlying our construction — i.e., Pietrzak's protocol [36] — has only logarithmic number of rounds compared to polynomial number of rounds in the sumcheck protocol [32], which forms the basis for [12]. Hence, it is potentially easier to instantiate (with a concrete hash function) the Fiat-Shamir transform for the construction in this paper than in [12]. Taken together, one could argue that the two results are in some sense incomparable. In addition, the construction in this paper is conceptually simpler and therefore could lead to simpler algorithms for sampling hard PPAD instances.

1.4 Open Problems

Our results and techniques motivate several natural research directions.

Removing the Random Oracle: Our construction of a hard-on-average PPAD distribution employs a random oracle. Specifically, we use it to obtain a non-interactive version of Pietrzak's proof for certifying that $y = f(N, x, T) = x^{2^T} \bmod N$. Given that 1) the algebraic statement that is being proved has a very specific structure and 2) Pietrzak's proof system has statistical soundness, it might be possible to design a non-interactive proof system for certifying $y = f(N, x, T)$ directly, i.e., without relying on any general

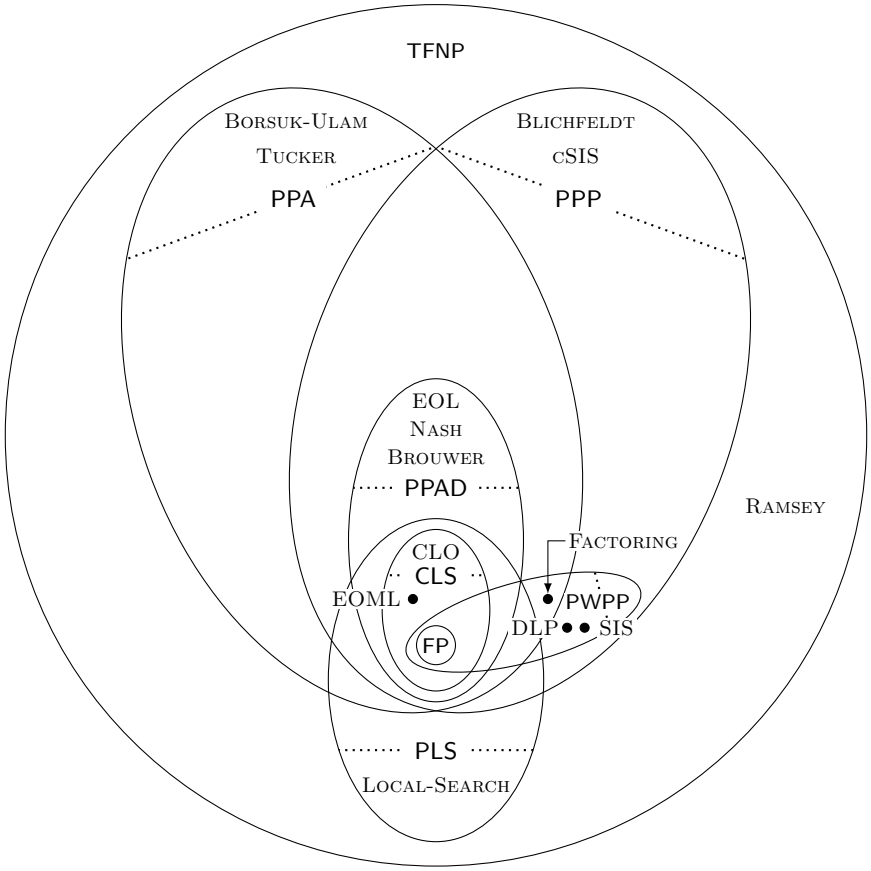

Figure 1: The TFNP landscape.

transformation such as Fiat-Shamir. Towards this goal, in the context of sumcheck proofs, the recent work of Choudhuri et al. [12] instantiate the hash function in the Fiat-Shamir transformation assuming the existence of optimally secure fully homomorphic encryption schemes against quasi-polynomial time adversaries.

Reducing from factoring: Hardness of integer factoring is necessary for our hardness assumption (Assumption 2) to hold. However, similarly to the RSW assumption or the RSA assumption, it is not clear if hardness of factoring is sufficient. It is natural to ask if our techniques can be improved to reduce from a weaker hardness assumption. Note that our reduction basically shows that the access to an rSVL oracle enables one to compute exponential powers modulo a composite efficiently. Can this ability be exploited either towards breaking the RSA assumption or for efficient integer factoring?

Exploiting somewhat-efficient merging of proofs: The crucial observation we make in this work is that the possibility to merge proofs somewhat-efficiently is sufficient for performing the computation of $f(N, x, T)$ in an incrementally verifiable manner. This was exploited to a certain extent in the recent work by Choudhuri et al. [12]. We expect this technique to find applications in different contexts.

1.5 Related Work

Systematic study of total search problems (i.e., with the guaranteed existence of a solution) was initiated by Megiddo and Papadimitriou [34], who defined a corresponding complexity class, called TFNP. They observed that unless NP = co-NP, a "semantic" class such as TFNP is unlikely to have complete problems. Motivated by this observation, Papadimitriou [35] defined

"syntactic" subclasses of TFNP with the goal of clustering search problems based on the various (non-constructive) existential theorems used to argue their totality (cf. Figure 1). Perhaps the best known such class is PPAD [35] which captures the computational complexity of finding Nash equilibria (NASH) in bimatrix games [14, 11], amongst other natural problems [29].

Other subclasses of TFNP include PPA [35], which captures computational problems related to the Borsuk-Ulam theorem (BORSUK-ULAM) or Tucker's lemma (TUCKER) [16], the class PLS [26] that was defined to capture the computational complexity of problems amenable to local search such as LOCAL-MAXCUT, and the class CLS [15], which captures finding approximate local optima of continuous functions (CLO) and contains finding Nash equilibria in congestion games or solving simple stochastic games of Condon or Shapley. Finally, the classes PPP [35] and PWPP [25] are motivated by the pigeonhole principle and contain important problems related to finding collisions in functions. Recently, Sotiraki, Zampetakis and Zirdelis [39] introduced a PPP-complete problem related to Blichfeldt's theorem in the theory of lattices (BLICHFELDT). Building on top of that, they showed that a constrained variant of the short integer solution problem (cSIS) is PPP-complete.

On the face of it, all TFNP problems could be potentially solvable in polynomial time without defying our understanding of the broader landscape of complexity theory (e.g. no surprising collapse of any important complexity classes seems to be implied by assuming TFNP $\subset$ FP). In light of this, it is natural to seek "extrinsic" evidence supporting TFNP hardness, for instance based on computational problems originating in cryptography. This approach would also have the added benefit of establishing average-case hardness, in some sense also indicating a certain level of resistance against heuristic algorithms.

Some of the works along these lines were already mentioned in §1.1 [4, 24, 21, 31] — here, we mention a few more. Hubáček, Naor and Yogev [23] recently constructed hard TFNP problems from one-way functions (in fact from any average-case hard NP language) under complexity theoretic assumptions used in the context of derandomization. Though, it is not known whether their distribution gives rise to average-case hardness in any of the syntactic subclasses of TFNP. Komargodski, Naor and Yogev [30] demonstrated a close connection between the Ramsey problem (RAMSEY) and the existence of collision-resistant hashing.

The relatively small progress on showing average-case hardness of total search problems from weak general assumptions motivated a line of works focusing on limits for proving average-case hardness. The implausibility of using worst-case NP hardness [26, 34] was later strengthened to show that it is unlikely to base average-case TFNP hardness even on problems in the polynomial hierarchy [8], and to show that any randomized reduction from a worst-case NP language to an average-case TFNP problem would imply that SAT is checkable [33]. A recent result [38] applies to the whole of TFNP and shows that any attempt for basing average-case TFNP hardness on (trapdoor) one-way functions in a black-box manner must result in instances with exponentially many solutions. This is in contrast to all known constructions of average-case hard PPAD problems that result in instances with small number of solutions.

Orthogonally to the above works, the smoothed complexity approach was recently used to identify natural distributions of PLS-complete problems (such as the LOCAL-MAXCUT or the problem of finding pure Nash equilibria in network coordination games) that admit polynomial time algorithms [2, 7].

Prior to our work, arguably the most natural average-case hard distribution of structured TFNP problems followed from the randomized reduction from FACTORING to PPA developed in the works of Buresh-Oppenheim and Jeřábek [9, 25].

Concurrent work. In a concurrent and independent work, Ephraim et al. [17] construct objects called "continuous verifable delay functions" and show how they can be used to construct SVL instances. Their construction is similar to ours except that they use a $k+1$-ary tree instead of a ternary tree in our construction. Appropriately setting the parameter k allows them to

relax the assumption to the soundness of (i) $\omega(1)$-round Fiat-Shamir transform to construct hard SVL instances, and (ii) constant-round Fiat-Shamir transform to separate $\mathsf{P} \cap \mathsf{PPAD}$ from NC. Our construction given in §5 can be thought of as their construction with k set to 2.

2 Definitions

In this section, we recall the basic definitions for total search problems and promise problems from previous work. In §2.3, we define our relaxed version of Sink-of-Verifiable-Line.

2.1 Complexity Classes and Total Search Problems

An efficiently-verifiable search problem is described via a pair $(\mathcal{L}, \mathcal{R})$, where $\mathcal{L} \subseteq \{0,1\}^*$ is an efficiently-recognizable set of instances, and $\mathcal{R}$ is an efficiently-computable binary relation — the class that contains all such problems is known as *functional* NP (FNP). Such a search problem is *total* if for every instance $v \in \mathcal{L}$ there exists a witness w of length $\mathsf{poly}(|v|)$ such that $\mathcal{R}(v, w) = 1$. The class *total* FNP (TFNP) consists of all efficiently-verifiable search problem that are total.

The class *polynomial parity argument over directed graphs* (PPAD) is a *syntactical* sub-class of TFNP which consists of all problems that are polynomial-time reducible to the End-of-Line problem (also known as the Source-or-Sink problem) [35].

Definition 1. An End-of-Line (EOL) instance (S, P) consists of a pair of circuits $\mathsf{S}, \mathsf{P} : \{0,1\}^m \to \{0,1\}^m$ such that $\mathsf{P}(0^m) = 0^m$ and $\mathsf{S}(0^m) \neq 0^m$. The goal is to find a vertex $v \in \{0,1\}^m$ such that $\mathsf{P}(\mathsf{S}(v)) \neq v$ or $\mathsf{S}(\mathsf{P}(v)) \neq v \neq 0^m$.

Intuitively, the circuits S and P can be viewed as implementing the successor and predecessor functions of a directed graph over $\{0,1\}^m$, where for each pair of vertices v and u there exists an edge from v to u if and only if $\mathsf{S}(v) = u$ and $\mathsf{P}(u) = v$ (note that the in-degree and out-degree of every vertex in this graph is at most one, and the in-degree of 0^m is 0). The goal is to find any vertex, other than 0^m, with either no incoming edge or no outgoing edge. Such a vertex must always exist by a parity argument.

The class *continuous local search* (CLS) lies in the intersection of PPAD and PLS and consists of all problems that are polynomial-time reducible to the Continuous-Local-Optimum problem (cf. [15] for the formal definition). Another problem that is known to lie in CLS (but not known to be complete) is End-of-Metered-Line [24]. (Below $[a]$ denotes $\{0, 1, \ldots, a\}$.)

Definition 2. An End-of-Metered-Line (EOML) instance $(\mathsf{S}, \mathsf{P}, \mathsf{M})$ consists of circuits $\mathsf{S}, \mathsf{P} \colon \{0,1\}^m \to \{0,1\}^m$ and $\mathsf{M} \colon \{0,1\}^m \to [2^m]$ such that $\mathsf{P}(0^m) = 0^m \neq \mathsf{S}(0^m)$ and $\mathsf{M}(0^m) = 1$. The goal is to find a vertex $v \in \{0,1\}^m$ satisfying one of the following:

(i) **End of line:** either $\mathsf{P}(\mathsf{S}(v)) \neq v$ or $\mathsf{S}(\mathsf{P}(v)) \neq v \neq 0^m$,

(ii) **False start:** $v \neq 0^m$ and $\mathsf{M}(v) = 1$,

(iii) **Miscount:** either $\mathsf{M}(v) > 0$ and $\mathsf{M}(\mathsf{S}(v)) - \mathsf{M}(v) \neq 1$ or $\mathsf{M}(v) > 1$ and $\mathsf{M}(v) - \mathsf{M}(\mathsf{P}(v)) \neq 1$.

The goal in EOML is the same as in EOL, but now the task is made easier as one is also given an "odometer" circuit M. On input a vertex v, this circuit M outputs the number of steps required to reach v from the source. Since the behaviour of M is not guaranteed syntactically, any vertex that attests a deviation in the correct behaviour of M also acts as a solution (and thus puts End-of-Metered-Line in TFNP).

2.2 The Sink-of-Verifiable-Line Problem

The Sink-of-Verifiable-Line problem is a *promise* search problem introduced by Abbot, Kane and Valiant [1] and further developed by [4]. It is defined as follows:

Definition 3. A Sink-of-Verifiable-Line (SVL) instance $(\mathsf{S}, \mathsf{V}, L, v_0)$ consists of $L \in [2^m]$, $v_0 \in \{0,1\}^m$, and circuits $\mathsf{S} : \{0,1\}^m \to \{0,1\}^m$ and $\mathsf{V} : \{0,1\}^m \times [2^m] \to \{0,1\}$ with the guarantee that for every $v \in \{0,1\}^m$ and $i \in [2^m]$, it holds that $\mathsf{V}(v,i) = 1$ if and only if $v = \mathsf{S}^i(v_0)$. The goal is to find a vertex $v \in \{0,1\}^m$ such that $\mathsf{V}(v, L) = 1$ (i.e., the sink).

Intuitively, the circuit S can be viewed as implementing the successor function of a directed graph over $\{0,1\}^m$ that consists of a single line starting at v_0. The circuit V enables to efficiently test whether a given vertex v is of distance i from v_0 on the line, and the goal is to find the vertex at distance L from v_0. Note that not every tuple $(\mathsf{S}, \mathsf{V}, L, v_0)$ is a *valid* SVL instance since V might not satisfy the promise about its behaviour. Moreover, there may not be an efficient algorithm for verifying whether a given tuple $(\mathsf{S}, \mathsf{V}, L, v_0)$ is a valid instance, hence this problem lies outside of TFNP.

Remark 1. The definition of SVL with an arbitrary source vertex v_0, as above, is equivalent to the definition in [4] where the source is 0^m. First, any SVL instance $(\mathsf{S}, \mathsf{V}, L)$ where the source is 0^m can be trivially transformed to an instance $(\mathsf{S}, \mathsf{V}, L, v_0 = 0^m)$. Second, we can reduce in the opposite direction by shifting the main line by v_0 as follows. Given an SVL instance $(\mathsf{S}, \mathsf{V}, L, v_0)$, define the new SVL instance as $(\mathsf{S}', \mathsf{V}', L)$ with source 0^m, where $\mathsf{S}'(v) := \mathsf{S}(v \oplus v_0)$ and $\mathsf{V}'(v) := \mathsf{V}(v \oplus v_0)$, where $\oplus$ denotes the bitwise XOR operation. Note that this general technique can be applied in the context of TFNP to any search problem where part of the instance is some significant vertex (e.g. the trivial source at 0^m in End-of-Line).

2.3 The Relaxed Sink-of-Verifiable-Line Problem

The formal definition of the *relaxed* Sink-of-Verifiable-Line problem from [12] is given in Definition 4. The main difference from Definition 3 is that the promise about the behaviour of the verifier circuit V is relaxed so that V can also accept vertices off the line starting at the vertex v_0. However, any vertex off the main line accepted by V is an additional solution.

Definition 4. A relaxed-Sink-of-Verifiable-Line (rSVL) instance $(\mathsf{S}, \mathsf{V}, L, v_0)$ consists of $L \in [2^m]$, $v_0 \in \{0,1\}^m$, and circuits $\mathsf{S} : \{0,1\}^m \to \{0,1\}^m$ and $\mathsf{V} : \{0,1\}^m \times [2^m] \to \{0,1\}$ with the guarantee that for every $(v,i) \in \{0,1\}^m \times [L]$ such that $v = \mathsf{S}^i(v_0)$, it holds that $\mathsf{V}(v,i) = 1$. The goal is to find one of the following:

(i) **The sink:** a vertex $v \in \{0,1\}^m$ such that $\mathsf{V}(v, L) = 1$; or

(ii) **False positive:** a pair $(v,i) \in \{0,1\}^m \times [L]$ such that $v \neq \mathsf{S}^i(v_0)$ and $\mathsf{V}(v,i) = 1$.

It is shown in [12] that, despite the relaxed promise, rSVL reduces to EOML and, thus, average-case hardness of rSVL is sufficient to imply average-case hardness of EOML.

Lemma 1. relaxed-Sink-of-Verifiable-Line *is many-one reducible to* End-of-Metered-Line.

3 Assumptions

We begin this section with the Rivest, Shamir and Wagner (RSW) time-lock puzzle and the hardness assumption that underlies its security (Assumption 1). Then, in §3.2, we describe the weaker assumption (Assumption 2) used in our variant of Pietrzak's protocol, to be presented in §4.

3.1 The RSW Time-Lock Puzzle

Rivest, Shamir and Wagner [37] introduced the notion of time-lock puzzles. Such a puzzle is specified by a sampling algorithm sample which, on input a security parameter n and a time parameter T, outputs a puzzle instance ι and the corresponding solution σ. The solution σ can be computed given only the puzzle ι making T simple sequential steps, using an algorithm solve. The security property requires that even an adversary with polynomially-many parallel cores cannot compute the solution much faster than the honest (sequential) algorithm.

They also propose a simple and elegant construction: on input (n, T), a puzzle is sampled by choosing two random n bit primes p, q (which then define an RSA modulus $N = pq$) together with any $x \in \mathbb{Z}_N^*$. The puzzle and solution are then defined as

$$\iota = (N, x, T) \quad , \quad \sigma = f(N, x, T) = x^{2^T} \bmod N.$$

The solution can be efficiently computed by the puzzle sampling algorithm in two steps using the knowledge of the group order $\phi(N) = (p-1)(q-1)$ as

$$e = 2^T \bmod \phi(N) \quad , \quad \sigma = x^e \bmod N. \tag{2}$$

It is conjectured in [37] that the fastest way to compute $x^{2^T} \bmod N$ is through repeated squaring:

$$x \to x^2 \to x^{2^2} \to x^{2^3} \to \ldots \to x^{2^T} \pmod N. \tag{3}$$

In particular, parallelism (beyond what can be used to speed up a single squaring) does not allow to compute the solution any faster.

Below we state this conjecture explicitly: we use "running time" to denote the total computation of an algorithm, while actual clock time of a computation is referred to by "sequential computation". For instance, if the algorithm is given as a circuit, then the running time would be its size, whereas the sequential computation is its depth.

Assumption 1 ([37]). *For a security parameter $n \in \mathbb{N}$, let N be the product of two random $n/2$-bit primes and $x \in \mathbb{Z}_N^*$ be sampled uniformly at random. For any $T \in \mathbb{N}$ and any algorithm* A *(whose running time is significantly smaller than what is required to factor N) that, on input (N, x, T), outputs $y = x^{2^T} \bmod N$ with overwhelming[1] probability the* sequential *computation performed by* A *is not much less than what is required to compute T sequential squarings in $\mathbb{Z}_N^*$.*

3.2 Our Number-Theoretic Assumption

The hardness result in this paper is based on a weaker assumption where we just require that for some superpolynomial T, $f(N, x, T)$ cannot be computed in polynomial time. The exact algebraic setting for our assumption, as formally stated in Assumption 2, differs slightly from that in Assumption 1. First, we require the primes p, q that define the modulus N to be *safe* primes (where, recall that, p is safe if $(p-1)/2$ is also prime). Second, the group that we assume our hardness is the group of signed quadratic residues (defined below), which is slightly more structured than $\mathbb{Z}_N^*$ – this extra structure helps enforce unique proofs for Pietrzak's protocol (cf. §4 for the details). However, we justify in Remarks 2 and 4 below that these changes do not really affect the strength of the assumption.

Signed quadratic residues. For two safe primes p and q, and $N := p \cdot q$ the signed quadratic residues [19, 22] is defined as the group

$$QR_N^+ := \{|x| \ : \ x \in QR_N\},$$

[1] Recall that a function $g : \mathbb{N} \to [0, 1]$ is negligible if for every polynomial $p(n) \in \mathsf{poly}(n)$ there's an n_0 s.t. $g(n) \leq p(n)$ for all $n \geq n_0$. g is overwhelming if $1 - g$ is negligible.

where $|x|$ is the absolute value when representing the elements of $\mathbb{Z}_N^*$ as $\{-(N-1)/2, \ldots, (N-1)/2\}$. Since $-1 \in \mathbb{Z}_N^*$ is a quadratic non-residue with Jacobi symbol $+1$, the map $|\cdot|$ acts as an (efficiently-computable) isomorphism[2] from QR_N to QR_N^+, and as a result QR_N^+ is also a cyclic group, with the group operation defined as

$$a \circ b := |a \cdot b \bmod N|.$$

However, unlike for QR_N, membership in QR_N^+ *can* be efficiently tested since $QR_N^+ = J_N^+$ where J_N is the group of elements with Jacobi symbol $+1$ and

$$J_N^+ := \{|x| \ : \ x \in J_N\} = J_N/\{\pm 1\}.$$

In other words, to test whether a given $x \in \mathbb{Z}_N^*$ (represented as $\{-(N-1)/2, \ldots, (N-1)/2\}$) belongs also to QR_N+, ensure that $x \geq 0$ *and* that its Jacobi symbol is $+1$.

The assumption. The hardness assumption that underlies the rSVL instance proposed in this paper is stated below.

Assumption 2. *For a security parameter n, let $N = p \cdot q$ be the product of two random $n/2$-bit* safe *primes p, q and $x \in QR_N^+$ be sampled uniformly at random. There exists some $T = n^{\omega(1)}, T \leq 2^n$, such that no $\mathsf{poly}(n)$-time algorithm, on input (N, x, T), can output $x^{2^T} \in QR_N^+$ except with non-negligible probability.*

Remark 2 (On using safe primes)**.** The primes p and q in Assumption 2 are *safe* primes to make sure that QR_N^+ contains no sub-group of small order – this property is exploited later to prove statistical soundness of the proof system in the next section.

It is conjectured that for some constant c, there are $c \cdot 2^n/n^2$ safe n-bit primes (cf. [41]), so a random n bit prime is safe with probability $\approx c/n$. Under the weaker requirement that there are at least $2^n/\mathsf{poly}(n)$ n-bit primes for some polynomial in $\mathsf{poly}(n)$, Assumption 1 is at least as strong as an assumption where we additionally require p and q to be safe, since if a $\Theta(1/\mathsf{poly}(n))$ fraction of all n-bit primes is safe, an N sampled as in Assumption 1 will be the product of two safe primes with noticeable probability $\Theta(1/\mathsf{poly}(n)^2)$.

Remark 3 (Using other groups and operations)**.** One can prove soundness of the protocol also when a standard RSA modulus is used (i.e., p and q are just random primes), or in fact any other group, but then one needs a computational assumption to argue soundness of the protocol, namely, that it is hard to find elements of small order [6]. So the hardness of our rSVL instance would rely on hardness of computing x^{2^T} as in Assumption 2, and additionally on the hardness of finding some element z where $z^e = x$ for some e of polynomial size.

We also note that instead of computing x^{2^T}, one can use the function x^{e^T} for any e for which x^e can be computed efficiently given x and e.

Remark 4 (On assumption in $(\mathbb{Z}_N^*, \cdot)$ vs. $(QR_N^+, \circ)$)**.** Although Assumption 2 concerns the hardness of exponentiation with respect to $(QR_N^+, \circ)$ (compared to Assumption 1 which applies to hardness of exponentiation modulo N), we explain below why restricting to QR_N^+ can only make the assumption milder. We argue in two steps using the assumption in QR_N as the intermediate step. To be specific, first we show that since QR_N is a subgroup of $\mathbb{Z}_N^*$ of sufficiently large size, Assumption 2 in QR_N is at least as strong (Step (i)); then, we exploit the isomorphism between QR_N and QR_N^+ to argue that if one breaks the assumption in QR_N^+ then one can break the assumption also in QR_N (Step (ii)).

[2]Note, however, that the inverse of this isomorphism is hard to compute exactly because of the quadratic residuosity assumption.

Step (i) As $|QR_N| = |\mathbb{Z}_N^*|/4$, a random element in $\mathbb{Z}_N^*$ also belongs to QR_N with probability $1/4$. Thus the reduction, on challenge $x \in \mathbb{Z}_N^*$, just invokes the algorithm A that breaks the assumption in QR_N on x, and is guaranteed to succeed at least a fourth of the time A succeeds.

Step (ii) Consider any $x \in QR_N$ and $y := x^{2^T} \bmod N$. By the properties of the isomorphism, the image of y in QR_N^+ is $y' = |x|^{2^T} \bmod N = x^{2^T} \in QR_N^+$. Thus given y' we know that $y \in \{y', N - y'\}$ is one of two possible values. Although the exact value cannot be computed (as it would contradict the quadratic residuosity assumption), we can guess one of the two values. Thus the assumption in $(QR_N^+, \circ)$ is as strong as the assumption in $(QR_N, \cdot)$.

Remark 5 (On the range of T)**.** Note that we allow T to be any superpolynomial value. Even though it seems that computing $x^{2^T} \bmod N$ only can get harder as T increases, we cannot actually prove this. Therefore, instead of just setting $T = 2^n$, we allow T to take any value to ensure Assumption 2 really is weaker than Assumption 1.

Let us observe that if T is the product of all n bit primes, then computing $x^{2^T} \bmod N$ is actually trivial as (using $T \bmod \phi(\phi(N)) = 0$ below)

$$x^{2^T} \bmod N = x^{2^T \bmod \phi(N)} \bmod N = x^{2^{T \bmod \phi(\phi(N))} \bmod \phi(N)} \bmod N = x \bmod N \ .$$

As this T is doubly exponentially large (while we require $T \leq 2^n$) this is not a valid choice, but this observation indicates why showing that computing x^{2^T} only gets harder as T increases might be tricky.

Before moving on the proof system, we point out some properties of the set of generators of the quadratic residues. This will prove useful later in establishing hardness of the rSVL instance proposed in §5 (Claim 1.2 in Theorem 1).

Generators of QR_N^+. Let's denote by $QR_N^\star \subset QR_N^+$ the set of generators of QR_N^+:

$$QR_N^\star = \{x \in QR_N^+ \ : \ \langle x \rangle = QR_N^+\} \ .$$

If $N = p \cdot q = (2p'+1)(2q'+1)$ is the product of $n/2$-bit safe primes, then we have

$$|QR_N| = |QR_N^+| = p' \cdot q' \qquad \text{and} \qquad |QR_N^\star| = (p'-1)(q'-1) = p' \cdot q' - p' - q' + 1 \ .$$

Our first observation is that a random element in QR_N^+ almost certainly also belongs to $QR_N^\star$:

$$\Pr_{x \leftarrow QR_N^+}[x \in QR_N^\star] = 1 - \frac{p' + q' - 1}{p' \cdot q'} \geq 1 - \frac{1}{2^{n/2-2}} \ . \tag{4}$$

Looking ahead, we will only be able to prove soundness of the protocol for statements (N, x, T, y) if $x \in QR_N^\star$. Although we can efficiently check if some x is in QR_N^+, we cannot efficiently check if it also belongs to $QR_N^\star$ (without knowing the factorization of N). But as a consequence of the above observations, an x chosen at random from QR_N^+ is almost certainly also in $QR_N^\star$.

Secondly, since the squaring map is an automorphism of QR_N^+[3] (and also QR_N) $x \in QR_N^\star$ implies $x^2 \in QR_N^\star$. As a result, starting with any $x \in QR_N^\star$, repeated squaring generates a subset of $QR_N^\star$: i.e., for any $x \in QR_N^\star$ we have

$$\{x, x^2, x^{2^2}, x^{2^3}, \ldots, x^{2^{(p'-1)(q'-1)-1}}\} \subseteq QR_N^\star \quad \text{with} \quad x^{2^{(p'-1)(q'-1)}} = x. \tag{5}$$

[3]Note that $(a \circ b)^2 = (a \circ b) \circ (a \circ b) = (a \circ a) \circ (b \circ b) = a^2 \circ b^2$, and since $\cdot^2$ is a permutation on QR_N^+ – or on QR_N as originially shown by Blum [5] – it is an automorphism.

4 Pietrzak's Proof System

The key component of our construction is Pietrzak's interactive protocol for showing that a tuple (N, x, T, y) satisfies $y = f(N, x, T) \bmod N$ [36]. There, the motivation was to construct a so-called "verifiable delay function" [6]. The protocol we use in this work differs in some minor aspects from the one in [36]. Most importantly, in order to enforce unique proofs (i.e., for every challenge (N, x, T), one cannot find more than one accepting proof), we switch the algebraic setting of the protocol from $\mathbb{Z}_N^*$ to QR_N^+, the group of signed quadratic residues we defined in §3.2. The other changes that we introduce make the proof system less efficient but enable a cleaner description — see Remark 6 for further details.

4.1 The Interactive Protocol

Our variant of Pietrzak's protocol allows a prover to convince a verifier that a tuple $(N, x, T = 2^t, y)$ satisfies the relation $y = x^{2^T}$ in the group QR_N^+ (i.e., here $x^2 := x \circ x$) using $\log T$ rounds of interaction. In the first step, the prover sends the midpoint $\mu = x^{2^{T/2}}$ as a commitment to the verifier. Note that if $x^{2^{T/2}} = \mu$ and $\mu^{2^{T/2}} = y$ are both true, so is the original claim $y = x^{2^T}$.

At this point we have reduced the task to prove a statement for time parameter T to proving two statements $(N, x, T/2, \mu)$ and $(N, \mu, T/2, y)$ for time parameter $T/2$. Next, the verifier merges these two statements into a single statement $(N, x', T/2, y')$ by computing a random linear combination: using the challenge r it computes $x' := x^r \circ \mu$ and $y' := \mu^r \circ y$.

One can show that with overwhelming probability over the choice of r the following holds: if the original statement was wrong, i.e., $x^{2^T} \neq y$, so will the new one, i.e., $x'^{2^{T/2}} \neq y'$. This basic protocol is repeated $\log T$ times, halving the time parameter every time, until we arrive at a claim for $T = 1$ at which point the verifier can efficiently check correctness without the help of the prover by making one squaring.

Removing interaction. The interactive proof system just outlined is public-coin and has an exponentially-small soundness error, which means we could make it non-interactive via the Fiat-Shamir transform [18]. However, Fiat-Shamir is known to be sound in the random-oracle model for *constant-round* interactive protocols, unlike the proof system above which involves $O(\log T)$ rounds of interaction.

Nonetheless, we show that a close analogue of the Fiat-Shamir transform does yield a non-interactive protocol that is sound in the random-oracle model. In particular, to remove interaction, we derive verifier's message (i.e., the r's) in each round by applying a hash function $H : \{0,1\}^* \to \{0,1\}^{3n}$ to the prover's message for that round — the range of H (thought of as $\mathbb{Z}_{2^{3n}}$) is chosen to be sufficiently large so that $H(\cdot) \bmod p'q'$, when H is modelled as a random oracle, is extremely close to uniform over $\mathbb{Z}_{p'q'}$. As a result, we get a non-interactive proof system (also with exponentially-small soundness error) for the statement "(N, x, T, y) satisfies $y = x^{2^T}$ in QR_N^+".

A number of recent works have aimed at relaxing the assumptions under which the Fiat-Shamir transformation can be proved sound [28, 27, 10]. In particular, [10] shows that for statistically-sound protocols, exponentially-hard key-derivation mechanism implies that Fiat-Shamir is sound. They also construct such KDMs under some strong assumptions related to learning with errors and the discrete-logarithm problem. Therefore, one could potentially argue the soundness above under much weaker assumptions than random oracles or indistinguishability obfuscation.

Remark 6 (On differences from [36])**.** In addition to the switch from $\mathbb{Z}_N^*$ to QR_N^+, we have introduced a few changes to make the proof system simpler at the cost of efficiency. We are able to employ these changes, listed below, as concrete efficiency is not the focus of our work — i.e., we only require the circuits S and V in our rSVL instance to be of polynomial size.

1. The range of the hash function H is much larger in this work than in [36]. This allows us to argue that $x^{H(\cdot)} \bmod N$ is close to uniform.

2. The prover described above uses the minimum space necessary, even though additional space can significantly improve the efficiency of the computation of the proof.

3. It suffices for us to consider a time parameter of the form $T = 2^t$ — the original protocol is described for arbitrary T.

4. We iterate the protocol until the parameter $T = 2^t$, which is halved in every round, is down to 1, even though it is more efficient to stop at an earlier round. In other words, the *base proof* in our case is of time parameter 1, whereas it was greater than that in [36].

4.2 The Non-Interactive Protocol

We describe here in full detail the non-interactive protocol (prove,verify) that results from the discussion in the previous section. (prove,verify) will serve as the basis for our main construction of RELAXED-SINK-OF-VERIFIABLE-LINE in §5. Also described is the simple algorithm solve that computes the solution (and is identical to the solver of the RSW time-lock puzzle).

- *Computing the solution,* solve. The algorithm solve on input $(N, T, x) \in \mathbb{Z} \times QR_N^+ \times \mathbb{Z}$ computes and outputs x^{2^T} by sequentially squaring x T times. Note that the first element $N \in \mathbb{Z}$ of the input defines the domain QR_N^+ of subsequent elements.

- *Computing the proof,* prove. The algorithm prove on input $(N, x, T = 2^t, y) \in \mathbb{Z} \times QR_N^+ \times \mathbb{Z} \times QR_N^+$ computes a proof for the claim $x^{2^T} = y$ as follows. Let $(x_1, y_1) = (x, y)$ and for $i = 1 \dots t$ recursively compute:

$$\begin{aligned} \mu_i &:= x_i^{2^{T/2^i}} \qquad \in QR_N^+ \\ r_i &:= H(\mu_i, x_i, y_i, T/2^{i-1}) \\ x_{i+1} &:= x_i^{r_i} \circ \mu_i \\ y_{i+1} &:= \mu_i^{r_i} \circ y_i. \end{aligned}$$

  Then $\mathsf{prove}(N, x, T, y)$ outputs the proof

$$\pi_{x \to y}^T = (N, T, x, y, \mu_1, \dots, \mu_t) = (N, T, x, y, \boldsymbol{\mu}) \quad \in \mathbb{Z}^2 \times {QR_N^+}^{t+2}. \tag{6}$$

- *Verifying a proof,* verify. The verification algorithm verify on input $\tilde{\pi}_{x \to y}^T = (N, T, x, y, \mu_1, \dots, \mu_t)$ first checks that x, y and all μ_i are in QR_N^+ — if this check fails $\mathsf{verify}(\tilde{\pi}_{x \to y}^T)$ outputs 0. Otherwise let $(x_1, y_1) = (x, y)$ and then for $i = 1 \dots t$ compute:

$$\begin{aligned} r_i &:= H(\mu_i, x_i, y_i, T/2^{i-1}) \\ x_{i+1} &:= x_i^{r_i} \circ \mu_i \\ y_{i+1} &:= \mu_i^{r_i} \circ y_i. \end{aligned}$$

  The output of $\mathsf{verify}(\tilde{\pi}_{x \to y}^T)$ is 1 if $x_{t+1}^2 = y_{t+1}$ and 0 otherwise.

We end the description of (prove,verify) with a comment on notation that will be used in this paper: we reserve

$$\pi_{x \to y}^T := \mathsf{prove}(N, x, T, x^{2^T})$$

to denote honestly computed proofs for true statements, and $\tilde{\pi}_{x \to y}^T$ for any string that parses as a possible proof, i.e., starts with N, x, T, y and is in $\mathbb{Z}^2 \times {QR_N^+}^{t+2}$.

4.3 Soundness

The soundness of the proof system (prove,verify) can be shown in the random-oracle model assuming that an adversary never finds a "bad query" as defined in Definition 5. To be precise, we first argue that these bad queries are hard to find provided that the adversary is allowed bounded number of queries to the random oracle (Lemma 2); conditioned on the adversary not making a bad query, we prove that soundness is hard to break (Lemma 3).

Definition 5 (Bad query)**.** A query is a tuple (μ, x, y, T) where $\mu, x, y \in QR_N^+$ and $T \in \mathbb{Z}$. Let

$$r := H(\mu, x, y, T) \quad , \quad x' := x^r \circ \mu \quad \text{and} \quad y' := \mu^r \circ y.$$

We say the query (μ, x, y, T) is *bad* if $x \in QR_N^\star$ and moreover either

(i) $x' \notin QR_N^\star$; or

(ii) $\left(x^{2^T} \neq y \text{ or } \mu \neq x^{2^{T/2}}\right)$ and $x'^{2^{T/2}} = y'$.

Lemma 2 (Bad queries are hard to find)**.** *For any $N = p \cdot q$ where $p = 2p' + 1, q = 2q' + 1$ are $(n+1)$-bit safe primes, the following holds: any adversary that makes at most Q queries to the random oracle H will make a bad query with probability at most*

$$\frac{3 \cdot Q}{2^n}$$

The proof of Lemma 2 is an adaptation of [36, Lemma 1] to the setting of QR_N^+, and therefore is given in Appendix A. As a corollary of Lemma 2 we get a strong soundness guarantee for the proof system. It not only states that it is hard to find proofs for wrong statements, but it is even hard to find any accepting proofs that differ from honestly generated proofs for true statements.

Lemma 3 (Soundness)**.** *For any $N = p \cdot q$ where $p = 2p' + 1, q = 2q' + 1$ are $(n/2)$-bit safe, no adversary that makes at most Q queries to the random oracle H (but is otherwise computationally unbounded) will find a proof $\tilde{\pi}^T_{x \to y}$ where*

- *$x \in QR_N^\star$ (we let the adversary choose T and x, but require x to be in $QR_N^\star$).*
- *$\mathsf{verify}(\tilde{\pi}^T_{x \to y}) = 1$ (proof verifies)*
- *$\pi^T_{x \to x^{2^T}} \neq \tilde{\pi}^T_{x \to y}$ (proof is different from an honestly generated proof for a true statement)*

except with probability $\leq 3 \cdot Q / 2^{n/2-1}$.

Proof. Let "break" denote the event that an adversary that makes at most Q queries to the random oracle finds a proof $\tilde{\pi}^T_{x \to y}$ such that $\mathsf{verify}(\tilde{\pi}^T_{x \to y}) = 1$ and $\pi^T_{x \to x^{2^T}} \neq \tilde{\pi}^T_{x \to y}$. The probability of "break" can be bounded as follows:

$$\begin{aligned}
\Pr[\text{break}] &= \Pr[\text{break} \wedge (\text{bad query} \vee \neg\text{bad query})] \\
&\leq \Pr[\text{break} \wedge \text{bad query}] + \Pr[\text{break} \wedge \neg\text{bad query})] \\
&= \Pr[\text{break} | \text{bad query}] \cdot \Pr[\text{bad query}] \leq 3 \cdot Q / 2^{n/2-1}
\end{aligned}$$

Note that $\Pr[\text{break} \wedge \neg\text{bad query})] = 0$ since if no bad queries were made then the proof $\tilde{\pi}^T_{x \to y}$ must equal $\pi^T_{x \to x^{2^T}}$. □

4.4 Efficiency

Finally, we point out three properties concerning the efficiency of (prove,verify). In particular Property 3, which allows for a somewhat-efficient merging of two proofs, will be absolutely crucial in our construction of hard rSVL instance that follows next in §5.

Property 1 (Cost of computing solutions and proofs)**.** The computational cost incurred to compute $x^{2^T} := \mathsf{solve}(N, x, T)$ is $T + \mathsf{poly}(n)$ multiplications in $(QR_N, \circ)$. The computational cost incurred to compute $\pi^T_{x\to y} := \mathsf{prove}(N, x, T, y)$ is also $T + \mathsf{poly}(n)$ multiplications in $(QR_N, \circ)$. The space required is $\mathsf{poly}(n)$ in both cases.

To see this, note that the cost of computing $\mathsf{prove}(N, x, T, y)$ is dominated by computing the μ_i's, which requires $T/2$ squarings for μ_1, $T/4$ for μ_2 etc., for a total of $T - 1$ squarings.

Property 2 (Size of the proof)**.** The size of a proof $\pi^T_{x\to y}$ is $O(n \cdot \log T)$ bits.

Property 3 (Cost of merging proofs)**.** Given two proofs $\pi^T_{x\to y}, \pi^T_{y\to z}$ as "advice", computing the proof $\pi^{2T}_{x\to z}$ can be efficiently reduced to computing a proof $\pi^T_{x'\to y'}$.

This property emerges from the recursive nature of the protocol: we can completely avoid computing the μ_1 component in the proof $\pi^{2T}_{x\to z} = (N, 2T, x, z, \mu_1, \mu_2, \ldots, \mu_{t+1})$ since it is already present in $\pi^T_{x\to y}$ in the form of the element y (i.e., $\mu_1 = x^{2^T} = y$). That is, to compute $\pi^{2T}_{x\to z}$ given $\pi^T_{x\to y}$ and $\pi^T_{y\to z}$, we first compute the merged statement

$$r := H(y = \mu_1, x, z, 2 \cdot T) \quad , \quad x' := x^r \circ y \quad , \quad y' := y^r \circ z \tag{7}$$

and then, making $T + \mathsf{poly}(n)$ multiplications, compute its proof

$$\pi^T_{x'\to y'} = (N, T, x', y', \mu'_1, \ldots, \mu'_t) := \mathsf{prove}(N, x', T, y').$$

From the proof for the merged statement, we can reconstruct the proof for $\pi^{2T}_{x\to z}$ as

$$(N, 2T, x, z, \mu_1, \mu'_1, \ldots, \mu'_t). \tag{8}$$

5 The Reduction

To construct a hard RELAXED-SINK-OF-VERIFIABLE-LINE instance, we rely on the hardness of computing x^{2^T} in the group QR^+_N as stated in Assumption 2. In particular, we aim to construct an efficient successor circuit S such that applying it iteratively to the initial state $(x, \ldots)$ we reach a (final) state $(x^{2^T}, \ldots)$. Meanwhile, every intermediate state can be efficiently certified to lie on the line using the verifier V — in order to construct such a V, we intend to use Pietrzak's proof system for certifying $y = x^{2^T}$ just described in §4.

We sketch in §5.1 why some simple approaches do not work. The reader, however, can skip these and directly jump to §5.2 where we discuss the solution using Property 3. But first, we fix some notation that will be used throughout this section.

Notation. Let Σ be an alphabet (we will use the binary $\{0, 1\}$ and ternary $\{0, 1, 2\}$ alphabets). Σ^t denotes the set of all strings of length t over Σ; $\Sigma^{\le t}$ denotes $\cup_{j\in[t]}\Sigma^j$ (with the empty string denoted by ε). For a string $a \in \Sigma^t$ and $j \in [t]$, $a[j]$ refers to the j-th symbol in a. For two strings a and b, ab represents their concatenation.

We address each node of a complete binary tree of depth t using the binary string that encodes its position — i.e., a node at level $l \in [t]$ is encoded by an l-bit string and, e.g., the root is ε, its children 0, 1 and so on. An analogous system is used for the complete trinary tree (cf. Figure 3).

Finally, we reserve "nodes" to refer only to the vertices of a tree, to avoid confusion with the vertices of the rSVL instance.

5.1 Intuition

We consider SVL or rSVL instances where (N, x, T) is first sampled as in Assumption 2.

Inefficient verifier circuit. The first idea is to sample an SVL instance as $(\mathsf{S}, \mathsf{V}, x, T)$ where

$$\mathsf{V}((y,i),j) = 1 \iff j = i \le T \text{ and } y = x^{2^i}$$

$$\mathsf{S}((y,i)) = \begin{cases} (y,i) & \text{if } i \ge T \\ (y^2, i+1) & \text{otherwise} \end{cases}$$

The only way to solve this instance is to find the end of the line (x^{2^T}, T), which under Assumption 2 is hard. Unfortunately without knowing the group order $\phi(N)$ we can't realize V efficiently as computing x^{2^i} requires i squarings.

Inefficient successor circuit. To allow efficient verification, we can replace the state (x^{2^i}, i) with a proof $\pi^i_{x\to y}$ establishing that $x^{2^i} = y$. That is, for $x_i := x^{2^i}$, we consider an rSVL instance $(\mathsf{S}, \mathsf{V}, \pi^0_{x_0\to x_0}, T)$ where S and V are defined as

$$\mathsf{V}(\tilde\pi^i_{x_0\to y}, j) = 1 \iff j = i \le T \text{ and } \mathsf{verify}(\tilde\pi^i_{x_0\to y}) = 1$$

$$\mathsf{S}(\tilde\pi^i_{x_0\to y}) = \begin{cases} \tilde\pi^i_{x_0\to y} & \text{if } \mathsf{V}(\tilde\pi^i_{x_0\to y}, i) = 0 \text{ or } i = T \\ \pi^{i+1}_{x_0\to x_{i+1}} & \text{else if } \tilde\pi^i_{x_0\to y} = \pi^i_{x_0\to x_i} \\ \text{unspecified} & \text{otherwise.} \end{cases}$$

Using soundness as stated in Lemma 3, we can argue that an adversary making a $\mathsf{poly}(n)$ number of oracle queries will not be able to find a wrong accepting proof $\tilde\pi^i_{x_0\to y}$, i.e.,

$$\tilde\pi^i_{x_0\to y} \neq \pi^i_{x_0\to x_i} \quad \wedge \quad \mathsf{verify}(\tilde\pi^i_{x_0\to y}) = 1$$

happens only with exponentially-small probability. Assuming the adversary does not find such a wrong proof, the only other way to solve the instance is by finding the correct sink $\mathsf{S}^T(\pi^0_{x_0\to x_0}) = \pi^T_{x_0\to x_T}$ (i.e., a solution of type (i) as per Definition 4), which under Assumption 2 is hard.

Unfortunately now it's not clear how to implement the successor circuit S efficiently, as computing a proof $\pi^{i+1}_{x_0\to x_{i+1}}$ seems to require around i exponentiations even when given a proof for the previous state $\pi^i_{x_0\to x_i}$.[4]

A solution assuming efficient merge. Assume it is possible, say using an algorithm merge, to merge proofs $\pi^\ell_{x\to y}$ and $\pi^\ell_{y\to z}$ into a single proof $\pi^{2\ell}_{x\to z}$ in just $\mathsf{poly}(n)$ steps (rather than $\ell + \mathsf{poly}(n)$ steps required by Property 3). This allows us to define a very simple hard rSVL instance using the *recursive* approach of Valiant [40]: reduce the computation of a proof for time parameter 2ℓ to the computation of two proofs for time parameter ℓ, and then use merge. The resulting algorithm f is given in Algorithm 1.

The description of the successor and verifier circuits for the corresponding rSVL instance can now be obtained by simulating $\mathsf{f}(N, x, T)$ using iterations and stack traces. We will see in

[4] As a way around the above problem, instead of assuming that S outputs the proof $\pi^{i+1}_{x_0\to x_{i+1}}$ in one invocation, we can split this computation into i efficient steps. However, we again run into the problem of implementing V efficiently, as – when computing this proof in a straight forward manner – we have no efficient way of verifying that the intermediate states are correct. If we just let V output 1 on states where it can't verify correctness, we will introduce "uninteresting" accepting states that neither contain $x_0^{2^T}$ nor break the soundness of the protocol (and thus we can't conclude that solving this instance breaks Assumption 2 or soundness).

$\mathsf{f}(N, x, \ell)$

1: **if** $x \notin QR_N^+$ or ℓ is not a power of 2 **then**
2: return $\perp$ ▷ Invalid input
3: **end if**
4: **if** $\ell = 1$ **then** ▷ Base case
5: **return** $\pi^1_{x \to x^2} := \mathsf{prove}(N, x, 1, x^2)$
6: **else**
7: $\pi^{\ell/2}_{x \to y} := \mathsf{f}(N, x, \ell/2)$ ▷ First recursive call
8: $\pi^{\ell/2}_{y \to z} := \mathsf{f}(N, y, \ell/2)$ ▷ Second recursive call
9: **return** $\pi^{\ell}_{x \to z} := \mathsf{merge}(N, \pi^{\ell/2}_{x \to y}, \pi^{\ell/2}_{y \to z})$ ▷ Do the efficient merge
10: **end if**

Algorithm 1: Recursive description of the rSVL instance with efficient merge.

§5.2 how this can be exactly (and succinctly) accomplished using the tree that captures the execution of f (see Figure 2.(a)), and limit below to an informal overview.

Let $x_i := x^{2^i}$ as before. Starting at $\emptyset$ and ending at $\pi^T_{x \to x_T}$, the rSVL instance has a main line of length T. The first few vertices on this line are:

$$\emptyset \to \{\pi^1_{x_0 \to x_1}\} \to \{\pi^2_{x_0 \to x_2}\} \to \{\pi^2_{x_0 \to x_2}, \pi^1_{x_2 \to x_3}\} \to \{\pi^4_{x_0 \to x_4}\} \to \{\pi^4_{x_0 \to x_4}, \pi^1_{x_4 \to x_5}\} \to \{\pi^4_{x_0 \to x_4}, \pi^2_{x_4 \to x_6}\} \to \cdots \to \{\pi^T_{x \to x_T}\}.$$

The verifier V simply checks if the input label corresponds to a valid sequence of accepting proofs, which can be done efficiently. The successor S — given that an input label is a valid sequence — looks at the last proof in this sequence, denoted $\pi^{\ell}_{a \to b}$, adds the base proof $\pi^1_{b \to b^2}$ to the sequence and then keeps merging the last two proofs in this sequence as long as they have the same time parameter. For example, the third label above is obtained by first adding the base proof $\pi^1_{x_1 \to x_2}$ to the previous label $\{\pi^1_{x_0 \to x_1}\}$ and then doing the merge; the fifth label is obtained by adding $\pi^1_{x_3 \to x_4}$ to $\{\pi^2_{x_0 \to x_2}, \pi^1_{x_2 \to x_3}\}$ and then merging *twice*.

Note that because of the merging, the number of proofs in the labels is guaranteed to stay below $\log T$ — the size of the input labels is therefore $\mathsf{poly}(n, \log T)$. Since $\pi^T_{x \to x_T}$ contains the value of $x_T = x^{2^T}$, finding the sink $\pi^T_{x \to x_T}$ is hard under Assumption 2. Moreover, coming up with a state that passes verification but is not of the form $\mathsf{S}^i(\emptyset)$ for some i requires breaking soundness of the underlying proof system.

The solution using somewhat-efficient merge. Unfortunately, as noted in Property 3, Pietrzak's proof system only allows for somewhat-efficient merging: two proofs $\pi^{\ell}_{x \to y}, \pi^{\ell}_{y \to z}$ can be merged into $\pi^{2\ell}_{x \to z}$ at the cost of computing a proof $\pi^{\ell}_{x' \to y'}$ (which still requires $i + \mathsf{poly}(n)$ multiplications). However, as we will explain in the next section, this property already suffices for a reduction.

5.2 The Reduction

We start below with a recursive formulation of the solution using somewhat-efficient merge as it is intuitive and easy to understand (and is an extension of the approach using Algorithm 1 we employed when merging is efficient). The description of the successor and verifier circuits is later obtained by using the standard trick of simulating a recursive algorithm using iterations and stack traces.

A recursive prelude. The main idea behind our construction is to merge the proofs recursively, exploiting Property 3: given $\pi^{\ell}_{x \to y}$ and $\pi^{\ell}_{y \to z}$, we efficiently reduce the computation of

$\pi^{2\ell}_{x\to z}$ to the computation of the proof $\pi^{\ell}_{x'\to y'}$ for x', y' as in the merged statement given in eq.(7). Thus, the computation of a proof for time parameter 2ℓ is reduced to the computation of 3 proofs for time parameter ℓ (unlike 2 proofs for the case merging is efficient), which then can be reduced to computing $3 \cdot 3$ proofs of time parameter $\ell/2$, and so on and so forth until $\ell = 1$ at which point we can efficiently compute the *base proof*.

The resulting recursive algorithm f is given in Algorithm 2. Note that the way f is structured, the whole computation is being carried out in a verifiable manner: the midpoint and the endpoint are both accompanied by proofs that they are the correct power of x and *only* certified values are being used in the subsequent calls.[5]

$\mathsf{f}(N, x, \ell)$

if $x \notin QR^{+}_{N}$ or ℓ is not a power of 2 **then**
  return $\perp$ ▷ Invalid input
end if
if $\ell = 1$ **then** ▷ Base case
  return $\pi^{1}_{x\to x^2} := \mathsf{prove}(N, x, 1, x^2)$
else
  $\pi^{\ell/2}_{x\to y} := \mathsf{f}(N, x, \ell/2)$ ▷ First recursive call
  $\pi^{\ell/2}_{y\to z} := \mathsf{f}(N, y, \ell/2)$ ▷ Second recursive call
  $\mu := y$, $r := H(\mu, x, z, \ell/2)$ and $x' := x^r \circ \mu$ ▷ Reduce finding $\pi^{\ell}_{x\to z}$ to finding $\pi^{\ell/2}_{x'\to y'}$
  $\pi^{\ell/2}_{x'\to y'} := \mathsf{f}(N, x', \ell/2)$ ▷ Third recursive call
  Parse $\pi^{\ell/2}_{x'\to y'}$ as $(\ell/2, x', y', \mu'_1, \ldots, \mu'_{\log(\ell/2)})$
  return $\pi^{\ell}_{x\to z} := (\ell, x, z, \mu, \mu'_1, \ldots, \mu'_{\log(\ell/2)})$ ▷ Reconstruct $\pi^{\ell}_{x\to z}$
end if

Algorithm 2: Recursive description of the rSVL instance.

The rSVL instance. As already pointed out, we obtain the successor and verifier circuits for the rSVL instance by simulating f using iterations and stack traces. To this end, we view the execution of $\mathsf{f}(N, x, T)$ as a complete *ternary* tree τ of depth $t = \log T$, where each node represents a call to f. In particular, a node $i \in \{0, 1, 2\}^{\leq t}$ in τ is labelled by the proof π_i that is computed using that particular call to f. The children of a particular node are, therefore, labelled by the three proofs that result from recursive calls made within.

To be precise, the root of τ is labelled $\pi^{T}_{x\to x_T}$ (where, if you recall, $x_i := x^{2^i}$), its three children

$$\pi^{T/2}_{x\to x_{T/2}}, \pi^{T/2}_{x_{T/2}\to x_T} \text{ and } \pi^{T/2}_{x'\to x'^{2^{T/2}}},$$

where x' is computed as in eq.7, and so on until the leaves which are labelled using base proofs. For example, the tree corresponding to $\mathsf{f}(N, x, 4)$ is depicted in Figure 2.(b).

The rSVL instance we propose consists of a main line of length $3^{\log T}$ starting at $\emptyset$ and ending at $\pi^{T}_{x\to x_T}$. The intermediate vertices can be described using τ thanks to a one-to-one correspondence: for $i \in \{0, 1, 2\}^t$, the i-th vertex on the main line consists of set the of proofs in the stack of $\mathsf{f}(N, x, T)$ when its execution begins the recursion at i — *the stack trace at* i, for short.

[5] As noted in Section 5.1 this is reminiscent of the approach used by Valiant [40] to construct incrementally verifiable computation from computationally-sound proofs of knowledge (especially the idea of merging proofs, using stack traces for incremental computation etc.). However, Valiant works in a setting where the prover is efficient and as a consequence, merging proofs also turns out to be efficient, and thus his approach is closer to the solution using efficient merge described in §5.1. Our main observation is that even in some cases where the merging is not efficient, his ideas might still apply.

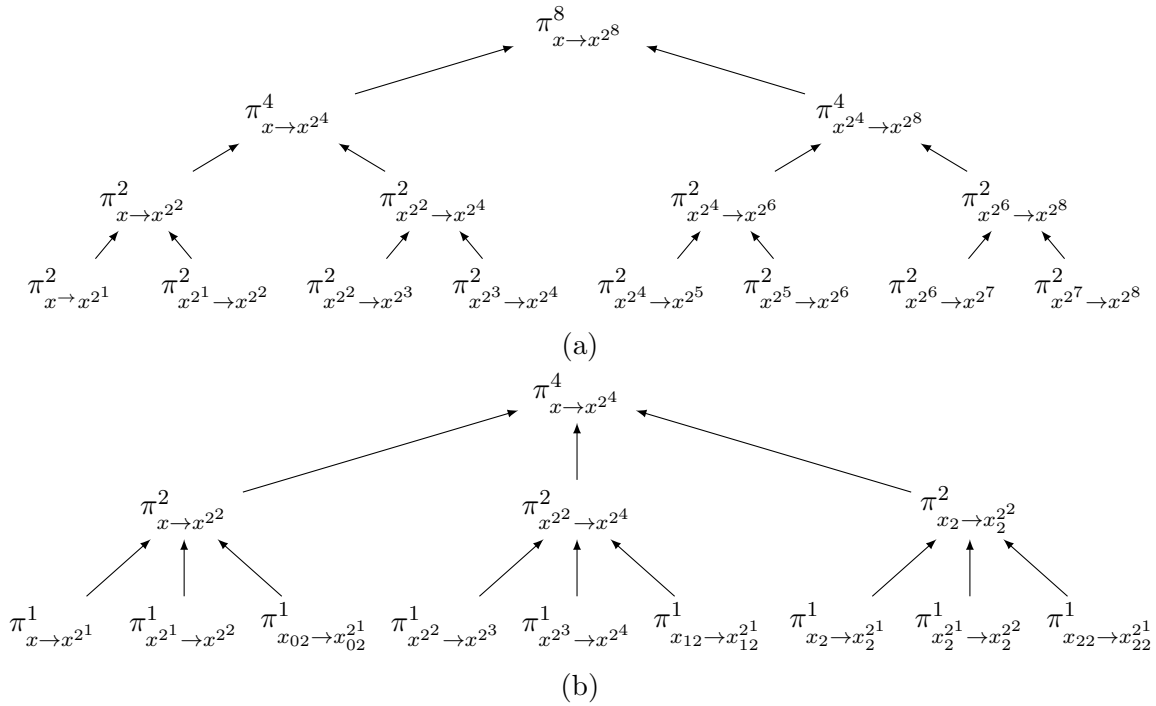

Figure 2: (a) The complete binary tree that corresponds to $\mathsf{f}(N, x, 8)$ from Algorithm 1. (b) The complete ternary tree that corresponds to $\mathsf{f}(N, x, 4)$ from Algorithm 2. The value of the x_2, x_{02}, x_{12} and x_{22} can be computed using eq.7.

These proofs can be described in terms of τ, but we have to first recall certain definitions pertaining to trees. The sibling of a node i in a tree is defined as the set of nodes that have the same parent as i. By "left" siblings of a node $i \in \tau$, we refer to the siblings that lie topologically to the left of that node. The ancestor of a node i in a tree is the set of node that lie on the path from i to the root. By "inclusive" ancestors of i, we refer to set containing the ancestors of i and i itself.

A quick inspection of Algorithm 2 (and Figure 2.(b)) reveals that the stack trace at i comprises of a sequence of proofs, one for each left sibling of the inclusive ancestors of i. On denoting these set of nodes of τ by $\mathsf{trace}(i)$, the main line in our rSVL instance is defined as

$$\emptyset = \{\pi_j\}_{j\in\mathsf{trace}(0^t)} \to \{\pi_j\}_{j\in\mathsf{trace}(0^{t-1}1)} \to \{\pi_j\}_{j\in\mathsf{trace}(0^{t-1}2)} \to \{\pi_j\}_{j\in\mathsf{trace}(10^{t-1})} \to \\ \ldots \to \{\pi_j\}_{j\in\mathsf{trace}(2^{t-1}1)} \to \{\pi_j\}_{j\in\mathsf{trace}(2^t)} \to \pi^T_{x\to x_T}.$$

Consequently, the label for a vertex consists of at most $2\log T$ proofs of the underlying proof system (i.e., eq.6), and we assume that labels with fewer proofs are padded accordingly.

For example, consider the toy rSVL instance from Figure 3. For the node 122 (red square), the path to the root is dashed in red, and thus its inclusive ancestors are $\{122, 12, 1, \varepsilon\}$. The $\mathsf{trace}(122) = \{0, 10, 11, 120, 121\}$ is hatched north east in red, and the correct label for the 122-th vertex is thus $\{\pi_0, \pi_{10}, \pi_{11}, \pi_{120}, \pi_{121}\}$ where, for example, $\pi_0 = \pi^4_{x_0\to x_4}$ and $\pi_{10} = \pi^2_{x_4\to x_6}$.

With the rSVL line defined as above, the successor and verifier functions follow quite logically. The verifier, given as input a vertex v and an index i, ensures that v is indeed the valid stack trace at i. The successor, on the other hand, generates the stack trace at $i+1$ given the stack trace at i. The formal description of the circuits V and S is given in Algorithms 3 and 4, respectively; an intuitive exposition follows.

The verifier circuit. On input an index i and a vertex v, parsed as a sequence of proofs $\{\tilde{\pi}_j\}_{j\in\mathsf{trace}(i)}$, the verifier V ensures that v is a valid stack trace at i by performing a series of

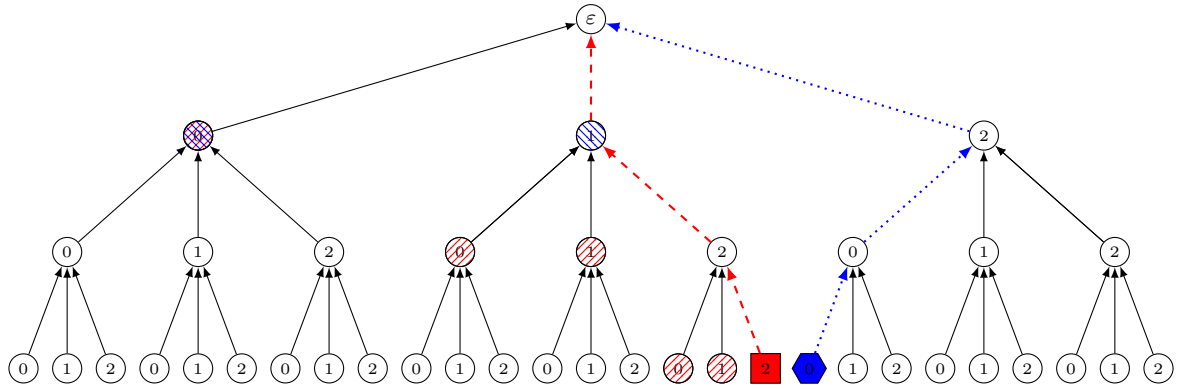

Figure 3: A schematic diagram of τ for $\mathsf{f}(N, x, 8)$. The resulting rSVL instance is of length 27. The red square node denotes the vertex $i = 122$, whereas the blue hexagon node denotes $i + 1 = 200$. The path from $i = 122$ to the root is dashed in red and the vertices in $\mathsf{trace}(i)$ are hatched north east in red. Similarly, the path from $i + 1 = 200$ to the root is dotted in blue and the vertices in $\mathsf{trace}(i + 1)$ are hatched north west in blue. (The vertex 0, as it appears in both the traces, is double-hatched.)

checks on the sequence. Recall from eq.6 that each proof $\tilde{\pi}_j$ in the sequence is of the form $(N, \ell_j, x_j, y_j, \boldsymbol{\mu}_j)$, where ℓ_j denotes the time parameter of the proof, x_j and y_j are its start and end points respectively, and $\boldsymbol{\mu}_j$ denotes the midpoints.

The verifier V first ensures that each $\tilde{\pi}_j$ is valid by invoking the verify algorithm of the underlying proof system (from §4.2).

Second, V checks whether the time parameter of each proof matches its level: the correct time parameter of a proof $\tilde{\pi}_j$ is $T/2^{|j|}$, where $|j|$ is the level of the node j in the tree τ (with the root at level 0 and the leaves at level t). As a concrete example, consider the node 122 from Figure 3 and the corresponding vertex $\{\pi_0, \pi_{10}, \pi_{11}, \pi_{120}, \pi_{121}\}$. For this vertex to be valid, π_0 must have length 4 (i.e., $\ell_0 = 4$) whereas π_{121} must be a base proof (i.e., $\ell_{121} = 1$).

Finally, provided that each proof satisfies the first two conditions, V checks if the end points of the proofs in the sequence chain appropriately. For every proof $\tilde{\pi}_j$ in the sequence, there are two possibilities depending on whether or not $\tilde{\pi}_j$ corresponds to a merged statement – we denote them cases (i) and (ii), respectively. In case (i), the start of $\tilde{\pi}_j$ is computed from the two proofs that precede $\tilde{\pi}_j$ in the sequence by merging them using eq.7. The start of $\tilde{\pi}_j$ in case (ii), however, just coincides with the endpoint of the proof that precedes it in the sequence (and in case $\tilde{\pi}_j$ is the first proof in the sequence, it must start at x).

Going back to the earlier example, the sequence of proofs $\{\pi_0, \pi_{10}, \pi_{11}, \pi_{120}, \pi_{121}\}$ is valid if

$$\pi_0 \leftrightarrow \pi_{10} \leftrightarrow, \pi_{11} \leftsquigarrow \pi_{120} \leftrightarrow \pi_{121}, \tag{9}$$

where the '$\leftsquigarrow$' denotes case (i) and the '$\leftrightarrow$' denotes case (ii). That is, for example, $x_{10} = y_0$ and $x_{121} = y_{120}$ but since π_{120} is a merged proof, x_{120} is computed from π_{10} and π_{11} using eq.7.

The successor circuit. Given as input a vertex v, the successor circuit S first uses a function[6] $\mathsf{index}(\cdot)$ to extract the index i that is implicitly embedded in the sequence of proofs in v. Next, it confirms whether or not v is the valid i-th vertex on the line, i.e. the stack trace at i, by invoking the verifier V. In case v is invalid, S forms a self-loop at v; otherwise, v is the valid stack trace at i and S utilises it to compute the stack trace at $i + 1$.

[6]To be precise, the function $\mathsf{index}(\cdot)$ on input a sequence of proof v computes the index i as follows: it counts the number of proofs of length k in v and then encodes this count in the $\log \ell$-th trit position of i. If there are more than two proofs of a particular length, then we assume that the function just returns $\perp$.

$\mathsf{V}_{N,x,T}(v, i)$ ▷ i is a t-trit string

1: **if** $i > 3^{\log T}$ **then return** 0
2: Parse $v =: \{\tilde{\pi}_j\}_{j \in \mathsf{trace}(i)}$
3: Set $x_{next} = x$ ▷ Set starting point of the chain
4: **for** $j \in \mathsf{trace}(i)$ **do** ▷ Verify the sequence topologically from left to right
5: Parse $\tilde{\pi}_j =: (N, x_j, y_j, \ell_j, \boldsymbol{\mu}_j)$
6: **if** $\mathsf{verify}(\tilde{\pi}_j) = 0$ or $\ell_j \neq T/2^{|j|}$ **then return** 0 ▷ Checks 1 and 2
7: **else if** $x_j \neq x_{next}$ **then return** 0 ▷ Check 3
8: **else** ▷ Compute the next point on the chain
9: **if** $j[|j|] = 1$ **then** ▷ case (i): second recursion
10: Set $\mu := x_j$ and compute $r = H(\mu, x_{j-1}, y_j, \ell_j)$
11: Set $x_{next} = x_{j-1}^r \circ \mu$ ▷ Compute the merged statement
12: **else** Set $x_{next} := y_j$ ▷ case (ii): first or third recursion
13: **end if**
14: **end if**
15: **end for**
16: **return** 1 ▷ Valid stack trace at i

Algorithm 3: The verifier circuit for our rSVL instance.

S first simulates the next recursion in the pipeline by adding the base proof π_i to $v := \{\tilde{\pi}_j\}_{j \in \mathsf{trace}(i)}$ (cf. lines 5 to 11 in Algorithm 4 for the exact computation involved) and then keeps merging the last *three* proofs in this sequence as long as they have the same time parameter. In particular, in the case that π_i corresponds to a merged statement (i.e., if $i[t] = 2$) — since some recursive call to f (up the tree τ) has been completed — S has to reconstruct the resulting proof. We denote the node in τ where this recursive call originates by $\mathsf{source}(i)$, and it can be obtained by truncating the trailing 2s of i. We refer to Algorithm 4 (lines 14 and 15) for the exact procedure used for reconstructing the proof for $\mathsf{source}(i)$, but once it possesses this proof, S has all the components of the next vertex on the line, the stack trace at $i + 1$.

For example, let's consider the successor circuit applied to $\{\pi_0, \pi_{10}, \pi_{11}, \pi_{120}, \pi_{121}\}$, the $i = 122$-th vertex in the rSVL instance given in Figure 3. S first computes π_{122}, and since this concludes the recursive call $\mathsf{f}(N, x^4, 4)$ at the (source) vertex 1, S reconstructs the corresponding proof π_1 by merging twice: first it merges π_{120}, π_{121} and π_{122} (using eq.8) to reconstruct π_{12}, and then it merges π_{10}, π_{11} and π_{12} to obtain π_1. Finally it assembles the next vertex ($i + 1 = 200$) as $\{\pi_0, \pi_1\}$, and since $\mathsf{trace}(200) = \{0, 1\}$ the newly assembled proof indeed is the stack trace at $i + 1$.

However applying the successor again, as no merging is involved, requires simply adding the base proof π_{200} to the input label $\{\pi_0, \pi_1\}$. That is, the label for the 201-th vertex is $\{\pi_0, \pi_1, \pi_{200}\}$.

5.3 Hardness

In this section we state and prove Theorem 1, the formal counterpart of Theorem 1 from §1.2.

Theorem 1. *For a security parameter n, let (N, x, T) be sampled as in Assumption 2 and*

$$\mathsf{S} := \mathsf{S}_{N,x,T} : \{0,1\}^m \to \{0,1\}^m \text{ and } \mathsf{V} := \mathsf{V}_{N,x,T} : \{0,1\}^m \times [2^m]$$

be defined as in Algorithms 3 and 4, where $m := m(T, n) = 2 \log T \cdot ((\log T + 4) \cdot n$. The family of distributions $\{(\mathsf{S}, \mathsf{V}, \emptyset, T)\}_{n \in \mathbb{N}}$ constitutes a family of hard rSVL instances relative to the random oracle H.

Proof. First, in Claim 1.1 we show that the rSVL instance is efficient: i.e., S and V are both polynomial-sized circuits. Then, to establish hardness, we show that any adversary that runs

$\mathsf{S}_{N,x,T}(v)$

1: $i := \mathsf{index}(v)$ ▷ Extract the index
2: **if** $\mathsf{V}(v,i) = 0$ or $i \geq 3^{\log T}$ **then return** v ▷ Invalid stack trace at i or $i \geq L$: form self-loop
3: Parse $v =: \{\tilde{\pi}_j\}_{j \in \mathsf{trace}(i)}$
4: **for** each $j \in \mathsf{trace}(i)$ **do** Parse $\tilde{\pi}_j =: (N, x_j, y_j, \ell_j, \boldsymbol{\mu}_j)$
5: **if** $i[t] = 2$ **then** ▷ Compute start of next base proof
6: Set $\mu = y_{i-1}$ and compute $r = H(\mu, x_{i-1}, y_i, 1)$
7: Compute $x_{next} = x_{i-1}^r \circ \mu$ ▷ case (i): compute the merged statement
8: **else**
9: Let l denote the last index in $\mathsf{trace}(i)$
10: Set $x_{next} = y_l$ ▷ case (ii)
11: **end if**
12: Set $\tilde{\pi}_i := \pi_i := \mathsf{prove}(N, x_{next}, 1, x_{next}^2)$ ▷ Next base proof
13: **if** $i[t] = 2$ **then** ▷ Merge
14: Let $s := \mathsf{source}(i)$
15: Set $\tilde{\pi}_s := (N, 2\ell_{s0}, x_{s0}, y_{s1}, y_{s0}, y_{s20}, y_{s220}, \ldots, y_{i-2})$ ▷ Reconstruct proof $\tilde{\pi}_s$ for source
16: **end if**
17: **return** $\{\tilde{\pi}_j\}_{j \in \mathsf{trace}(i+1)}$ ▷ Return stack trace at $i+1$

Algorithm 4: The successor circuit for our rSVL instance.

in time $\mathsf{poly}(n)$, making up to $Q = Q(n) \leq \mathsf{poly}(n)$ queries to the random oracle H, has a negligible probability of success.

Recall that by Definition 4 the adversary can solve an rSVL instance in two ways: find either (i) the real sink, which in our case contains the value x^{2^T}; *or* (ii) a false positive i.e., a pair (v, i) s.t. $\mathsf{V}(v,i) = 1$ while $\mathsf{S}^i(\emptyset) \neq v$.

Let $p(n)$ denote the probability that a $\mathsf{poly}(n)$-time adversary on input (N, x, T) as above finds a type (i) solution: under Assumption 2, $p(n)$ is negligible in n (even if we put no bound on Q). In Claim 1.2 below we will show that the probability of a type (ii) solution is at most $4 \cdot Q/2^{n/2-1}$. Therefore, the total probability of the adversary breaking the hardness of our rSVL instance is

$$\Pr[\text{type (i)} \vee \text{type (ii)}] = \Pr[\text{type (i)}] + \Pr[\text{type (ii)}] \leq \frac{4 \cdot Q}{2^{n/2-1}} + p(n) \in \mathsf{negl}(n), \tag{10}$$

completing the proof. □

Claim 1.1. S *and* V *are both efficient, i.e. have size* $\mathsf{poly}(\log T, n)$ *which is* $\mathsf{poly}(n)$ *for* $T \in n^{\omega(1)}$.

Proof. As a first step, we show that the size of the input vertices $m(T,n)$ is $\mathsf{poly}(\log T, n)$. As noted in §5.2, a vertex consists of at most $2 \log T$ proofs of the underlying proof system. Since the proofs in consideration have time parameter at most T, from eq.6 we infer that $m(T,n) \leq 2 \log T \cdot ((\log T + 4) \cdot n)$, i.e. $m(T,n) \in \mathsf{poly}(\log T, n)$ as claimed.[7]

Next, let's consider the verifier circuit V given in Algorithm 3. Note that the size of V is dominated by the call to verify (line 5) and the group operation $\circ$ for QR_N^+ (line 10) inside the loop (line 3). Since the output of $\mathsf{trace}(\cdot)$ consists of at most $2 \log T$ elements, and verify and $\circ$ are both efficient, the size of V is roughly $2 \log T \cdot \mathsf{poly}(\log T, n)$ which is still $\mathsf{poly}(\log T, n)$.

A similar argument holds for the successor circuit S. □

[7] This can also be established by analysing Algorithm 2. Let $m(\cdot)$ denote the upper bound on the size of the vertices (in bits). This parameter is governed by the recursion $m(T) \leq 2 \log T + m(T/2)$, with $m(1) \leq 4n$. The $2 \log T$ factor here is the cost of storing completed proofs from the first two recursions, whereas $m(T/2)$ is the cost of computing the proof for the merged statement (which is half the length). Therefore $m(T) < 8 \log^2 T \cdot n \in \mathsf{poly}(\log T, n)$.

Claim 1.2. *For (N, x, T) as in the theorem, the probability that any adversary, which makes at most Q queries to the random oracle H, finds a solution of type (ii), i.e. a false positive (v, i) s.t. $\mathsf{V}(v, i) = 1$ but $\mathsf{S}^i(\emptyset) \neq v$, is upper bounded by $4 \cdot Q/2^{n/2-1}$.*

Proof. For the event "bad query" as defined in Definition 5, the probability that an adversary produces a solution of type (ii) is

$$\begin{aligned}
\Pr[\text{type (ii)}] &= \Pr[\text{type (ii)} \wedge (\text{bad query} \vee \neg\text{bad query})] \\
&= \Pr[\text{type (ii)}|\text{bad query}] \cdot \Pr[\text{bad query}] + \Pr[\text{type (ii)} \wedge \neg\text{bad query}] \\
&\leq \Pr[\text{type (ii)}|\text{bad query}] \cdot \Pr[\text{bad query}] + 1/2^{n+1} \qquad (11) \\
&\leq 3 \cdot Q/2^{n/2-1} + 1/2^{n/2-2} \qquad (12) \\
&\leq 4 \cdot Q/2^{n/2-1}.
\end{aligned}$$

The upper bound in eq.12 above directly follows Lemma 2, and we argue below that eq.11 is a consequence of Lemma 3. The properties of $QR^\star_N$ (i.e. the generators of QR^+_N) that were discussed in §3.2 will be crucial as it allows repeated application of Lemma 3.

Let's suppose that the adversary outputs a solution (v, i) of type (ii), i.e., $\mathsf{V}(v, i) = 1$ but $\mathsf{S}^i(\emptyset) \neq v$, *without* having made a bad query. Since V accepts, v is of the form $\{\tilde{\pi}_j\}_{j \in \mathsf{trace}(i)}$, and this sequence is guaranteed to be a valid chain starting at x as described in §5.2. We argue that, provided $x \in QR^\star_N$, the adversary *could not* have output the type (ii) solution (v, i) since such a vertex v would equal $\mathsf{S}^i(\emptyset)$ and hence lie on the rSVL line leading to a contradiction. Since a random $x \in QR^+_N$ also belongs to $QR^\star_N$ with a overwhelming probability of $1 - 1/2^{n/2-2}$ (cf. eq.4), the upper bound in eq.11 follows.

Provided $x \in QR^\star_N$ and that the adversary never makes a bad query, let's see why the type (ii) solution it outputs lies on the rSVL line. Assume that $v =: \{\tilde{\pi}_j\}_{j \in \mathsf{trace}(i)} = \{\tilde{\pi}_{j_1}, \tilde{\pi}_{j_2}, \ldots, \tilde{\pi}_{j_\ell}\}$. Note that $\tilde{\pi}_{j_1}$ is of the form $\tilde{\pi}^{\ell_1}_{x_1 \to y_1}$, where

$$x_1 = x \text{ and } y_1 = x^{2^{\ell_1}}.$$

Since the verifier guarantees that the start x, the end point y_1 and the time parameter ℓ_1 all match the honestly generated proof, as a consequence of the soundness of the proof system (Lemma 3), we get $\tilde{\pi}^{\ell_1}_{x \to y_1} = \pi^{\ell_1}_{x \to y_1}$.

Next, there are two possibilities: either $\tilde{\pi}_{j_1} \leftrightarrow \tilde{\pi}_{j_2}$ or $\tilde{\pi}_{j_1} \nleftrightarrow \tilde{\pi}_{j_2}$ (with $\leftrightarrow$ and $\nleftrightarrow$ as defined in eq.9). In the first case, $x_{j_2} = y_1$ and by the property of the generators given in eq.5, we have $y_1 \in QR^\star_N$. Since $y_1 \in QR^\star_N$, we can again apply Lemma 3 and therefore $\tilde{\pi}_{j_2} = \pi_{j_2}$. As for the second case, let $\tilde{\pi}_{j_2} =: \tilde{\pi}^{\ell_2}_{x_2 \to y_2}$. Since we assume that the adversary did not make bad queries, it is guaranteed that $x_2 \in QR^\star_N$ and, by Lemma 3, we get $\tilde{\pi}_{j_2} = \pi_{j_2}$.

On iterating the above argument over all the proofs in v, we get $v = \{\pi_j\}_{j \in \mathsf{trace}(i)} = \mathsf{S}^i(\emptyset)$ contradicting the premise of the claim.

□

References

[1] Abbot, T., Kane, D., and Valiant, P. On algorithms for Nash equilibria. Unpublished manuscript, 2004. http://web.mit.edu/tabbott/Public/final.pdf. (Cited on pages 1 and 8.)

[2] Angel, O., Bubeck, S., Peres, Y., and Wei, F. Local max-cut in smoothed polynomial time. In *49th Annual ACM Symposium on Theory of Computing* (Montreal, QC, Canada, June 19–23, 2017), H. Hatami, P. McKenzie, and V. King, Eds., ACM Press, pp. 429–437. (Cited on page 6.)

[3] Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S. P., and Yang, K. On the (im)possibility of obfuscating programs. *J. ACM 59*, 2 (2012), 6:1–6:48. (Cited on page 1.)

[4] Bitansky, N., Paneth, O., and Rosen, A. On the cryptographic hardness of finding a Nash equilibrium. In *56th Annual Symposium on Foundations of Computer Science* (Berkeley, CA, USA, Oct. 17–20, 2015), V. Guruswami, Ed., IEEE Computer Society Press, pp. 1480–1498. (Cited on pages 1, 2, 6 and 8.)

[5] Blum, M. Coin flipping by telephone. In *Advances in Cryptology – CRYPTO'81* (Santa Barbara, CA, USA, 1981), A. Gersho, Ed., vol. ECE Report 82-04, U.C. Santa Barbara, Dept. of Elec. and Computer Eng., pp. 11–15. (Cited on page 11.)

[6] Boneh, D., Bünz, B., and Fisch, B. A survey of two verifiable delay functions. Cryptology ePrint Archive, Report 2018/712, 2018. https://eprint.iacr.org/2018/712. (Cited on pages 10 and 12.)

[7] Boodaghians, S., Kulkarni, R., and Mehta, R. Nash equilibrium in smoothed polynomial time for network coordination games. *CoRR abs/1809.02280* (2018). (Cited on page 6.)

[8] Buhrman, H., Fortnow, L., Koucký, M., Rogers, J. D., and Vereshchagin, N. Does the polynomial hierarchy collapse if onto functions are invertible? *Theory of Computing Systems 46*, 1 (Dec 2008), 143. (Cited on page 6.)

[9] Buresh-Oppenheim, J. On the TFNP complexity of factoring. Unpublished, http://www.cs.toronto.edu/~bureshop/factor.pdf, 2006. (Cited on page 6.)

[10] Canetti, R., Chen, Y., Reyzin, L., and Rothblum, R. D. Fiat-Shamir and correlation intractability from strong KDM-secure encryption. In *Advances in Cryptology – EUROCRYPT 2018, Part I* (Tel Aviv, Israel, Apr. 29 – May 3, 2018), J. B. Nielsen and V. Rijmen, Eds., vol. 10820 of *Lecture Notes in Computer Science*, Springer, Heidelberg, Germany, pp. 91–122. (Cited on page 12.)

[11] Chen, X., Deng, X., and Teng, S. Settling the complexity of computing two-player Nash equilibria. *J. ACM 56*, 3 (2009). (Cited on pages 1 and 6.)

[12] Choudhuri, A. R., Hubacek, P., Kamath, C., Pietrzak, K., Rosen, A., and Rothblum, G. N. Finding a nash equilibrium is no easier than breaking fiat-shamir. Cryptology ePrint Archive, Report 2019/549, 2019. https://eprint.iacr.org/2019/549. (Cited on pages 1, 2, 4, 5 and 8.)

[13] Cover, T. M., and Thomas, J. A. *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*. Wiley-Interscience, New York, NY, USA, 2006. (Cited on page 27.)

[14] Daskalakis, C., Goldberg, P. W., and Papadimitriou, C. H. The complexity of computing a Nash equilibrium. *SIAM J. Comput. 39*, 1 (2009), 195–259. (Cited on pages 1 and 6.)

[15] Daskalakis, C., and Papadimitriou, C. H. Continuous local search. In *22nd Annual ACM-SIAM Symposium on Discrete Algorithms* (San Francisco, CA, USA, Jan. 23–25, 2011), D. Randall, Ed., ACM-SIAM, pp. 790–804. (Cited on pages 6 and 7.)

[16] Deng, X., Edmonds, J. R., Feng, Z., Liu, Z., Qi, Q., and Xu, Z. Understanding PPA-completeness. In *31st Conference on Computational Complexity (CCC 2016)* (Dagstuhl, Germany, 2016), R. Raz, Ed., vol. 50 of *Leibniz International Proceedings in Informatics (LIPIcs)*, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, pp. 23:1–23:25. (Cited on page 6.)

[17] Ephraim, N., Freitag, C., Komargodski, I., and Pass, R. Continuous verifiable delay functions. Cryptology ePrint Archive, Report 2019/619, 2019. https://eprint.iacr.org/2019/619. (Cited on page 6.)

[18] Fiat, A., and Shamir, A. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology – CRYPTO'86* (Santa Barbara, CA, USA, Aug. 1987), A. M. Odlyzko, Ed., vol. 263 of *Lecture Notes in Computer Science*, Springer, Heidelberg, Germany, pp. 186–194. (Cited on pages 2 and 12.)

[19] Fischlin, R., and Schnorr, C.-P. Stronger security proofs for RSA and Rabin bits. *Journal of Cryptology 13*, 2 (2000), 221–244. (Cited on page 9.)

[20] Garg, S., Gentry, C., Halevi, S., Raykova, M., Sahai, A., and Waters, B. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *FOCS* (2013). (Cited on page 1.)

[21] Garg, S., Pandey, O., and Srinivasan, A. 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* (2016), pp. 579–604. (Cited on pages 1 and 6.)

[22] Hofheinz, D., and Kiltz, E. The group of signed quadratic residues and applications. In *Advances in Cryptology – CRYPTO 2009* (Santa Barbara, CA, USA, Aug. 16–20, 2009), S. Halevi, Ed., vol. 5677 of *Lecture Notes in Computer Science*, Springer, Heidelberg, Germany, pp. 637–653. (Cited on page 9.)

[23] Hubáček, P., Naor, M., and Yogev, E. The journey from NP to TFNP hardness. In *8th Innovations in Theoretical Computer Science Conference, ITCS 2017, January 9-11, 2017, Berkeley, CA, USA* (2017), pp. 60:1–60:21. (Cited on page 6.)

[24] Hubáček, P., and Yogev, E. 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* (2017), pp. 1352–1371. (Cited on pages 1, 2, 6 and 7.)

[25] Jeřábek, E. Integer factoring and modular square roots. *J. Comput. Syst. Sci. 82*, 2 (2016), 380–394. (Cited on pages 1 and 6.)

[26] Johnson, D. S., Papadimitriou, C. H., and Yannakakis, M. How easy is local search? *Journal of Computer and System Sciences 37*, 1 (1988), 79 – 100. (Cited on page 6.)

[27] Kalai, Y. T., Khurana, D., and Sahai, A. Statistical witness indistinguishability (and more) in two messages. In *Advances in Cryptology – EUROCRYPT 2018, Part III* (Tel Aviv, Israel, Apr. 29 – May 3, 2018), J. B. Nielsen and V. Rijmen, Eds., vol. 10822 of *Lecture Notes in Computer Science*, Springer, Heidelberg, Germany, pp. 34–65. (Cited on page 12.)

[28] Kalai, Y. T., Rothblum, G. N., and Rothblum, R. D. From obfuscation to the security of Fiat-Shamir for proofs. In *Advances in Cryptology – CRYPTO 2017, Part II* (Santa Barbara, CA, USA, Aug. 20–24, 2017), J. Katz and H. Shacham, Eds., vol. 10402 of *Lecture Notes in Computer Science*, Springer, Heidelberg, Germany, pp. 224–251. (Cited on page 12.)

[29] Kintali, S., Poplawski, L., Rajaraman, R., Sundaram, R., and Teng, S. Reducibility among fractional stability problems. *SIAM Journal on Computing 42*, 6 (2013), 2063–2113. (Cited on page 6.)

[30] Komargodski, I., Naor, M., and Yogev, E. White-box vs. black-box complexity of search problems: Ramsey and graph property testing. In *58th Annual Symposium on Foundations of Computer Science* (2017), IEEE Computer Society Press, pp. 622–632. (Cited on page 6.)

[31] Komargodski, I., and Segev, G. From minicrypt to obfustopia via private-key functional encryption. In *Advances in Cryptology – EUROCRYPT 2017, Part I* (Paris, France, Apr. 30 – May 4, 2017), J. Coron and J. B. Nielsen, Eds., vol. 10210 of *Lecture Notes in Computer Science*, Springer, Heidelberg, Germany, pp. 122–151. (Cited on pages 1 and 6.)

[32] Lund, C., Fortnow, L., Karloff, H., and Nisan, N. Algebraic methods for interactive proof systems. *J. ACM 39*, 4 (Oct. 1992), 859–868. (Cited on pages 1 and 4.)

[33] Mahmoody, M., and Xiao, D. On the power of randomized reductions and the checkability of SAT. In *2010 IEEE 25th Annual Conference on Computational Complexity* (June 2010), pp. 64–75. (Cited on page 6.)

[34] Megiddo, N., and Papadimitriou, C. H. On total functions, existence theorems and computational complexity. *Theor. Comput. Sci. 81*, 2 (1991), 317–324. (Cited on pages 5 and 6.)

[35] Papadimitriou, C. H. On the complexity of the parity argument and other inefficient proofs of existence. *J. Comput. Syst. Sci. 48*, 3 (1994), 498–532. (Cited on pages 1, 5, 6 and 7.)

[36] Pietrzak, K. Simple Verifiable Delay Functions. In *10th Innovations in Theoretical Computer Science Conference (ITCS 2019)* (Dagstuhl, Germany, 2018), A. Blum, Ed., vol. 124 of *Leibniz International Proceedings in Informatics (LIPIcs)*, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, pp. 60:1–60:15. (Cited on pages 3, 4, 12, 13 and 14.)

[37] Rivest, R. L., Shamir, A., and Wagner, D. A. Time-lock puzzles and timed-release crypto. Tech. rep., Cambridge, MA, USA, 1996. (Cited on pages 2 and 9.)

[38] Rosen, A., Segev, G., and Shahaf, I. Can PPAD hardness be based on standard cryptographic assumptions? In *TCC 2017: 15th Theory of Cryptography Conference, Part II* (Baltimore, MD, USA, Nov. 12–15, 2017), Y. Kalai and L. Reyzin, Eds., vol. 10678 of *Lecture Notes in Computer Science*, Springer, Heidelberg, Germany, pp. 747–776. (Cited on page 6.)

[39] Sotiraki, K., Zampetakis, M., and Zirdelis, G. PPP-completeness with connections to cryptography. Cryptology ePrint Archive, Report 2018/778, 2018. https://eprint.iacr.org/2018/778. (Cited on page 6.)

[40] Valiant, P. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In *TCC 2008: 5th Theory of Cryptography Conference* (San Francisco, CA, USA, Mar. 19–21, 2008), R. Canetti, Ed., vol. 4948 of *Lecture Notes in Computer Science*, Springer, Heidelberg, Germany, pp. 1–18. (Cited on pages 16 and 18.)

[41] von zur Gathen, J., and Shparlinski, I. E. Generating safe primes. *J. Mathematical Cryptology 7*, 4 (2013), 333–365. (Cited on page 10.)

A Proof of Lemma 2

Lemma 2 (Bad queries are hard to find)**.** *For any $N = p \cdot q$ where $p = 2p' + 1, q = 2q' + 1$ are $(n+1)$-bit safe primes, the following holds: any adversary that makes at most Q queries to the random oracle H will make a bad query with probability at most*

$$\frac{3 \cdot Q}{2^n}$$

Proof. Recall that a query is a tuple (μ, x, y, T) where $\mu, x, y \in QR_N^+$ and $T \in \mathbb{Z}$, and the query (μ, x, y, T) is *bad* if $x \in QR_N^\star$ and moreover either

(i) $x' \notin QR_N^\star$; or

(ii) $\left(x^{2^T} \neq y \text{ or } \mu \neq x^{2^{T/2}}\right)$ and $x'^{2^{T/2}} = y'$,

where $r := H(\mu, x, y, T)$, $x' := x^r \circ \mu$ and $y' := \mu^r \circ y$.

Since the range of the range of the random oracle H is $\{0,1\}^{3n}$ (cf. §4.1), its output is distributed 2^{-n}-close to uniform over $\mathbb{Z}_{p'q'}$. In the analysis below we assume that it is actually uniform over $\mathbb{Z}_{p'q'}$ and then apply the data processing lemma [13] () to get the desired bound. That is, we show that

$$\Pr_r[(y' = x'^{2^{T/2}}) \ \vee \ (x' \notin QR_N^\star)] \leq 3/2^n.$$

where r is chosen uniformly at random from $\mathbb{Z}_{p'q'}$. Using $\Pr[a \vee b] = \Pr[a \wedge \bar{b}] + \Pr[b]$, this can be rewritten as

$$\Pr_r[(y' = x'^{2^{T/2}}) \wedge (x' \in QR_N^\star)] + \Pr_r[x' \notin QR_N^\star] \leq 3/2^n. \tag{13}$$

We bound the two probabilities separately in Claims 1.3 and 1.4, and the lemma follows by a union bound over all the queries. □

Claim 1.3. $\Pr_r[x' \notin QR_N^\star] \leq 2/2^n$.

Proof. By e_μ we denote the unique value in $\mathbb{Z}_{p'q'}$ satisfying $x^{e_\mu} = \mu$ (it's unique as $\mu \in \langle x \rangle = QR_N^+$ and $|QR_N^+| = p'q'$). As $x, \mu \in QR_N^+$, also $x' = x^r \circ \mu = x^{r+e_\mu}$ is in QR_N^+, and $\langle x' \rangle = QR_N^+$ holds if $ord(x') = p'q'$, which is the case except if $(r + e_\mu) = 0 \bmod p'$ or $(r + e_\mu) = 0 \bmod q'$ or equivalently (using that $2^n < \min(p', q')$) if

$$r \in \mathcal{B} := \left\{\mathbb{Z}_{2^n} \cap \{(-e_\mu \bmod p'), (-e_\mu \bmod q')\}\right\} \ . \tag{14}$$

Clearly $|\mathcal{B}| \leq 2$ and the claim follows. □

Claim 1.4. $\Pr_r[(y' = x'^{2^{T/2}}) \wedge (x' \in QR_N^\star)] \leq 1/2^n$.

Proof. If $y \notin QR_N^+$, then also $y' = \mu^r \circ y \notin QR_N^+$ (as $a \in QR_N^+, b \notin QR_N^+$ implies $a \circ b \notin QR_N^+$). As $x' \in QR_N^\star$ and $y' \neq x'^{2^{T/2}}$ cannot hold simultaneously in this case the probability in the claim is 0. From now on we consider the case $y \in QR_N^+$. We have

$$\Pr_r[y' = x'^{2^{T/2}} \wedge x' \in QR_N^\star] = \Pr_r[y' = x'^{2^{T/2}} \mid x' \in QR_N^\star] \cdot \Pr_r[x' \in QR_N^\star] \tag{15}$$

For the second factor in (15) we have with $\mathcal{B}$ as in (14)

$$\Pr_r[x' \in QR_N^\star] = \frac{2^n - |\mathcal{B}|}{2^n} \ . \tag{16}$$

Conditioned on $x' \in QR_N^\star$ the r is uniform in $\mathbb{Z}_{2^n} \setminus \mathcal{B}$, so the first factor in (15) is

$$\Pr_r[y' = x'^{2^{T/2}} \mid x' \in QR_N^\star] = \Pr_{r \in \mathbb{Z}_{2^n} \setminus \mathcal{B}}[y' = x'^{2^{T/2}}] \ . \tag{17}$$

Let $e_y \in \mathbb{Z}_{p'q'}$ be the unique value such that $x^{e_y} = y$. Using $\langle x \rangle = QR_N^+$ in the last step below we can rewrite

$$\begin{aligned} y' = x'^{2^{T/2}} &\iff \\ \mu^r y = (x^r \mu)^{2^{T/2}} &\iff \\ x^{r \cdot e_\mu + e_y} = x^{(r+e_\mu) \circ 2^{T/2}} &\iff \\ r \cdot e_\mu + e_y = (r + e_\mu) \cdot 2^{T/2} \bmod p'q' & \end{aligned}$$

rearranging terms

$$r(e_\mu - 2^{T/2}) + e_y - e_\mu 2^{T/2} = 0 \bmod p'q' \ . \tag{18}$$

If $e_\mu = 2^{T/2}$ this becomes

$$e_y - 2^T = 0 \bmod p'q'$$

which does not hold as by assumption we have $y \neq x^{2^T}$. So from now on we assume $e_\mu \neq 2^{T/2} \bmod p'q'$. Then for $a = e_\mu - 2^{T/2} \neq 0 \bmod p'q'$ (and $b = e_y - e_\mu 2^{T/2}$) eq.(18) becomes

$$r \cdot a = b \bmod p'q'$$

which holds for at most one choice of r from its domain $\mathbb{Z}_{2^n} \setminus \mathcal{B}$, thus

$$\Pr_{r \in \mathbb{Z}_{2^n} \setminus \mathcal{B}}[y' = x'^{2^{T/2}}] \leq \frac{1}{2^n - |\mathcal{B}|}$$

and the claim follows from the above equation and (15)-(17) as

$$\begin{aligned} \Pr_r[(y' = x'^{2^{T/2}}) \wedge (x' \in QR_N^\star)] &= \Pr_{r \in \mathbb{Z}_{2^n} \setminus \mathcal{B}}[y' = x'^{2^{T/2}}] \cdot \Pr_r[x' \in QR_N^\star] \\ &\leq \frac{1}{2^n - |\mathcal{B}|} \cdot \frac{2^n - |\mathcal{B}|}{2^n} \leq \frac{1}{2^n} \ . \end{aligned}$$

□