On the Average-Case Hardness of Total Search Problems

by

Chethan Kamath

May, 2020

A thesis presented to the Graduate School of the
Institute of Science and Technology Austria, Klosterneuburg, Austria
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

Institute of Science and Technology

The thesis of Chethan Kamath, titled *On the Average-Case Hardness of Total Search Problems*, is approved by:

Supervisor: Krzysztof Pietrzak, IST Austria, Klosterneuburg, Austria

Signature: _____

Committee Member: Vladimir Kolmogorov, IST Austria, Klosterneuburg, Austria

Signature: _____

Committee Member: Alon Rosen, IDC Herzliya, Israel

Signature: _____

Defense Chair: Calin Gŭet, IST Austria, Klosterneuburg, Austria

Signature: _____

signed page is on file

IST Austria Thesis, ISSN: 2663-337X

I hereby declare that this thesis is my own work and that it does not contain other people's work without this being so stated; this thesis does not contain my previous work without this being stated, and the bibliography contains all the literature that I used in writing the dissertation.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee, and that this thesis has not been submitted for a higher degree to any other university or institution.

I certify that any republication of materials presented in this thesis has been approved by the relevant publishers and co-authors.

Signature: _____

Chethan Kamath
May, 2020

signed page is on file

Abstract

A search problem lies in the complexity class **FNP** if a solution to the given instance of the problem can be verified efficiently. The complexity class **TFNP** consists of all search problems in **FNP** that are *total* in the sense that a solution is guaranteed to exist. **TFNP** contains a host of interesting problems from fields such as algorithmic game theory, computational topology, number theory and combinatorics. Since **TFNP** is a semantic class, it is unlikely to have a complete problem. Instead, one studies its syntactic subclasses which are defined based on the combinatorial principle used to argue totality. Of particular interest is the subclass **PPAD**, which contains important problems like computing Nash equilibrium for bimatrix games and computational counterparts of several fixed-point theorems as complete. In the thesis, we undertake the study of *average-case* hardness of **TFNP**, and in particular its subclass **PPAD**.

Almost nothing was known about average-case hardness of **PPAD** before a series of recent results showed how to achieve it using a cryptographic primitive called program obfuscation. However, it is currently not known how to construct program obfuscation from standard cryptographic assumptions. Therefore, it is desirable to relax the assumption under which average-case hardness of **PPAD** can be shown. In the thesis we take a step in this direction. First, we show that assuming the (average-case) hardness of a number-theoretic problem related to factoring of integers, which we call Iterated-Squaring, **PPAD** is hard-on-average in the *random-oracle model*. Then we strengthen this result to show that the average-case hardness of **PPAD** reduces to the (adaptive) soundness of the Fiat-Shamir Transform, a well-known technique used to compile a public-coin interactive protocol into a non-interactive one. As a corollary, we obtain average-case hardness for **PPAD** in the random-oracle model assuming the worst-case hardness of #SAT. Moreover, the above results can all be strengthened to obtain average-case hardness for the class **CLS** $\subseteq$ **PPAD**.

Our main technical contribution is constructing *incrementally-verifiable* procedures for computing Iterated-Squaring and #SAT. By incrementally-verifiable, we mean that every intermediate state of the computation includes a proof of its correctness, and the proof can be updated and verified in polynomial time. Previous constructions of such procedures relied on strong, non-standard assumptions. Instead, we introduce a technique called *recursive proof-merging* to obtain the same from weaker assumptions.

Acknowledgments

First and foremost, I would like to express my deepest gratitude to my adviser, Krzysztof, for handing me an opportunity to pursue a PhD. Krzysztof is a unending source of interesting problems and more often than not to their solutions. I really appreciate his calm, patient approach to guidance, which allowed me develop not only as a researcher but also as a person during my PhD.

Secondly I would like to profusely thank Alon and Pavel: Alon for having given me the chance to visit IDC Herzliya, and Pavel for his tutelage during my time there. The discussions with Alon and Pavel were enlightening, and enabled me to appreciate both the intuitive and meticulous nature of research. The inception of this thesis can be traced back to the research conducted during this visit.

I would also like to give special thanks to Georg for having co-supervised me during the early phase of the PhD, Vladimir for being part of my committee, and Calin for agreeing to chair my defense.

I am grateful to the co-authors of all the papers that I was involved during my PhD. In particular, I would like to thank the co-authors of the two papers that constitute the bulk of this thesis: they were closely involved not only in developing most of the ideas therein, but also in writing most of its content. (They say that a thesis doesn't write itself, but in my case it mostly did.)

For the six memorable years I am thankful to my colleagues and friends at IST Austria, particularly to the members of the Pietrzak group over the years with whom I had great discussions and even more fun. I also thank Stephan and Simon for their warm friendship and constant company which helped me plough through the PhD.

I gratefully acknowledge the financial support from the IST Austria Excellence Scholarship during my first year and European Research Council starting grant (259668-PSPC) and consolidator grant (682815-TOCNeT) during the rest of the PhD, which allowed me to focus on research without any monetary bothers.

Finally I would like to thank my family for their unwavering support.

About the Author

Chethan Kamath obtained his B.Tech degree in Computer Science from University of Kerala (at TKM College of Engineering) in 2009. In 2010 he joined Indian Institute of Science, Bangalore for an M.Sc in Computer Science with focus on cryptography. After its completion, he joined Institute of Science and Technology Austria in 2014 for his Ph.D studies in the group of Krzysztof Pietrzak. His research interests include foundations of cryptography, computational complexity theory, and theoretical computer science in general.

List of Publications

Publications related to the thesis:

1. Choudhuri, A. R., Hubáček, P., Kamath, C., Pietrzak, K., Rosen, A., and Rothblum, G. N. *Finding a Nash equilibrium is no easier than breaking Fiat-Shamir.* Symposium on Theory of Computing – STOC 2019.

2. Choudhuri, A. R., Hubáček, P., Kamath, C., Pietrzak, K., Rosen, A., and Rothblum, G. N. *PPAD-hardness via iterated squaring modulo a composite.* Cryptology ePrint Archive, Report 2019/667.

Rest of the publications:

1. Abusalah, H., Kamath, C., Klein, K., Pietrzak, K., and Walter, M. *Reversible proofs of sequential work.* EUROCRYPT 2019.

2. Fuchsbauer, G., Kamath, C., Klein, K., and Pietrzak, K. *Adaptively secure proxy re-encryption.* Public Key Cryptography – PKC 2019.

3. Alwen, J., Capretto, M., Cueto, M., Kamath, C., Klein, K., Pascual-Perez, G., Pietrzak, K., and Walter, M. *Keep the dirt: Tainted TreeKEM, an efficient and provably secure continuous group key agreement protocol.* Cryptology ePrint Archive, Report 2019/1489.

4. Alwen, J., Gazi, P., Kamath, C., Klein, K., Osang, G., Pietrzak, K., Reyzin, L., Rolinek, M., and Rybár, M. *On the memory-hardness of data-independent password-hashing functions.* AsiaCCS 2018.

5. Jafargholi, Z., Kamath, C., Klein, K., Komargodski, I., Pietrzak, K., and Wichs, D. *Be adaptive, avoid overcommitting.* CRYPTO 2017.

6. Alwen, J., Chen, B., Kamath, C., Kolmogorov, V., Pietrzak, K., and Tessaro, S. *On the complexity of scrypt and proofs of space in the parallel random oracle model.* EUROCRYPT 2016.

7. Fuchsbauer, G., Hanser, C., Kamath, C., and Slamanig, D. *Practical round-optimal blind signatures in the standard model from weaker assumptions.* Security in Communication Networks – SCN 2016.

8. Chatterjee, S., and Kamath, C. *A closer look at multiple forking: Leveraging (in)dependence for a tighter bound.* Algorithmica 2016.

Contents

List of Figures

List of Algorithms

List of Abbreviations

Complexity classes:

NP Non-deterministic Polynomial-time
FNP *Functional* Non-deterministic Polynomial-time
TFNP *Total* Functional Non-deterministic Polynomial-time
PTFNP Provable Total Functional Non-deterministic Polynomial-time
PPP Polynomial Pigeonhole Principle
PWPP Polynomial Weak Pigeonhole Principle
PPA Polynomial Parity Argument
PPAD Polynomial Parity Argument on Digraphs
PLS Polynomial Local Search
CLS Continuous Local Search
PP Probabilistic Polynomial-time
BPP Bounded Probabilistic Polynomial-time
RP Randomised Polynomial-time
NC Nick's Class
IP Interactive Protocols

Computational problems:

EOL End-of-Line
EOML End-of-Metered-Line
SVL Sink-of-Verifiable-Line
RSVL Relaxed-Sink-of-Verifiable-Line
DLP Discrete-Logarithm Problem
IS Iterated-Squaring

Cryptographic primitives:

OWF One-Way Function
OWP One-Way Permutation
PRF Pseudo-Random Generator
PRF Pseudo-Random Function
CRHF Collision-Resistant Hash Function
CIHF Correlation-Intractable Hash Function
FHE Fully-Homomorphic Encryption
IO Indistinguishability Obfuscation

Miscellaneous:

DAG Directed Acyclic Graph
IVC Incrementally-Verifiable Computation

Chapter 1
Introduction

1.1 Total Search Problems

Consider the following[1] combinatorial problem: for some parameter $M \in \mathbb{N}$, we are given an $M \times M$ triangular grid like the one in Figure 1.1.(a) with the vertices colored either red, blue or green, as prescribed by a function

$$c : [0, M-1] \times [0, M-1] \to \{R, G, B\},$$

where $[a, b]$ denotes $\{a, a+1, \ldots, b-1, b\}$. There are restrictions to the colouring of the vertices that lie on the boundary:

1. the ones at the bottom are all red: i.e., $\forall i \in [0, M-1] : c(0, i) = R$;
2. the ones on the left, excluding the one coloured red, are all green: i.e., $\forall i \in [1, M-1] : c(i, 0) = G$; and
3. the rest of the vertices, excluding the ones already coloured, are all blue: i.e., $\forall i \in [1, M-1] : c(M-1, i) = c(i, M-1) = B$.

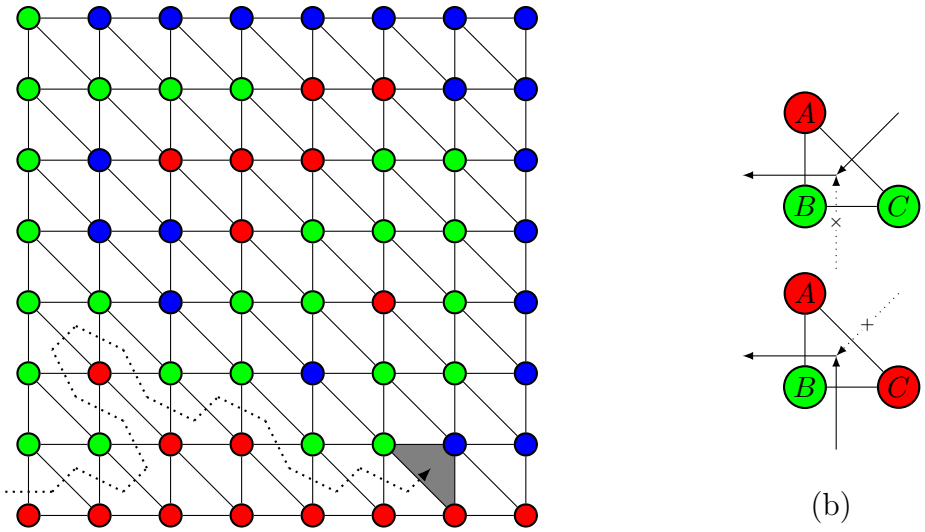
However, no restriction is placed to colouring the internal vertices. The goal of the problem is to find a trichromatic triangle, i.e., a triangle consisting of a red, green and blue vertex each. The problem is trivial to solve if the colouring function is represented explicitly, say as an $M \times M$ table: for example, an algorithm that simply searches all the $2M^2$ triangles is efficient. But what if we set $M = 2^m$ for some parameter $m \in \mathbb{N}$ and then present the colouring function *succinctly* as a circuit

$$\mathsf{C} : \{0,1\}^m \times \{0,1\}^m \to \{0, 1, 2\} \tag{1.1}$$

that is of size $\mathbf{poly}(m)$? The circuit C would take as input the coordinates of a vertex represented as two m-bit strings and return its colour encoded using $\{0, 1, 2\}$. For a algorithm to be deemed efficient, it would now have to run in time $\mathbf{poly}(m)$ and therefore the brute force approach from before is no longer efficient. Let us call this problem Sperner.

Totality of Sperner. Although Sperner may at first seem like any other search problem that one encounters in computational complexity theory, it has one distinguishing feature: a theorem of Sperner [101] guarantees the existence of a solution, i.e. trichromatic triangle. Hence, Sperner is *total*. To see this, let's consider a walk starting from the bottom left triangle as shown in Figure 1.1.(a). One enters this triangle via the green-red edge and continues the walk by entering one of the adjacent triangles always taking a *green-red edge with the green vertex on the left* — this is an invariant. Let us see what can be said about the walk.

[1] This example is taken from a lecture given by Constantinos Daskalakis at MIT.



(a) (b)

Figure 1.1: (a) Sperner's Lemma on an 8×8 grid. The walk is continued by always taking a green-red edge with the green vertex on the left (invariant) (b) Why the invariant rules out the walk looping.

1. Firstly, the walk cannot exit the grid because of the invariant: there is only one green-red edge (which is the one we took into the grid) and this cannot be used as an exit as it is incorrectly oriented.

2. Secondly, the walk cannot end up in a loop either. To see this, suppose for contradiction that the walk does loop at some ΔABC as shown in Figure 1.1.(b). Further suppose, without loss of generality, that the triangle was exited through the edge AB, which means that $c(A) = R$ and $c(B) = G$. Since the walk loops at ΔABC, it must be entered through both of the other edges AC and BC. However, we can argue that this is impossible without violating the invariant: if $c(C) = G$ then the triangle can be entered only via AC; on the other hand, if $c(C) = R$ then it can be entered only via BC. (In case ΔABC is trichromatic, then it cannot be *both* entered and exited).

Therefore, since the grid is finite, the only reason the walk halts must be because it encountered a trichromatic triangle.

1.1.1 TFNP and its Subclasses

Sperner is archetypal of the type of problems that we consider in this thesis. Firstly, it is a *search* problem: we are asked to find a solution, unlike decision problems which are posed as yes-no questions. Moreover, its search space (i.e., the set of all triangles) is exponentially-large but is succinctly-representable (via the colouring circuit C). Secondly, given a solution, it is possible to efficiently *verify* its validity: given ΔABC purported to be trichromatic, one simply checks that $\{c(A), c(B), c(C)\} = \{R, G, B\}$ using the circuit C. Finally, and most importantly, such a trichromatic triangle can be, as we saw, *guaranteed to exist* making the problem *total*. Let us formalise these notions a bit more (the full definitions are given later in §2.1).

A search problem P is defined by an efficiently-decidable set of instances $\mathcal{I} \subseteq \{0,1\}^*$ such that for every instance $I \in \mathcal{I}$ there exists a set of solutions $s(I) \subseteq \{0,1\}^*$. It is an **NP** search problem if the set of solutions is *also* efficiently-decidable: i.e., given a purported solution $S \in \{0,1\}^*$ to an instance $I \in \mathcal{I}$, checking whether $S \in s(I)$ is efficient in the size of the instance. The class of all **NP** search problems is called *Functional Non-deterministic Polynomial-time* (**FNP**). Thus, **FNP** is the search counterpart of the (decision) class **NP**. Moreover, an **NP** search problem is total if the set of solutions for every instance is non-empty. With the goal of collectively studying all such problems, including Sperner, Megiddo and Papadimitriou [78] defined the class *Total Functional Non-deterministic Polynomial-time* (**TFNP**). They went on to show (see Theorem 2) that this class is in fact the same as $\mathbf{F}(\mathbf{NP} \cap \mathbf{co\text{-}NP})$, i.e. the search counterpart of $\mathbf{NP} \cap \mathbf{co\text{-}NP}$. Here **co-NP** is the complementary class of **NP** (i.e., strings not in the language have succinct refutations).

TFNP contains a host of interesting, non-trivial problems from fields such as algorithmic game theory, computational topology, number theory and combinatorics. Perhaps the two most widely-known examples are:

1. Nash, the problem of finding the equilibrium strategy in a (bimatrix) game, which is guaranteed to exist by Nash's celebrated theorem [81], and

2. Factoring, the problem of computing the (unique) prime factorisation of a given integer, which is guaranteed to exist by the fundamental theorem of arithmetic.

The former is fundamental to understanding game theory and therefore economics, whereas the latter forms the backbone of cryptography and internet security. Studying the structure of such diverse problems and the relationship between them was one of the original motivation behind the definition of **TFNP**. For instance, reducing Factoring to Nash– i.e., showing that solving Nash is at least as hard as factoring integers – would bridge two seemingly different fields of research. One could then borrow ideas from one field and apply it to the other.

1.1.1.1 The Subclasses of TFNP

It is known that **TFNP** is unlikely to contain **FNP**-complete problems unless **NP** = **co-NP** [78] (see Theorem 3). Moreover, **TFNP** is a *semantic* class and therefore unlikely to have a complete problem.[2] Therefore, we need a different notion to qualify the hardness of problems in **TFNP**. To this end, one studies its *syntactic* subclasses which are defined based on the *combinatorial principle* used to argue totality [84]. For instance, one of the subclasses is defined based on the well-known pigeonhole principle (**PPP**) and another based on the parity argument (**PPA**). These subclasses do contain canonical complete problems: i.e., the subclass is defined as the set of all search problems that are polynomial-time Karp-reducible (Definition 8) to this canonical problem. Therefore in order to study the class in its entirety, it suffices to focus on the canonical complete problem.

[2] A complexity class is said to be *syntactic* if it possible to efficiently check whether an appropriately standardized machine (say a Turing machine or a Boolean circuit) indeed defines a language in the class. On the other hand, a class is *semantic* if this is *not* possible and there is a "promise" involved [85]. Phrased differently, for a syntactic class it is guaranteed that a machine defines *some* language in that class, whereas this might not be the case for a semantic class. Examples for syntactic classes are **P**, **NP** and **PP**; whereas **RP** and **co-RP** are semantic classes. It is believed that semantic classes are unlikely to have complete problems. A thorough discussion can be found at this thread on StackExchange.

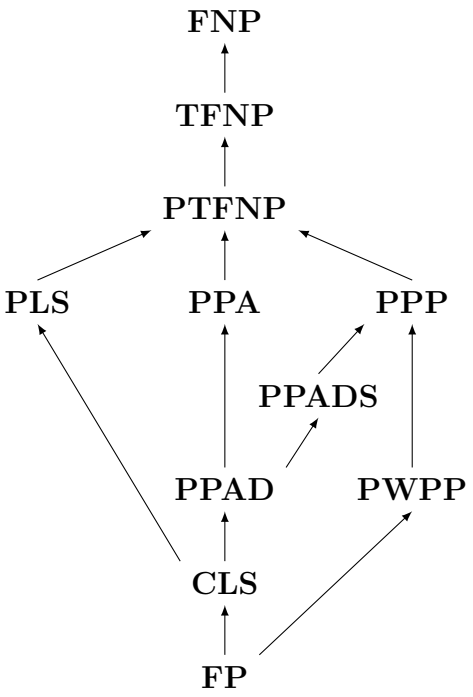


Figure 1.2: The **TFNP** landscape. The arrows indicate containment.

The subclasses of **TFNP**, along with the relationship between them, are illustrated in Figure 1.2. It is believed that the major subclasses **PPP**, **PPA** and **PLS** are fundamentally different [6]. More formally, it was shown in [6] that there exist oracles relative to which the complete problem of one is not (even) Turing-reducible to that of others. This is not surprising since the combinatorial existence theorem that is used to guarantee totality is of different flavour for different classes. With this in mind, we describe the subclasses below, especially the canonical complete problem that captures its structure, and mention some of the interesting problems that they contain (see Figure 1.4). Since one of the subclasses, **PPAD**, is the focus of this thesis, we will cover it in more details in §1.1.2, following the discussion.

Polynomial Pigeonhole Principle (PPP). As the name suggests, the existence of solution for problems in **PPP** is guaranteed by the pigeonhole principle: *if n pigeons are placed in $m < n$ pigeonholes then at least one pigeonhole must contain more than one pigeon.* The hardness of this class is captured by its canonical complete problem Pigeon, which is described below.

Definition 1 ([84])**.** Pigeon

- *Instance.* A circuit $\mathsf{C} : \{0,1\}^m \to \{0,1\}^m$
- *Solution.* One of the following:
  1. Preimage of 0^m: $x \in \{0,1\}^m$ such that $\mathsf{C}(x) = 0^m$
  2. Collision: $x, y \in \{0,1\}^m$ such that $\mathsf{C}(x) = \mathsf{C}(y)$

An interesting problems that lies inside the class (but is not known to be complete) is EqualSums [4]: given $a_1, \ldots, a_n \in \mathbb{N}$ such that $\sum_{i\in[1,n]} a_i < 2^n - 1$, find two subsets $S \neq T \subseteq [1, n]$ such that $\sum_{i\in S} a_i = \sum_{i\in T} a_i$. Recently, it was shown that

Figure 1.3: Sample LONELY instance with eight vertices $\{0,1\}^3$. The output of the circuit is indicated in (a) by the dotted, directed edges: i.e., an edge (u, v) implies that $\mathsf{C}(x) = y$. The solid, undirected edges are the edges in the implicit (involution) graph shown in (b): i.e. an edge (u, v) implies that $\mathsf{C}(x) = y$ and $\mathsf{C}(y) = x$. The unpaired vertices are in black. Note that the two nodes 010 and 101 remain unpaired because they are violations.

a problem related to Blichfeldt's theorem on lattices (BLICHFELDT) is complete for **PPP** [100]. This is the first *natural* problem that was shown to be complete for the class.[3]

Polynomial Parity Argument (PPA). The class captures computational problems whose totality is rooted in the handshaking lemma for undirected graphs: *every (finite) undirected graph has an even number of vertices of odd degree.* The hardness of this class is captured the problem LONELY described below.[4]

Definition 2 ([84])**.** LONELY

- *Instance.* A circuit $\mathsf{C} : \{0,1\}^m \to \{0,1\}^m$
- *Guarantee.* 0^m is unpaired: $\mathsf{C}(0^m) = 0^m$
- *Solution.* One of the following:
  1. Unpaired vertex: $x \in \{0,1\}^m \setminus \{0^m\}$ such that $\mathsf{C}(x) = x$
  2. Violation: $x, y \in \{0,1\}^m$ such that $\mathsf{C}(x) = y$ but $\mathsf{C}(y) \neq x$

Intuitively, C defines an involution (i.e., a graph with degree at most one) over $\{0,1\}^m$ with (x, y) being an edge if and only if $\mathsf{C}(x) = \mathsf{C}(y)$ but with 0^m unpaired (see Figure 1.3). By handshaking lemma, therefore, there exists another vertex that is paired with itself. Some of the problems that belong to **PPA** include the Borsuk-Ulam theorem from topology (BORSUK-ULAM) and Necklace Splitting [45] from combinatorics. The aforementioned problems are also known to be complete for **PPA**.

Polynomial Local Search (PLS). PLS was defined to capture the difficulty of finding a local optimum in optimization problems such as the Travelling Salesman Problem. The principle that guarantees the existence of such optima is that *every directed*

[3]A problem is considered to be "natural" if its description does not inherently involve a circuit. For instance SPERNER is a natural (combinatorial) problem, whereas PIGEON is not as its definition is tied to a circuit.

[4]We note that problems like LEAF, ODD and EVEN are all polynomial-time equivalent to LONELY (with respect to Karp reductions) and therefore are also complete for **PPA** [6]. Although they are also canonical in nature, their definition relies on a circuit that maps n bits to $O(n)$ or $\mathbf{poly}(n)$ bits. We find the n to n circuit in LONELY to be more minimal and therefore prefer to use it over the others.

acyclic graph (DAG) has a sink. The canonical complete problem for the class is called Local-Search (LS) and is described below [65].

Definition 3 ([65])**.** Local-Search (LS)

- *Instance.*
  1. A successor circuit $\mathsf{S} : \{0,1\}^m \to \{0,1\}^m$
  2. A potential circuit $\mathsf{F} : \{0,1\}^m \to [0, 2^m]$
- *Solution.* Local optimum: $x \in \{0,1\}^m$ such that $\mathsf{F}(x) \geq \mathsf{F}(\mathsf{S}(x))$

Intuitively S and F, together, implicitly define a DAG where (x, y) is an edge if $y = \mathsf{S}(x)$ and $\mathsf{F}(y) \geq \mathsf{F}(x)$, and the goal is to find the sink of this DAG. The well known algorithms for finding a local optimum such as Simplex Algorithm or Gradient Descent can be considered as simple traversals of this graph towards to the sink.

Polynomial Parity Argument on Digraphs (PPAD). **PPAD** is contained in **PPA**$\cap$**PPP** and could be regarded as the *directed* version of the class **PPA**. The class became a subject of intensive study due to its relation to the problem Nash. Papadimitriou showed that Nash belongs to **PPAD** via a reduction to End-of-Line (EOL), its canonical complete problem. A reduction in the opposite direction was later established in a sequence of works [39, 33]. **PPAD** also contains other important problems like computational counterparts of Brouwer and Katukani fix-point theorems as complete. We will see in §1.1.2 why Sperner, the combinatorial problem that we started off our discussion of **TFNP** with, also belongs to this class.

Continuous Local Search (CLS). Roughly speaking, the class **CLS** $\subseteq$ **PLS**$\cap$**PPAD** contains "continuous" variants of the problems in **PLS** [40]. The complete problem for the class is called Continuous-Local-Optimum (CLO), and it is similar to Local-Search but the successor and value circuits are now guaranteed to behave "smoothly" (through Lipschitz continuity). The class is particular interesting to the game theory community as a plethora of games (e.g., Simple Stochastic Games, Mean-Payoff Games) are known to lie inside this class [43]. Another interesting problem that lies in this class (but not known to be complete) is End-of-Metered-Line (EOML) [62]. As the name suggests, EOML is closely-related to EOL and we will exploit this connection to extend all of our results for the class **PPAD** to **CLS** (see §2.3).

Before moving on to **PPAD**, we briefly discuss the remaining subclasses. The class *Polynomial Weak Pigeonhole Principle* (**PWPP**) [64] contains problems related to finding collisions in functions. Although it is contained in **PPP**, the relationship of **PWPP** with the rest of the subclasses is not well-understood. **PPADS** is a more restricted version of **PPAD**. Finally, the subclass *Provable Total Functional Non-deterministic Polynomial-time* (**PTFNP**) was recently introduced with the goal of developing a more unified complexity theory of **TFNP** problems [52]. It contains all of the subclasses that we have discussed above, but the membership of some of the "rogue" **TFNP** problems like Factoring or Ramsey (given a graph of size 2^{2m} find either a clique or an independent set of size m) in **PTFNP** is yet to be established.

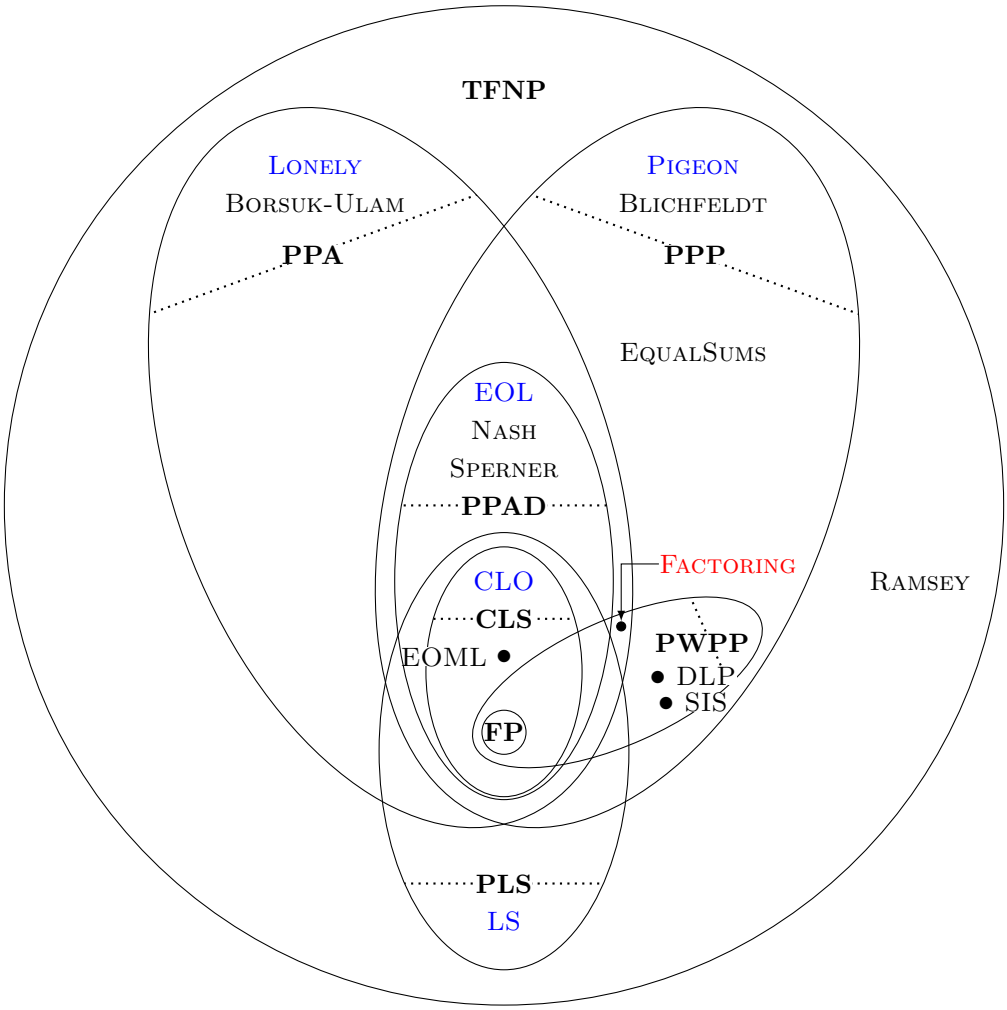


Figure 1.4: Problems in **TFNP**. The canonical complete problems for their classes are in blue. The dotted line is used to separate complete problems from the rest. Factoring is in red to highlight that the inclusion is via a *randomised* Karp reduction.

1.1.2 The Complexity Class PPAD

PPAD can be considered to be the "directed" or "oriented" analogue of the class **PPA** (which makes it easier than **PPA**). The principle that underlies this class is the *directed* formulation of the handshaking lemma: *for any (finite) digraph, if there exists an unbalanced vertex, i.e. a vertex with unequal in- and out-degree, then there exists another unbalanced vertex.* The canonical complete problem for this class goes by the name End-of-Line (EOL): *given a (standard) source in a directed graph where every vertex has both in-degree and out-degree at most one, find a sink or another source* [84]. This 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, as described below.

Definition 4 ([84])**.** End-of-Line (EOL)

- *Instance.*
  1. A successor circuit $\mathsf{S} : \{0,1\}^m \to \{0,1\}^m$
  2. A predecessor circuit $\mathsf{P} : \{0,1\}^m \to \{0,1\}^m$
- *Guarantee.* 0^m is unbalanced: $\mathsf{P}(0^m) = 0^m$ and $\mathsf{S}(0^m) \neq 0^m$
- *Solution.* An unbalanced vertex $v \in \{0,1\}^m$: $\mathsf{P}(\mathsf{S}(v)) \neq v$ or $\mathsf{S}(\mathsf{P}(v)) \neq v \neq 0^m$

To see why totality is not semantic but syntactic, we have to first explain a few technicalities [51]. Note that any badly defined edge, i.e. $\mathsf{S}(u) = v$ and $\mathsf{P}(v) \neq u$ or $\mathsf{P}(v) = u$ and $\mathsf{S}(u) \neq v$, qualifies as a solution of EOL as defined above (because $\mathsf{P}(\mathsf{S}(u)) \neq u$ or $\mathsf{S}(\mathsf{P}(u)) \neq u$ respectively: see Figure 1.5.(a). Note that 0^m is a source of the graph, unless $\mathsf{P}(\mathsf{S}(0^m)) \neq 0^m$, in which case 0^m is a valid solution to the problem as stated above. One could now think of "sanitised" EOL instance $(\mathsf{S}', \mathsf{P}')$ with such badly defined edges turned into self-loops: see Figure 1.5.(b). The circuits S' and P' can intuitively be viewed as succinctly implementing a directed graph of degree at most one 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 $\mathsf{P}'(u) = v$ (see Figure 1.5.(c)). Given that 0^m is unbalanced, the goal is to another unbalanced vertex. Such a vertex must always exist by the handshaking lemma.

It is easy to see why EOL reduces to Pigeon: the reduction simply sets $\mathsf{C}(x) := \mathsf{S}(x)$. Since 0^m doesn't have a preimage in this Pigeon instance, the adversary must return a collision and these correspond to the sink and its predecessor. The reduction from EOL to Lonely, although intuitive, is trickier and can be found, for example, in [6].

Sperner is in PPAD. An obvious way to solve any given EOL instance is to simply follow the standard path, i.e. by starting from the standard source and iteratively applying the successor circuit until one hits a sink. An obvious way to solve any given Sperner instance is to simply follow the walk that we used to establish the existence of a trichromatic triangle. Both approaches could potentially take super-polynomially many steps. At first glance, from the way the problems are solved, it seems that Sperner and End-of-Line share some structural similarities: both problems can be solved by following a well-defined (directed) walk starting from a standard source vertex to its end. We formalise this intuition by showing that Sperner is indeed Karp-reducible to EOL and consequently that Sperner $\in$ **PPAD**.

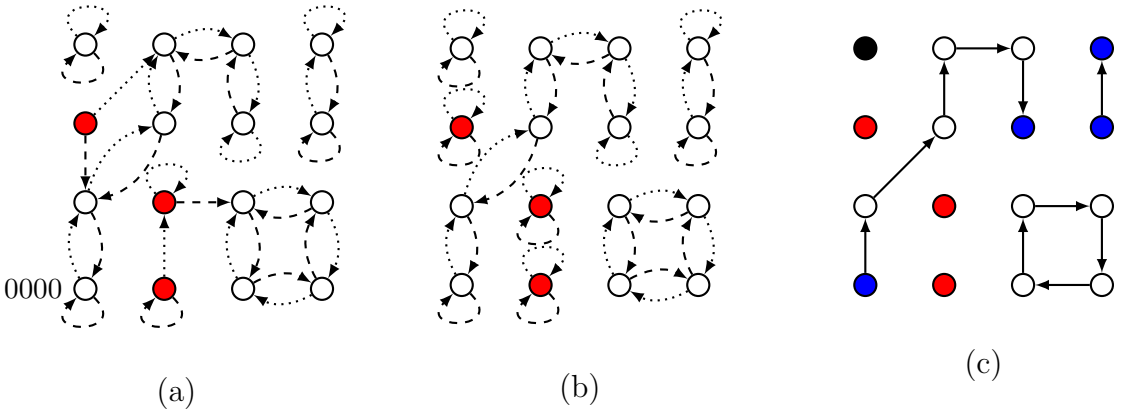


Figure 1.5: Sample End-of-Line instance with 16 vertices $\{0,1\}^4$. The output of the successor (resp., predecessor) circuit is indicated in (a) by the dotted (resp., dashed), directed edges: i.e., a dotted (resp., dashed) edge (u, v) implies that $\mathsf{S}(u) = v$ (resp., $\mathsf{P}(u) = v$). The vertices having badly defined edges are shown in red. The graph in (b) corresponds the "sanitised" EOL instance $(\mathsf{S}', \mathsf{P}')$ where the such edges have been turned into self loops. The solid, directed edges are the edges in the implicit graph shown in (c): i.e. an edge (u, v) implies that $\mathsf{S}(x) = y$ and $\mathsf{P}(y) = x$. The source and sink vertices are shown in blue; self-loops are in black.

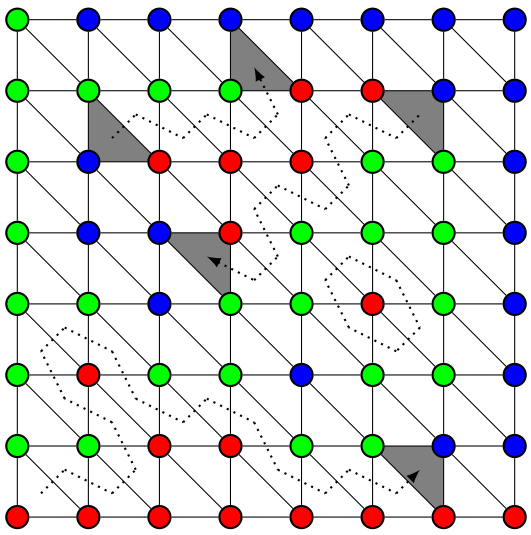


Figure 1.6: Sperner is Karp-reducible to End-of-Line. The End-of-Line instance that is constructed out of the Sperner instance is shown using dotted lines. The source and sink of the EOL instance corresponds to the trichromatic triangles (which are shaded). The triangles which are not the part of any walk are considered to be self-loops.

Theorem 1 ([84]). SPERNER *is Karp-reducible to* END-OF-LINE.

Proof (sketch). The reduction almost follows from the proof for totality of SPERNER that we explained in the introduction. We need an extended version of that argument which establishes the existence of an *odd* number of trichromatic triangles, discussed next. Let us call the walk we used to establish the first trichromatic triangle the standard walk. Consider any trichromatic triangle ΔABC other than the one established by the standard walk: e.g., see Figure 1.6. Suppose for the time being that it has green-red edge with a green vertex on the left. Let us start a walk from ΔABC using the same rules as before. By the same argument as before, this walk can neither exit the grid nor loop with itself. By the same argument that ruled out loops, it can also be established that this walk cannot collide with the standard walk. Therefore, since the grid is finite, the walk halts at another trichromatic triangle. In the complementary case that ΔABC has a green-red edge with a green vertex on a right, we simply perform the walk backwards with the rules reversed. That is the trichromatic triangles are all, except the standard one, paired up.

The actual reduction proceeds as follows. Let the SPERNER instance be presented using the colouring circuit $\mathsf{C} : \{0,1\}^m \times \{0,1\}^m \to \{0,1,2\}$. We associate every triangle ΔABC implicit in this instance with vertex of the EOL instance. (Therefore the size of labels of this EOL instance is $6m$.) In particular, the bottom-left triangle acts as the standard source of the EOL instance (0^{6m}) and the trichromatic triangles, depending on their orientation, will correspond to either a source or a sink. The successor circuit, on input a vertex interpreted as the coordinates of ΔABC, uses C to determine whether it is possible to walk out of this triangle, i.e., if there exists a green-red edge with the green vertex on the left. If so it returns the coordinates of next triangle as its output; otherwise, it returns the coordinates of ΔABC itself. The functioning of the predecessor is symmetrically opposite. Note that the standard walk, used to establish the existence of the first trichromatic triangle, is mapped to the path starting off at the standard source of the EOL instance. In case there exists other (pairs of) trichromatic triangles in the SPERNER instance, these will also result in directed paths: e.g., see Figure 1.6. (Note that cycles are a possibility in this EOL instance as shown in Figure 1.6). It follows that from any unbalanced vertex in the EOL instance, we can recover the coordinates of a trichromatic triangle, completing the reduction. □

1.2 Cryptography and Total Search Problems

Over the years, the research community has striven to classify total search problems into the subclasses of **TFNP**, which has resulted in a rich network of reductions. This, in some sense, reinforces the belief that these problems are hard in the *worst case* – i.e., for every efficient algorithm there exist instances of the problem that are hard to solve.[5] But it could very well be the case that these problems are easy in the *average case*, say, because of some heuristic algorithm that solves *most* instances of the problem. One such example is the

[5]Currently, no **PPAD**-complete problem is known to admit a sub-exponential-time worst-case algorithm. However, 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**) We do however know of worst-case hard instances for a *specific algorithm* for finding a Nash equilibrium [97] or of *oracle-based* worst-case hard instances of **PPAD**-complete problems [58, 3].

Lemke-Howson algorithm, which efficiently solves most instances of NASH that occur in practice [72]. In light of this, it is natural to seek "extrinsic" evidence supporting **TFNP** hardness. That is, as raised in Papadimitriou's original paper [84], whether something can be said about the average-case hardness of total problems. In particular, do there exist efficiently-sampleable *distributions* of instances on search problems that would fail all heuristics?

Since proving average-case hardness *unconditionally* implies that $\mathbf{P} \neq \mathbf{NP}$ we would have to instead reduce from search problems that are assumed to be hard-on-average, i.e. under a *average-case hardness assumption*. One source of such problems is modern cryptography where the ability to sample problems that are hard-on-average is necessary to build even the most basic functionalities like encryption and signatures – problems that are hard in the worst case are simply not sufficient. For instance, the security of one of the components of HTTPS (i.e., Hypertext Transfer Protocol Secure) is under the assumption that FACTORING is hard-on-average. Another example is the widely-deployed DSA (Digital Signature Algorithm) and EC-DSA (Elliptic-Curve DSA) signature schemes which have discrete-logarithm problem (DLP), the problem of computing the discrete logarithm in a prime-order group, at their heart. The main reason to believe that these problems are hard is that even after decades of search for efficient algorithms, they remain elusive.

1.2.1 Average-Case Hardness in TFNP

The aforementioned cryptographic hard problems, FACTORING and DLP, are both total and therefore imply average-case hardness in **TFNP**. Since we strive for minimality, the question that naturally follows is whether these problems also imply hardness in any of its lower subclasses. The benefits of such a result would be two-fold. Firstly, it would rule out heuristics for the subclass. To be more precise, it would give us a way to sample instances of the problem such that *any* efficient algorithm would fail to find a solution to the problem. For example, establishing average-case hardness in **PPAD** would allow us to sample instances of NASH that defeats the Lemke-Howson algorithm. Secondly, it would shed more light on the structure of that hard problem. This is interesting from the point of view of a cryptographer as it would have potential implications to cryptanalysis of the hard problem.

Average-case hardness under standard cryptographic assumptions was long known for some of the higher subclasses of **TFNP**. The most notable results are the following.

1. One-way permutations (OWPs) implies hardness of **PPP** [84]. Let (π, y^*) denote an instance of OWP, where $\pi : \{0,1\}^m \to \{0,1\}^m$ is a circuit evaluating the OWP and $y^* \in \{0,1\}^m$ is the challenge that is supposed to be inverted. The Karp reduction maps (π, y^*) to the PIGEON circuit

$$\mathsf{C}_{y^*}(x) := \pi(x) \oplus y^*$$

where $\oplus$ denotes the bitwise XOR operation. Since π is a permutation, this PIGEON instance does not have any collision. Therefore the only solution is the preimage of 0^m, i.e., $x^* \in \{0,1\}^m$ such that $\pi(x^*) = y^*$, which is the solution to the OWP instance.

2. Hardness of **PWPP** from collision-resistant hash functions (CRHFs) follows trivially. It is folklore that hardness of **PPP** also follows from CRHFs. Given a CRHF $H : \{0,1\}^m \to \{0,1\}^{m-1}$ that shrinks the input by one bit, the Karp reduction maps it to the PIGEON circuit

$$\mathsf{C}(x) := 1 \| H(x),$$

where $\|$ denotes the string-concatenation operator. Since this instance does not map any element to 0^m, the only solutions are collisions in the circuit and each such collision yields a collision to the CRHF. Since problems like FACTORING, DLP, SIS (i.e., the short integer solution problem on lattices [2]) imply CRHFs, it follows that they also give rise to **PPP**-hardness.

3. FACTORING implies hardness of **PPA** albeit via *randomised* reductions [25, 64]. Given an instance $N \in \mathbb{N}$ of FACTORING the reduction, *roughly speaking*, defines the LONELY instance as

$$\mathsf{C}(x) := x^{-1} \bmod N,$$

with $1 \in \mathbb{Z}_N^*$ set to be the trivial unpaired vertex. Since one of the other unpaired vertices in this instance is an element $x \neq 1 \in \mathbb{Z}_N^*$ such that $x^2 = 1 \bmod N$, one of the factors of N can be extracted by computing $\mathrm{GCD}(x-1, N)$.

Other results are known regarding average-case hardness in **TFNP**. Hubáček, Naor and Yogev [61] constructed hard **TFNP** problems from one-way functions (or, in fact from any average-case hard **NP** *language*) under complexity-theoretic assumptions used in the context of derandomization. As a consequence, **TFNP** is hard-on-average in Pessiland [63] (where hard-on-average languages exist but one-way functions don't) under the aforementioned complexity-theoretic assumptions. This result was recently strengthened in [86] to show unconditional average-case hardness of **TFNP** in Pessiland. However, it is not known whether these distributions gives rise to average-case hardness in any of the syntactic subclasses of **TFNP**. Komargodski, Naor and Yogev [69] demonstrated a close connection between the existence of collision-resistant hashing and the Ramsey problem (RAMSEY). RAMSEY is not known to lie inside any of the syntactic subclasses of **TFNP** though (see Figure 1.4).

1.2.1.1 Barriers to Average-Case Hardness

The relatively small progress on showing average-case hardness of total search problems from weak general assumptions (like one-way functions, which is very unstructured) motivated a line of works focusing on limits for proving average-case hardness. The implausibility of using worst-case **NP** hardness [65, 78] was later strengthened to show that it is unlikely to base average-case **TFNP** hardness even on problems in the polynomial hierarchy [24], 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 [77]. A recent result [95] 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 a small number of solutions.

1.2.2 Average-Case Hardness in PPAD

In the previous section we saw how certain cryptographic primitives impart hardness to some of the subclasses of **TFNP** that are higher up. It is currently not known whether any of these results can be extended to lower subclasses like **PPAD** or **CLS**. Neither was it not known if their average-case hardness could be based on other cryptographic primitives like fully-homomorphic encryption. This was the state of affairs until a series of results established a connection between a newly-introduced cryptographic primitive known as program obfuscation and the hardness of **PPAD**.

Loosely speaking, program obfuscation allows us to efficiently "scramble" programs *without* losing its functionality [5]. That is, given a program P represented in an appropriate model of computation (say, Turing machines or Boolean circuits), the obfusctor returns its scrambled version $\widehat{\mathsf{P}}$ such that $\mathsf{P}(x) = \widehat{\mathsf{P}}(x)$ on all inputs x (completeness). There are various notions of soundness, the strongest being that of *virtual black-box* (VBB) obfuscation: the obfuscated program $\widehat{\mathsf{P}}$ should not reveal any information on P other than its input-output behaviour. It was shown in [5] that this notion is not attainable in general, and they suggested weakening the soundness requirement to *indistinguishability obfuscation* (IO): the obfuscations of two functionally-equivalent programs should be indistinguishable. It turned out that this weakened notion is still strong enough to obtain interesting, non-trivial results (such as public-key encryption from private-key encryption) [96]. Moreover, many candidate construction were proposed, albeit from new, non-standard hardness assumptions [47].

PPAD-hardness via obfuscation. The connection between obfusction and **PPAD**-hardness was first observed in [1]. They showed how to obtain hard distribution of instances of the End-of-Line problem using VBB obfuscation. On a high level construction can be divided into two steps (see §2.2 for a detailed explanation):

1. construct hard distribution of a *promise* problem called Sink-of-Verifiable-Line (SVL) using VBB obfuscation and one-way functions; and
2. given this SVL instance, simulate End-of-Line using reversible black pebbling [8].

Bitansky, Paneth and Rosen [13], in their breakthrough paper, demonstrated how the first step above can be carried out using sub-exponentially-secure IO (i.e., the soundness should hold with respect to adversaries that run in time sub-exponential in the security parameter). This gave the first extrinsic evidence of **PPAD** hardness and provided a plausible method to sample potentially hard-on-average End-of-Line instances using the candidate for IO from [47]. The result in [13] was extended by Hubáček and Yogev [62], who observed that the second step in the above construction essentially, for free, yields instances the End-of-Metered-Line problem, which lies in the class $\mathbf{CLS} \subseteq \mathbf{PLS} \cap \mathbf{PPAD}$. Both results were subsequently strengthened by relaxing the underlying assumptions:

1. Garg, Pandey and Srinivasan [48] reduced from breaking IO with polynomial (instead of sub-exponential) hardness (or alternatively compact public-key functional encryption) and one-way permutations;
2. Komargodski and Segev [70] reduced from breaking quasi-polynomially secure private-key functional encryption and sub-exponentially-secure injective one-way functions.

1.3 Overview of the Thesis

In one way or another, all of the above assumptions used to construct hard instances of **PPAD** 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. In this thesis, we take a step in this direction by showing **PPAD**-hardness under assumptions that are of a different flavour from obfuscation (and arguably weaker). We give a summary of our results in §1.3.1 and then in §1.3.2 provide a high-level overview of the techniques used to obtain these results. Our results demonstrates new ways for sampling hard-on-average **PPAD** instances, based on assumptions of seemingly different nature than those required by prior work (e.g., number-theoretic, in contrast to ones related to obfuscation).

1.3.1 Main Results

In this thesis, we present two constructions of hard distribution of End-of-Line. The first construction relies on the (average-case) hardness of a number-theoretic search problem that we call Iterated-Squaring, and requires a random oracle, i.e. a random function that is accessible to all parties [7]. The second construction relaxes these assumptions to a considerable degree. On a high level, both the constructions deploy the same technique, i.e., *incrementally-verifiable computation via recursive proof-merging*. In particular, the second construction can be considered to be a strengthening of the first. We next state, informally, the two main theorems in this thesis.

1.3.1.1 PPAD-Hardness from Iterated-Squaring

Our first construction is based on the hardness of Iterated-Squaring (IS), a number-theoretic search problem related to Factoring, described below.

Definition 5 ([26, 94])**.** Iterated-Squaring (IS)

- *Instance.*
  1. Modulus $N \in \mathbb{N}$
  2. Group element $x \in \mathbb{Z}_N^*$
  3. Time parameter $T \in \mathbb{N}$
- *Solution.* $f(N, x, T) = x^{2^T} \bmod N$

To generate a hard distribution of IS instances, N is chosen as the product of two random $\lambda/2$-bit safe primes[6] and x is sampled uniformly at random from $\mathbb{Z}_N^*$. Computing f for moduli chosen as above was suggested as a hard problem by Rivest, Shamir and Wagner[94], who conjectured that for any T, computing f either requires $\Omega(T)$ *sequential* time or total computation sufficient to factor N. The latter requirement is in place as IS

[6] A prime of the form $p = 2p' + 1$ is a safe prime if p' is also a prime; conversely, p' is a Sophie Germain prime if $2p' + 1$ is also a prime.

is at most as hard as FACTORING. To be more precise, the knowledge of the factors of N allows computing f using a standard shortcut (see §3.1.1) Our hardness assumption on IS is milder: it is sufficient for us that the solution $f(N, x, T)$ cannot be computed in time $\mathbf{poly}(\lambda)$ for some (potentially exponentially large) T (see Assumption 2 in §3.1.2). Our reduction also requires assuming access to a random oracle, which is used in the context of transforming a $\log T$-round public-coin interactive proof from [88] into a non-interactive one (see the discussion below). The reduction can be informally stated as follows.

Theorem (informal)**.** *For a security parameter $\lambda \in \mathbb{N}$, let N be a modulus sampled as described above. If there exists a time parameter $T = T(\lambda) \in \left[\omega(\mathbf{poly}(\lambda)), 2^{\lambda}\right]$ such that no $\mathbf{poly}(\lambda)$-time algorithm can solve the* ITERATED-SQUARING *instance (N, x, T) with non-negligible probability, then there exists a hard distribution of* END-OF-LINE *instances.*

1.3.1.2 PPAD-Hardness from Fiat-Shamir Methodology

In our second construction, we relax the hardness assumption from ITERATED-SQUARING to the mild complexity-theoretic assumption that #SAT, the problem of counting the *number* of satisfying assignments to a CNF formula, is hard in the *worst case*. In addition we rely on the so-called Fiat-Shamir methodology [44], a technique used to transform a public-coin interactive protocol into a non-interactive protocol, which we explain next.

Recall that in an interactive protocol a prover (which is usually unbounded) tries to convince a computationally-bounded verifier of the validity of a statement, e.g. the number of satisfying assignments to a SAT instance. The protocol is usually executed over several rounds, over which the prover and verifier exchange messages with each other as shown in Figure 1.7.(a). The verifier is probabilistic and if the random coins used to sample its messages can be public, we say that the protocol is *public-coin* (and otherwise it is secret-coin). The protocol is said the be *sound* if it is hard for the prover to convince the verifier of a false statement, e.g. of wrong number of satisfying assignments to a SAT instance.

In the Fiat-Shamir methodology, a public-coin interactive protocol is compiled into a non-interactive protocol by, loosely speaking, replacing the verifier in the public-coin protocol with a hash function H sampled randomly from a family of hash functions $\mathcal{H}$. This is carried out as follows: in each round $i \in [1, \rho]$, instead of obtaining the message from the verifier, the prover computes it itself by hashing the transcript of the protocol (so far) using H. As a result, in the non-interactive version of the protocol, the prover simply simulates the verifier by computing the hash internally and then only sending the last message over as shown in Figure 1.7.(b). Given the original interactive protocol is sound, we say that the Fiat-Shamir Transform maintains soundness (or simply, is sound) if the non-interactive protocol that results by applying the transform is also sound.

For our construction, we require the Fiat-Shamir Transform to maintain soundness for the Sumcheck Protocol, an interactive protocol that was used in [75] to show that #SAT $\in$ **IP**. That is, it allows a prover to convince a verifier of the number of satisfying assignments to a SAT instance. The protocol runs in n rounds, where n denotes the number of variables in the #SAT instance. Since breaking the soundness of Fiat-Shamir is reducible to #SAT (in fact to SAT) it follows that efficiently solving the above distribution is no easier than breaking Fiat-Shamir.

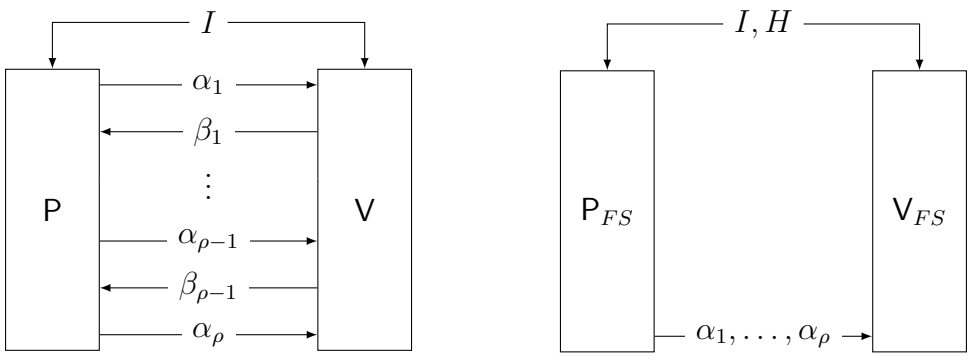


Figure 1.7: The Fiat-Shamir Transform. (a) An ρ-round interactive protocol between a prover P and a verifier V. I denotes the statement that the prover is trying to convince the verifier about. The i-th message from the prover to the verifier is denoted by α_i; the i-th message in the opposite direction is denoted by β_i. (b) The non-interactive protocol as a result of the transform. The i-th message from the verifier β_i is now computed by the prover (by itself) as the hash of the transcript (with respect to V) up to that point, i.e. $H(I, \alpha_1, \beta_1, \ldots, \alpha_{i-1})$. Therefore only the prover messages $\alpha_1, \ldots, \alpha_\rho$ are now required to be sent.

Theorem (informal)**.** *Solving the* End-of-Line *problem is at least as hard as breaking the (adaptive) soundness of the Fiat-Shamir Transformation, when applied to the Sum-check Protocol.*

Moreover, we show that the soundness indeed holds when H is instantiated with a random oracle.[7] Therefore, we get the following theorem and its corollary.

Corollary (informal)**.** *If* #SAT *is hard (in the worst case) then, relative to a random oracle, there exists a hard distribution of* End-of-Line *instances.*[8]

Two remarks are in order. Firstly, there is growing evidence that it might be possible to instantiate the Fiat-Shamir using hash functions that are constructed from standard assumption [27]. Therefore, it is plausible that the hardness of our construction is based an object other than the random oracle. We show in §4.4 that this is indeed that case under some strong assumptions on fully-homomorphic encryption (FHE). We defer this discussion to §4.1.1 in Chapter 4. Secondly, all of the above results (including the above instantiation using FHE) can be strengthened to achieve hardness in $\mathbf{CLS} \subseteq \mathbf{PLS} \cap \mathbf{PPAD}$ by applying the observations in [62]. We discuss more on this in §2.3 in the next chapter.

1.3.2 Techniques

In both constructions, we follow the blueprint from [1, 13] that we briefly discussed in §1.2.2. That is, we proceed via the following two steps:

[7] In fact, the usage of random oracle in the previous result can be thought of applying a variant of the Fiat-Shamir Transform to the protocol from [88].

[8] In the original paper [34], we had claimed that the corollary holds just relative to a random oracle, without the additional assumption on #SAT. However, this is not correct: see Remark 12.

1. construct hard distribution of the Sink-of-Verifiable-Line (SVL) problem; and
2. given an SVL instance from the above distribution, simulate End-of-Line using reversible black pebbling.

Since the second step is more or less similar to previous constructions, in this overview we focus on the first step. In particular, we explain it from the point of view of the first construction (from Iterated-Squaring) since it is conceptually simpler to explain. However, it still suffices to demonstrate the essential techniques that underlies both constructions (viz. incremental computation, recursive proof-merging, unique proofs). We defer the overview of the second construction to Chapter 4 and limit ourselves here to highlighting its differences from the first (at the end of the section). Since SVL is central to these constructions, first we formally define it the section below and explain its relationship with the notion of incrementally-verifiable computation [103].

1.3.2.1 Sink-of-Verifiable-Line

Definition 6 ([103, 13])**.** Sink-of-Verifiable-Line (SVL)

- *Instance.*
  1. A successor circuit $\mathsf{S} : \{0,1\}^m \to \{0,1\}^m$
  2. A verifier circuit $\mathsf{V} : \{0,1\}^m \times [1, 2^m] \to \{0,1\}$
  3. Length $L \in [1, 2^m]$
- *Promise.* For every $v \in \{0,1\}^m$ and $i \in [1, 2^m]$, $\mathsf{V}(v, i) = 1$ if and only if:
  1. $i \leq L$; and
  2. $v = \mathsf{S}^i(0^m)$.
- *Solution.* Sink vertex: $v \in \{0,1\}^m$ such that $\mathsf{V}(v, L) = 1$

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 path, the "standard path", starting at 0^m. The circuit V enables to efficiently test whether a given vertex v is of distance i from 0^m on this standard path, and the goal is to find the vertex that lies at a distance L from 0^m, the standard sink. Note that not every tuple $(\mathsf{S}, \mathsf{V}, L)$ is a *valid* SVL instance since V might not satisfy the promise about its behaviour: see Footnote 2. Moreover, there may not be an efficient algorithm for verifying whether a given tuple $(\mathsf{S}, \mathsf{V}, L)$ is a valid instance, hence this problem lies outside of **TFNP**.

Relationship with incrementally-verifiable computation (IVC). Consider a party A carrying out a time-intensive (deterministic) computation C like, e.g., testing whether a number M_p is a Mersenne prime (which takes months for the primes being presently tested). There could arise scenarios where A must discontinue computing C but would like to hand over the intermediate state to another party B who then continues the computation. However B might be distrustful of A and the purpose of IVC is to help bridge this distrust.

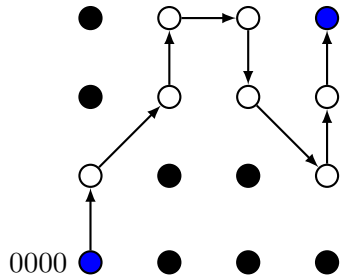


Figure 1.8: Sample Sink-of-Verifiable-Line instance with 16 vertices (i.e., $m = 4$) with a standard path of length $L = 8$. The output of the successor circuit is indicated by the directed edges: i.e., an edge (u, v) implies that $\mathsf{S}(u) = v$. The source and standard sink vertex are in blue. The vertices that lie off the standard path, thus rejected by the verifier, are in black.

Suppose that on input I, the computation of C goes through the sequence of configurations

$$C_0 \to \ldots \to C_T$$

for some (large) $T \in \mathbb{N}$. Instead of computing C plainly as above, the idea in IVC is to carry it out *verifiably* using an alternative program $\widehat{\mathsf{C}}$ which, to every intermediate configuration C_i, attaches a succinct proof $\pi_i := \pi(C_0 \xrightarrow{i} C_i)$ that attests to the fact that C_i is obtained from C_0 in i steps.[9] Therefore the sequence of configurations for the IVC would look like:

$$(C_0, \pi_0) \to \ldots \to (C_T, \pi_T). \tag{1.2}$$

Any party should be able to use π_i to efficiently verify that the C_i is indeed the i-th step in the computation. Moreover, given any intermediate configuration (C_i, π_i) of $\widehat{\mathsf{C}}$, computing the next configuration (C_{i+1}, π_{i+1}) should be *incremental*, i.e. takes time comparable to computing C_{i+1} from C_i. Together, the incremental and verifiable nature of $\widehat{\mathsf{C}}$, enables A to halt at any intermediate stage of the computation and B to take over from that point onward.

It is not difficult to see how IVC could potentially be used to design a hard SVL instance. We start off by setting the standard path in the SVL instance as the sequence of configurations in eq.(1.2). Given a vertex (C_i, π_i), the successor circuit uses the *incremental* property of IVC to (efficiently) compute (C_{i+1}, π_{i+1}) and returns it as the next vertex. The verifier circuit, on the other hand, simply invokes the *verifiable* property of IVC. Although there are several issues that needs to be addressed with the above approach (which we do in §1.3.2.3), we see next in our first construction how some of the ideas indeed turn out quite useful.

1.3.2.2 Sink-of-Verifiable-Line from Iterated-Squaring

To explain our construction, we rely on a slightly different formulation of SVL where the standard source can be any arbitrary vertex $v_0 \in \{0,1\}^m$. This formulation can be shown to be equivalent to Definition 6: see Lemma 1 and Remark 3.

[9] By succinct, we mean the size of the proof should be (almost) independent of the run-time T). Note that IVC can be trivially achieved without the succinctness requirement. For example, $\widehat{\mathsf{C}}$ could simply maintain all the previous configurations in the proof: i.e., $\pi_i := C_0, \ldots, C_{i-1}$.

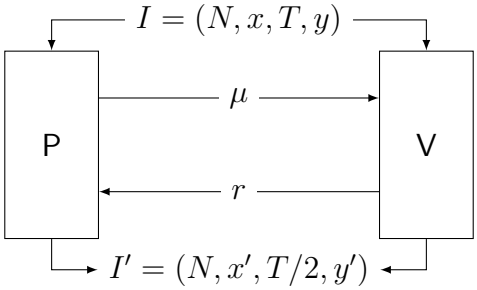


Figure 1.9: The halving sub-protocol between a prover P and a verifier V for $T > 2$ results in a statement I turned into a new statement I'.

Consider the following natural approach for reducing the computation of $f(N, x, T)$ to an instance of SVL of length $L = T$ and vertices of size λ (i.e. $m = \lambda$). 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 standard path:

$$x \to x^2 \to x^{2^2} \to x^{2^3} \cdots \to x^{2^T} \pmod{N}. \tag{1.3}$$

Notice that, assuming $x^{2^T} \bmod N$ cannot be computed in time $\mathbf{poly}(\lambda)$ for a sufficiently large T, 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 [88].

Pietrzak's proof system. Pietrzak's protocol allows a prover to convince a verifier that a tuple $I = (N, x, T = 2^t, y)$ satisfies the relation $y = x^{2^T} \bmod N$ – i.e., it is an interactive proof system for the *decision* problem corresponding to IS. Moreover, remarkably, it does not require either prover or verifier to know the factorization of N. The protocol is recursive in the time parameter T and invovles $t = \log T$ rounds of interaction. 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 $r \leftarrow \{0,1\}^\lambda$ (interpreted as an integer), the verifier and prover merge these two statements into a single statement by computing a random linear combination $x' := x^r \cdot \mu \bmod N$ and $y' := \mu^r \cdot y \bmod N$ and setting $I' = (N, x, T = 2^t, y)$ as the new statement. The above constitutes the "halving" sub-protocol, which is illustrated in Figure 1.9. 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 halving sub-protocol 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 Transform. 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 $\pi(x \xrightarrow{T} y)$. We point out the following three crucial properties of the protocol where λ, if you recall, denotes the size of N in binary representation:

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

An attempt using *efficient* proof-merging. 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 standard path described in eq.(1.3). A first attempt would be to simply augment the vertex labels $x_i = x^{2^i} \bmod N$ in eq.(1.3) with a corresponding proof, i.e., consider the standard path

$$\pi(x_0 \xrightarrow{0} x_0) \to \pi(x_0 \xrightarrow{1} x_1) \to \pi(x_0 \xrightarrow{2} x_2) \to \cdots \to \pi(x_0 \xrightarrow{T} x_T),$$

where the circuit V simply runs the efficient proof verification algorithm of Pietrzak's protocol. This change renders the standard path efficiently verifiable. However, it is now not entirely clear how to implement the successor circuit S efficiently. Since the labels now comprise of proofs, and S is consequently required to efficiently "update" a proof $\pi(x_0 \xrightarrow{i} x_i)$ to $\pi(x_0 \xrightarrow{i+1} x_{i+1})$. We do not know how to implement this for Pietrzak's proof system. To overcome this issue, we use the ability to "merge" proofs, in the sense that given proofs $\pi(x \xrightarrow{\ell} y), \pi(y \xrightarrow{\ell} z)$ one can efficiently compute a single proof $\pi(x \xrightarrow{2\ell} z)$.

Given the ability to merge proofs, we can construct a valid SVL instance by considering a standard path 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 $\pi(x_i \xrightarrow{1} x_{i+1})$, 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 $\pi(a \xrightarrow{\ell} b), \pi(b \xrightarrow{\ell} c)$ merge them into $\pi(a \xrightarrow{2\ell} c)$). This results in a standard path where the first few vertices are:

$$\pi(x_0 \xrightarrow{0} x_0) \to \pi(x_0 \xrightarrow{1} x_1) \to \pi(x_0 \xrightarrow{2} x_2) \to \pi(x_0 \xrightarrow{2} x_2) \| \pi(x_2 \xrightarrow{1} x_3) \to$$
$$\pi(x_0 \xrightarrow{4} x_4) \to \pi(x_0 \xrightarrow{4} x_4) \| \pi(x_4 \xrightarrow{1} x_5) \to \cdots$$

where crucially the number of proofs contained in each label always remains below $\log T$. We remark that this construction is conceptually similar to Valiant's construction of general-purpose incrementally-verifiable computation. We provide a detailed comparison below in §1.3.2.4.

Strictly speaking, Pietrzak's proof system does not support efficient merging of proofs as outlined above. However, it does support somewhat efficient proof-merging as in Property 3. Our key observation is that this somewhat-efficient merging is already sufficient,

as explained next, to construct a valid SVL instance where *both* the successor circuit S and verification circuit V run in $\mathbf{poly}(n)$ time.

Our construction via *recursive* proof-merging. 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 $\pi(x \xrightarrow{\ell} y)$ 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 $\pi(x \xrightarrow{2\ell} z)$ making $3 \cdot L(\ell)$ invocations of S. The idea is to first compute $\pi(x \xrightarrow{\ell} y)$ in $L(\ell)$ steps, then $\pi(y \xrightarrow{\ell} z)$ in another $L(\ell)$ steps (while keeping the first proof $\pi(x \xrightarrow{\ell} y)$ around in the label), and finally using another $L(\ell)$ steps to *recursively merge* those two proofs into $\pi(x \xrightarrow{2\ell} z)$ 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. The above (recursive) construction can be thought of as an *incrementally-verifiable* procedure [103] to compute $x^{2^T} \bmod N$: incremental in the sense that in each step we make some progress towards the goal of computing $x^{2^T} \bmod N$ and verifiable in the sense that each step of the computation can be validated.

Comparison with the second construction. The second construction can be viewed as an alternative instantiation of the ideas explained above with two main differences concerning the underlying assumptions. First, the underlying computational problem — and therefore the hardness assumption — is different: in the second construction we switch to the (worst-case) hardness of $\#\mathbf{P}$, which is weaker than our assumption on Iterated-Squaring. Therefore, secondly, the interactive protocol underlying the second construction needs to be switched accordingly to the Sumcheck Protocol. As we will explain later, this renders the second construction a significant strengthening of the first (with respect to the assumptions).

1.3.2.3 Dealing with Imperfect Verifier

One important detail that we have totally glossed over in the above description is that Pietrzak's proof system is not "perfect" and therefore neither is the verifier which is built on top of this proof system. Since the verifier is not perfect, the SVL instance that we obtain above does not quite meet the requirements laid down in the definition of SVL (Definition 6). In particular, there are two issues that are inherited from the proof system:

1. Since the proof system does not have perfect soundness, the verifier may accept vertices corresponding to wrong proofs as lying on the standard path.
2. It could very well be that there are multiple correct proofs for the same statement. That is the proofs might not be *unique* and therefore the proof systems might be *unambiguous*.

Note that these same issues affect the construction of SVL from incrementally-verifiable computation outlined in §1.3.2.1. We deal with this issue in two steps.

1. First, we define an imperfect version of the SVL problem. To be more precise, we relax the promise of SVL to accept vertices that lie off the standard path and

then add these off-the-path vertices as solutions to the instance. As long as these additional solutions are hard to find, breaking the SVL instance should still be tantamount to finding the sink of the standard path. We remark that this almost holds for the above proof system: it is *statistically* sound but *does not* have unique proofs. Fortunately though by slightly changing the algebraic setting we *can* obtain a proof system with *statistically-unique* proofs. We will formally define the new problem, which we call Relaxed-Sink-of-Verifiable-Line (RSVL), in the next chapter (§2.4).

2. Next, we show that RSVL is (randomised) Karp-reducible to EOL by carefully arguing that the reduction from SVL to EOL still goes through despite the introduction of additional off-the-path solutions.

1.3.2.4 Parallels with Valiant's Construction

In [103], Valiant describes a compiler that allows carrying out *any* given (even exponentially-long) computation in an incrementally-verifiable manner. Our attempt in §1.3.2.2 at constructing SVL instances assuming efficient merging of proofs is inspired by his approach, which we describe next. For $T \in \mathbb{N}$, let

$$\mathsf{C} = C_0 \to \ldots \to C_T$$

denote the computation that we would like to carry out in an incrementally-verifiable manner. Let us suppose there exists a (non-interactive) proof system that allows us to prove statements of the form $C_i \xrightarrow{\ell} C_j$, i.e. C_j is obtained from C_i in ℓ steps of computation. Further, let's assume that the proofs system supports efficient merging of proofs in the following sense: given two proofs $\pi(C_i \xrightarrow{\ell} C_j)$ and $\pi(C_j \xrightarrow{\ell} C_k)$, we can *efficiently* compute a single proof $\pi(C_i \xrightarrow{2\ell} C_k)$ that is of the *same size* as the original ones.

Given the ability to merge proofs, the incrementally-verifiable computation (IVC) of C can be described inductively as follows. To increment the computation by one step from the i-th configuration to the $i+1$-th configuration, we augment the configuration (now consisting of multiple "partial" proofs) with a proof for the single step $\pi(C_i \xrightarrow{1} C_{i+1})$, 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 $\pi(C_i \xrightarrow{\ell} C_j)$ and $\pi(C_j \xrightarrow{\ell} C_k)$ then merge them into $\pi(C_i \xrightarrow{2\ell} C_k)$. This results in a sequence of configurations (with the original configurations dropped for the sake of space):

$$\pi(C_0 \xrightarrow{0} C_0) \to \pi(C_0 \xrightarrow{1} C_1) \to \pi(C_0 \xrightarrow{2} C_2) \to \pi(C_0 \xrightarrow{2} C_2) \| \pi(C_2 \xrightarrow{1} C_3) \to$$
$$\pi(C_0 \xrightarrow{4} C_4) \to \cdots \to \pi(C_0 \xrightarrow{T/2} C_{T/2}) \| \ldots \| \pi(C_{T-2} \xrightarrow{1} C_{T-1}) \to \pi(C_0 \xrightarrow{T} C_T).$$

Note that, crucially, the number of proofs contained in each label always remains below $\log T$. Therefore, if the proof system is succinct, then the blow-up in the size of the configurations is polynomial (assuming T is at most exponential in the size of the input).

The main contribution in [103] was constructing a proof system that allows efficient merging using a technique called *recursive proof-composition*. The construction in [103] deployed strong non-interactive CS proofs of knowledge as the underlying proof system (also known as SNARKs.), with very efficient (e.g. linear-time) knowledge extractors required to enable recursive proof-composition. Constructing such proof systems under

more standard (falsifiable) assumptions is a notoriously hard proposition [10, 50]. Constructions usually rely on knowledge assumptions or are presented in the random oracle model. Our assumptions, on the other hand, are comparatively milder. For our second construction, we only assume standard (adaptive) soundness of the concrete and natural cryptographic protocol obtained by applying the Fiat-Shamir Transform to the interactive Sumcheck Protocol. One consequence of this distinction is that we prove our construction is sound in the random oracle model, whereas no such proof is known for Valiant's construction.[10]

We note, moreover, that in Valiant's construction the proofs are not unambiguous, and thus it is not clear how to use his scheme to obtain hard instances in **PPAD**.[11]

1.3.3 Organisation

In Chapter 2, we provide the prerequisite formal definitions (§2.1) and discuss in detail some of the previous works that serve as bases for the results in Chapters 3 and 4. In particular, we focus on the results in [13] and [62] which establish hardness in **PPAD** (§2.2) and **CLS** (§2.3), respectively, using indistinguishability obfuscation. We end the chapter with a section (§2.4) on the Relaxed-Sink-of-Verifiable-Line (RSVL) problem, which will serve as our gateway to **PPAD**/**CLS** hardness in the subsequent chapters.

In Chapter 3, we present the first of the two constructions of hard distributions of the RSVL problem. The hardness assumption on Iterated-Squaring along with its relevant background is described in §3.1. Our construction is inspired by an interactive protocol from [88], which is described in detail in §3.2. The construction itself is finally given in §3.3. There we will see how *recursive proof-merging* works and we regard this to be the main technical novelty in this thesis. Most of the content in this chapter is from [36].

In Chapter 4, we present the second construction of hard distributions of the RSVL problem. The construction is based on the classical Sumcheck Protocol from [75], which is described in detail in §4.2. The hardness of the construction is based on the soundness of the Fiat-Shamir Transform for the Sumcheck Protocol. We formulate this precisely in §4.2, and also show that it holds in the random-oracle model. The construction itself (§4.3) also relies on recursive proof-merging and is on a high level similar to the one in the previous chapter. Finally in §4.4, using ideas from [27], we show that the assumption on the soundness of the Fiat-Shamir Transform holds under strong assumptions on fully-homomorphic encryption. Most of the content in this chapter is from [34] and its full version [35].

We conclude with Chapter 5, where we discuss some of the avenues that could be further explored. We also briefly mention some of the concurrent works regarding **PPAD**-hardness that are pertinent to this thesis.

[10] The issue is that Valiant's proof system cannot be composed to prove statements about a non-explicit oracle.

[11] A key ingredient in Valiant's construction is a CS proof [79] obtained via a Merkel Hash applied to a PCP. This is not unambiguous because small changes to a correct PCP string will change the proof, but will only be noticed by the verifier with small probability. A recent work [80] has managed to construct incrementally-verifiable PCPs, but their construction is not unambiguous either.

Chapter 2
Background

In this chapter, we explain the previous works that serve as bases for the results in Chapters 3 and 4. In particular, we focus on the results in [13] and [62], which use indistinguishability obfuscation (IO) to establish hardness in **PPAD** (§2.2) and **CLS** (§2.3), respectively. We end the chapter with a section on a relaxed version of Sink-of-Verifiable-Line (§2.4) that will serve as our gateway to **PPAD** and **CLS** hardness. But first we fix the notation and recall some prerequisite formal definitions in §2.1.

2.1 Definitions

2.1.1 Notation

- Throughout, we use $[a, b]$ to denote $\{a, a+1, \ldots, b-1, b\}$, the sequence of integers from a to b (both inclusive).
- We use straight font to denote algorithms, circuits and protocols (e.g., A, C, PP), calligraphic font to denote sets (e.g., $\mathcal{I}$,$\mathcal{H}$), bold face to denote complexity classes (e.g., **P**, **NP**) or vectors (e.g., $\boldsymbol{v}$, $\boldsymbol{\mu}$), small caps to denote problems or languages (e.g., Factoring, SVL). Polynomials, functions and events are in normal math mode (e.g., $p(n)$, *trace*, *bad*).
- For a variable n, by $\mathbf{poly}(n)$ we refer to the set of all polynomials (over integers) in n. Whenever we say some quantity is in $\mathbf{poly}(n)$ (e.g., the running time of an algorithm), we mean that there exists a polynomial $p(n) \in \mathbf{poly}(n)$ defining that quantity. The definition of $\mathbf{poly}(n)$ is naturally extended to more than one variables. These conventions apply to the sets $\mathbf{quasipoly}(n)$ (quasi-polynomials) and $\mathbf{polylog}(n)$ (polynomials in $\log(n)$).
- We refer to an algorithm (resp., circuit), which can be probabilistic, as "efficient" if its running time (resp., size) is $\mathbf{poly}(n)$, where n denotes the size of its input.
- A function $\epsilon : \mathbb{N} \to [0, 1]$ is *negligible* if for every polynomial $p(n) \in \mathbf{poly}(n)$ there exists an $n_0 \in \mathbb{N}$ such that $\epsilon(n) \leq 1/p(n)$ for all $n \geq n_0$; $\mathbf{negl}(n)$ denotes the set of all negligible functions in n. If $\epsilon(\cdot)$ is not negligible, we say it is *non-negligible*; whereas $\epsilon(\cdot)$ is *overwhelming* if $1 - \epsilon$ is negligible. Finally, $\epsilon : \mathbb{N} \to [0, 1]$ is *noticeable* if there exist $c, n_0 \in \mathbb{N}$ such that $\epsilon(n) > 1/n^c$ for all $n \geq n_0$.
- For a distribution X, $x \leftarrow X$ denotes sampling randomly according to the distribution X. For a set $\mathcal{X}$, $x \leftarrow \mathcal{X}$ denotes sampling uniformly at random from the set $\mathcal{X}$. Similarly, $x \leftarrow \mathsf{X}$ is used to denote the output of a randomised algorithm (with fresh random coins).
- All logartihms are base 2 unless otherwise stated.

2.1.2 Search Problems

The definition of search problems is originally from [78, 85]. In this thesis we prefer to use an alternative, equivalent formulation from [65] given next. The justification is given in the remark following the definition.

Definition 7 (Search problem)**.** A search problem $\mathrm{P} = (\mathcal{I}, s)$ consists of an efficiently-decidable set of instances[1] $\mathcal{I} \subseteq \{0,1\}^*$ such that for every instance $I \in \mathcal{I}$ there exists a set of solutions $s(I) \subseteq \{0,1\}^*$. An algorithm A is said to *solve* P if, given as input an instance $I \in \mathcal{I}$, it:

1. outputs a solution $S \in s(I)$; or
2. outputs $\perp$ if $s(I) = \emptyset$ (i.e., if no solution exists).

The language $\mathrm{L_P}$ that encodes the corresponding decision problem can be defined as the subset of instances that have a solution:

$$\mathrm{L_P} := \{I \in \mathcal{I} : s(I) \neq \emptyset\} \subseteq \mathcal{I},$$

or equivalently using the relation

$$\mathcal{R}_\mathrm{P} := \{(I, S) : I \in \mathcal{I}, S \in s(I)\}.$$

Thus, intuitively, the solutions correspond to certificate of membership.

Remark 1 (On the domain of instances)**.** Observe that $\mathrm{L_P}$ is defined as a subset of the valid instances $\mathcal{I}$, and therefore the complement of $\mathrm{L_P}$ is defined with respect to $\mathcal{I}$ ($\overline{\mathrm{L}}_\mathrm{P} = \mathcal{I} \setminus \mathrm{L_P}$). This is contrary to the convention of defining a language as a subset of *all* strings and therefore every string is an instance (i.e., $\mathcal{I} = \{0,1\}^*$). The same applies to the definition of search problems in Definition 7. Note, however, that the two definitions are equivalent as long as the inclusion of a string in $\mathcal{I}$ is efficiently-decidable (which we assume). It turns out to be convenient in our context to restrict the domain to a subset of well-defined instances $\mathcal{I} \subseteq \{0,1\}^*$ since this allows a cleaner definition of a problem: simply describe the syntax of its instance and solution.[2] Consider, for example, Iterated-Squaring (IS) from Definition 5. A string parsed as $(N, x, T) \in \mathbb{N}^3$ is a valid IS instance only if $x \in \mathbb{Z}_N^*$. Therefore the set of instances of IS is

$$\mathcal{I}_\mathrm{IS} := \left\{(N, x, T) \in \mathbb{N}^3 : x \in \mathbb{Z}_N^*\right\}$$

and the algorithm that decides membership in $\mathcal{I}_\mathrm{IS}$ simply runs the Extended Euclidean Algorithm to check if x is invertible modulo N.

Notions of reducibility. We now define the two notions of reducibility between search problems that we use throughout: Karp reducibility (also known as Many-One reducibility) and Turing reducibility (also known as Cook reducibility). While the definition of Turing reducibility is similar to that for decision problems, the definition of Karp reducibility is slightly different [78].

---

[1] To be formal, there exists an efficient algorithm D that when given a string $I \in \{0,1\}^*$ *decides* (in time $\mathbf{poly}(|I|)$) whether $I \in \mathcal{I}$.

[2] Instead, if we had to work in a setting where all strings are required to be valid instances, then we would have to define a efficient encoding that *surjectively* maps valid tuples to strings.

Definition 8 (Karp reducibility)**.** A search problem $\mathrm{P}_1 = (\mathcal{I}_1, s_1)$ is said to be (polynomial-time) Karp-reducible to another search problem $\mathrm{P}_2 = (\mathcal{I}_2, s_2)$ if there exists two polynomial-time computable functions f and g such that

1. f maps every instance $I_1 \in \mathcal{I}_1$ of P_1 to a valid instance $I_2 \in \mathcal{I}_2$ of P_2; and
2. g maps every solution $S_2 \in s_2(I_2)$ of the instance I_2 to a solution $S_1 := g(S_2) \in s_1(I_1)$ of I_1.

Definition 9 (Turing reducibility)**.** A search problem P_1 is said to be (polynomial-time) Turing-reducible to another search problem P_2 if there exists a polynomial-time (oracle-aided) Turing machine $\mathsf{R}^{(\cdot)}$ that solves P_1 (as defined in Definition 7) when given oracle-access to a solver A for P_2.

It is worth pointing out that in cryptography, one usually deals with Turing reductions: given oracle access to an adversary that breaks a cryptographic protocol, the reduction algorithm breaks the cryptographic primitive. However for studying complexity classes, Karp reductions are preferred as they capture the finer structure of problems better than Turing reductions (e.g., **NP** = **co-NP** with respect to Turing reductions, but not Karp reductions).

2.1.3 NP Search Problems

In this section we formally define an **NP** search problem and the corresponding class **FNP**.

Definition 10 (Class **FNP**)**.** A search problem $\mathrm{P} = (\mathcal{I}, s)$ is a *Non-deterministic Polynomial-time* (**NP**) search problem if the set of solutions is efficiently-decidable: i.e., for every $I \in \mathcal{I}$ and its associated set of solutions $s(I) \subseteq \{0,1\}^{\mathbf{poly}(|I|)}$, given a candidate solution $S \in \{0,1\}^{\mathbf{poly}(|I|)}$ it is decidable in polynomial time (in $|I|$) whether $S \in s(I)$. The class of all **NP** search problems is called *Functional Non-deterministic Polynomial-time* (**FNP**).

A few remarks about **FNP**, in particular about its relationship with **NP**, are in place. The class **FNP** can alternatively be defined as the set of all **NP** search problems that are Karp-reducible to FSAT, the search counterpart of SAT [85]. This is analogous to the definition of **NP** in terms of SAT. Note that SAT is self-reducible (with respect to Turing reductions) [98]: i.e., given an oracle that *decides* SAT, it is possible to *find* a satisfying assignment and therefore break FSAT. It follows that **FP** = **FNP** if and only if **P** = **NP** [85]. This relationship would suggest, as pointed out in [85], that the study of **FNP** is uninteresting. However, it turns out that search problems corresponding to certain trivial decision problems in **NP** ∩ **co-NP** are themselves non-trivial. As we see next, this class corresponds exactly to **TFNP**.

2.1.4 Total NP Search Problems

The definition of **TFNP** is obtained from that of **FNP** by imposing the additional constraint that every instance must have a solution. This formal definition is given next, followed by the definition of the notion of average-case hardness that we use to study them.

Definition 11 (Class **TFNP**)**.** An **NP** search problem $\mathrm{P} = (\mathcal{I}, s)$ is *total* if the following holds:

- **Totality:** Every instance has a solution, i.e., $\forall I \in \mathcal{I} : s(I) \neq \emptyset$.

The class *Total Functional Non-deterministic Polynomial-time* (**TFNP**) consists of the set of all total problems in **FNP**.

Definition 12 (Average-case hardness in **TFNP**)**.** A **TFNP** search problem $\mathrm{P} = (\mathcal{I}, s)$ is hard-on-average if there exists an efficient instance-sampling algorithm I_{P} such that the following holds:

- **Totality:** Every instance has a solution, i.e., for every $\lambda \in \mathbb{N}$

$$\Pr_{I \leftarrow \mathsf{I}_{\mathrm{P}}(1^\lambda)} [s(I) \neq \emptyset] = 1.$$

- **Hardness:** For every efficient adversary A, there exists a negligible function ϵ such that for every $\lambda \in \mathbb{N}$

$$\Pr_{\substack{I \leftarrow \mathsf{I}_{\mathrm{P}}(1^\lambda) \\ S \leftarrow \mathsf{A}(I)}} [S \in s(I)] \leq \epsilon(\lambda)$$

We say that I_{P} defines a *hard distribution* on the instances of problem P.

Remark 2. The definition of average-case hardness for (general) search problems can be obtained by dropping the first point from Definition 11 .

Completeness in TFNP. We conclude the definitions by recalling a theorem from [78] which states that $\mathbf{TFNP} = \mathbf{F}(\mathbf{NP} \cap \mathbf{co\text{-}NP})$ (Theorem 2). It can then be argued that **TFNP** is unlikely to contain an **FNP**-complete problem (Theorem 3). This, taken into account with the fact that **TFNP** is semantic (cf. the discussion in Footnote 2), means that one ends up studying its syntactic sub-classes. We focus on two such classes, **PPAD** and **CLS**, in the two following sections.

Theorem 2 ([78])**.** $\mathbf{TFNP} = \mathbf{F}(\mathbf{NP} \cap \mathbf{co\text{-}NP})$.

Proof (Sketch). It is easy to show one direction of the containment, i.e. $\mathbf{TFNP} \subseteq \mathbf{F}(\mathbf{NP} \cap \mathbf{co\text{-}NP})$, by simply viewing a problem $\mathrm{P} \in \mathbf{TFNP}$ as a problem in **FNP**. To show the converse, i.e. that $\mathbf{F}(\mathbf{NP} \cap \mathbf{co\text{-}NP}) \subseteq \mathbf{TFNP}$, we exploit the correspondence between a search problem and its decision problem discussed in Definition 7. Consider a search problem $\mathrm{P} = (\mathcal{I}, s)$ in $\mathbf{F}(\mathbf{NP} \cap \mathbf{co\text{-}NP})$ and its corresponding decision problem $\mathrm{L}_{\mathrm{P}} \in \mathbf{NP} \cap \mathbf{co\text{-}NP}$. Since $\mathrm{L}_{\mathrm{P}} \in \mathbf{co\text{-}NP}$, it follows that its complement $\overline{\mathrm{L}}_{\mathrm{P}} \in \mathbf{NP}$. Therefore, the solution set $s(I)$ of an instance $I \in \mathcal{I}$ can be defined as the union of certificate of memberships in L_{P} and $\overline{\mathrm{L}}_{\mathrm{P}}$. Since L_{P} and $\overline{\mathrm{L}}_{\mathrm{P}}$ together cover whole of $\mathcal{I}$, it follows that every instance in P has at least one solution (viz., a proof of membership or a refutation). □

Theorem 3 ([78])**.** *There is an* **FNP***-complete problem in* **TFNP** *if and only if* $\mathbf{NP} = \mathbf{co\text{-}NP}$.

Proof. The "if" part follows Theorem 2. To see the converse, suppose for contradiction that **TFNP** contains a **FNP**-complete problem P. Moreover, let FSAT, the search counterpart of the **NP**-complete problem SAT, be Karp-reducible to P via two polynomial-time computable functions f and g. Next, let's consider an unsatisfiable FSAT instance ϕ. By the totality of P, there exists a solution S to the instance $f(\phi)$ of P (which is efficiently verifiable). It follows that ϕ has a (succinct) certificate of unsatisfiability in the form of $g(S)$ which puts UNSAT, the canonical **co-NP**-complete problem, in **NP**. This shows that **co-NP** $\subseteq$ **NP** and since these are complementary classes it follows that **NP** = **co-NP** (see [85, Proposition 2]). □

2.2 Hardness in PPAD

We start with a slightly different formulation of the End-of-Line problem in which the standard unpaired vertex can be an arbitrary vertex v_0. As shown in the lemma below, it is equivalent to the original formulation [84, 13] with 0^m as the standard unpaired vertex as in Definition 4.

Definition 13. End-of-Line (EOL)

- *Instance.*
  1. A successor circuit $\mathsf{S} : \{0,1\}^m \to \{0,1\}^m$
  2. A predecessor circuit $\mathsf{P} : \{0,1\}^m \to \{0,1\}^m$
  3. A vertex $v_0 \in \{0,1\}^m$
- *Guarantee.* v_0 is unbalanced: $\mathsf{P}(v_0) = v_0$ and $\mathsf{S}(v_0) \neq v_0$
- *Solution.* An unbalanced vertex $v \in \{0,1\}^m$: $\mathsf{P}(\mathsf{S}(v)) \neq v$ or $\mathsf{S}(\mathsf{P}(v)) \neq v \neq v_0$

Lemma 1. *Definition 4 is equivalent to Definition 13.*

Proof. First, any EOL instance (S, P) where the source is 0^m can be trivially transformed to an instance $(\mathsf{S}, \mathsf{P}, v_0 = 0^m)$. Second, we can reduce in the opposite direction by shifting the main line by v_0 as follows. Given an EOL instance $(\mathsf{S}, \mathsf{P}, v_0)$, define the new EOL instance as $(\mathsf{S}', \mathsf{P}')$ with source 0^m, where $\mathsf{S}'(v) := \mathsf{S}(v \oplus v_0)$ and $\mathsf{P}'(v) := \mathsf{P}(v \oplus v_0)$, where $\oplus$ denotes the bitwise XOR operation. □

Remark 3. Note that this general technique can be applied to any search problem where part of the instance is some significant vertex (e.g., Pigeon, SVL).

PPAD-hardness from IO. Recall from §1.3.2 that the construction of hard distribution on EOL instances in [13, 1] follows the following blueprint:

1. construct hard distribution of SVL; and
2. given an SVL instance from the above distribution, simulate EOL using reversible black pebbling.

The first step in [13] makes crucial use of indistinguishability obfuscation (IO) for circuits. Then they use techniques from reversible computing to obtain, in a fairly generic manner, an instance of EOL. As our constructions of hard End-of-Line instances follow a similar blueprint, let us go through these steps in more details.

2.2.1 Hard SVL Instances from IO

Consider the following construction of an SVL instance using a pseudorandom function (PRF) F. For a fixed key K picked at random from the key-space of the PRF, let v_i denote its evaluation using this key at i: i.e. $v_i = \mathsf{F}_K(i)$. For a security parameter λ, the SVL instance consists of a standard path of length $L = \omega(\mathbf{poly}(\lambda))$ and the i-th vertex on this path is set as (i, v_i) as shown below.

$$(0, \mathsf{F}_K(0)) \to (1, \mathsf{F}_K(1)) \to (2, \mathsf{F}_K(2)) \to \cdots \to (L-1, \mathsf{F}_K(L-1)) \to (L, \mathsf{F}_K(L))$$

The verifier circuit V on input a vertex (i, v) accepts if and only if $v = v_i$. The successor circuit S, given as input a valid vertex (i, v_i) returns $(i+1, v_{i+1})$. Although this construction is functional, it is inherently easy to solve: to ensure functionality, the key K must be hardwired into the successor and verifier circuits, and this allows anyone to compute the sink.

The issue with the above construction is that there is no means to ensure functionality without giving out the key (since a PRF is a symmetric-key primitive). This is where IO comes to the rescue. Recall that an obfuscator IO takes a program P, here represented as a Boolean circuit, as input and returns a functionally-equivalent circuit $\widehat{\mathsf{P}}$. Now, let us reconsider the construction described above but instead with the successor and verifier circuits obfuscated: $\widehat{\mathsf{S}} := \mathsf{IO}(\mathsf{S})$ and $\widehat{\mathsf{V}} = \mathsf{IO}(\mathsf{V})$. By the completeness of IO, the functionality of the successor and predecessor circuits is preserved. Moreover, its soundness property ensures that the hardwired key K remains "hidden" and consequently the sink too remains hidden thanks to the pseudorandomness of a PRF.

To formally show that the SVL instance is indeed hard, one uses the punctured programming technique from [96] to show that one cannot distinguish the above obfuscated circuits from their "punctured" counterparts where the standard path has been "erased" from a random point onwards. The sink in the SVL instance defined by such circuits is hidden information-theoretically. We refer the readers to [13] for further details.

2.2.2 Simulating the Predecessor Circuit

Given an SVL instance $\mathsf{SVL} = (\mathsf{S}, \mathsf{V}, L)$, the End-of-Line instance $\mathsf{EOL} = (\mathsf{S}, \mathsf{P}, v_0)$ is essentially simulated using *reversible black pebbling*, a game on directed acyclic graphs that was introduced to model reversible computation [8]. We formally define the game and the complexity-measure that is relevant to our discussion (i.e., space-complexity) in Definition 14 and then proceed to explain how the actual simulation is carried out in Lemma 2.

Definition 14 ([8])**.** For a directed acyclic graph $G = (\mathcal{V}, \mathcal{E})$, consider a sequence $\mathcal{P} := (\mathcal{P}_0, \ldots, \mathcal{P}_L)$ of pebbling configurations, where $\mathcal{P}_i \subseteq \mathcal{V}$ for all $i \in [0, L]$. We call such a sequence *a reversible black pebbling strategy* for G if the following two criteria are satisfied:

1. Two subsequent configurations in the sequence differ only in one vertex and the following rule is respected in each move: a pebble can be placed on or removed from a vertex if and only if all its parents carry a pebble.

2. The initial configuration is empty (i.e., $\mathcal{P}_0 = \emptyset$) and in the final configuration ($\mathcal{P}_L$) all the sink nodes are pebbled.

The *space-complexity* of a strategy $\mathcal{P} = (\mathcal{P}_0, \ldots, \mathcal{P}_L)$ for a DAG G is defined as

$$\mathbf{S}_G(\mathcal{P}) := \max_{i \in [0,L]} |\mathcal{P}_i|.$$

The space-complexity of a DAG G is the minimum space-complexity over all of its strategies $\mathcal{P}_G$:

$$\mathbf{S}(G) := \min_{\mathcal{P} \in \mathcal{P}_G} \mathbf{S}_G(\mathcal{P}). \tag{2.1}$$

A strategy matching the space-complexity of a DAG is deemed *space-optimal* for that DAG.

Lemma 2 ([1, 13])**.** Sink-of-Verifiable-Line *is Karp-reducible to* End-of-Line*.*

Proof. Notice that it is easy to construct the predecessor circuit in an *inefficient* way. One can simply modify the labels of the vertices to contain the entire history of the previous steps on the SVL line. Given such labels, implementing the predecessor is easy: simply remove the last element from the label. However, the obvious issue of this transformation is that the size of the labels eventually becomes exponentially large, which would render the resulting circuits S and P inefficient relative to the size of the SVL instance. To resolve this, we rely on reversible black pebbling.

For an SVL instance $\mathsf{SVL} = (\mathsf{S}, \mathsf{V}, L)$, let us consider the implicitly-defined standard path $\{0^m = v_0, \ldots, v_L\}$, where the i-th vertex is $v_i := \mathsf{SVL.S}^i(v_0)$ (see Figure 2.1). Since the EOL instance $\mathsf{EOL} = (\mathsf{S}, \mathsf{P}, \boldsymbol{v}_0)$ we define, with a standard path $\{\boldsymbol{v}_0, \ldots, \boldsymbol{v}_L\}$, is determined by the space-optimal reversible black pebbling strategy for this standard path, let us look at this particular strategy. For simplicity, let us consider the case when the length of the standard path is $L = 2^l$ for some $l \in \mathbb{N}$ – the case when L is not a power of 2 can be handled by dividing it into segments that *are* of length a power of 2.

There are l pebbles that can be placed on positions indexed by positive integers. The rules of the pebbling game from Definition 14 restricted to path graphs simplifies to: a pebble can be placed in or removed from position i if and only if either there is a pebble in position $i - 1$ or $i = 1$. The goal of the game is to place a pebble in position $2^l - 1$, the lone sink. As shown by Chung, Diaconis and Graham [37], the space-optimal strategy achieves the goal of the game in a recursive manner. Their main idea is to exploit the symmetry of the rules for placing and removing pebbles. Specifically, that it is always possible to reverse any sequence of moves. Suppose there is a way to get to $2^{l-1} - 1$ using only $l - 1$ pebbles. Then, place an additional pebble at 2^{l-1}. Next, free the first $l - 1$ pebbles by reversing the original sequence of moves performed in the first part. Finally, perform the same sequence starting from 2^{l-1}. As illustrated in Figure 2.1, this strategy will end with a pebble at position $2^l - 1$ while using only l pebbles.

The standard path implicit in EOL corresponds to the above space-optimal strategy, and its successor and predecessor circuits simply simulate it (see Figure 2.1). To be more precise, each vertex $\boldsymbol{v}_i$ in EOL corresponds to a configuration in the strategy and thus its label consists of the labels of all those vertices *in* SVL that are pebbled. Subsequently, a vertex in EOL has a label representing the states of at most $l = \log(L)$ pebbles, i.e., a tuple of pairs $\boldsymbol{v}_i := ((v_{i_1}, i_1), \ldots, (v_{i_l}, i_l))$ where each pebble corresponds to a pair (v_i, i) where, if you recall, $v_i = \mathsf{SVL.S}^i(v_0)$. (We presume that configurations with strictly less than l pebbles are padded accordingly.) For example, the labels on the standard path $\boldsymbol{v}_0, , \ldots, \boldsymbol{v}_{27}$ of the EOL instance given in Figure 2.1 are:

$$(v_0, 0) \to (v_0, 0) \| (v_1, 1) \to (v_1, 1) \to (v_1, 1) \| (v_2, 2) \to \ldots \to (v_0, 0) \| (v_7, 7) \to (v_7, 7).$$

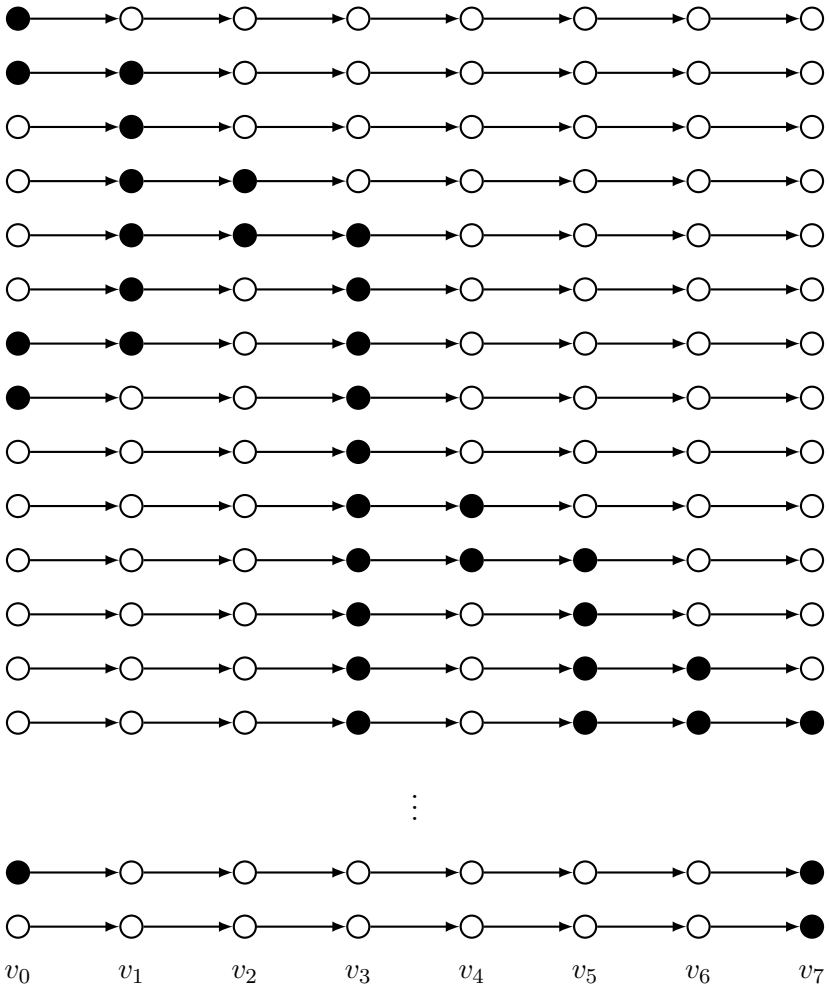


Figure 2.1: Simulating S and P using reversible pebbling. Each path graph above corresponds to an SVL instance $(v_0, \ldots, v_7)$ of length 8 (with the other vertices in self-loops). The EOL instance is determined by the space-optimal strategy for this path graph given above.

The rest of the tuples, that is the ones containing pairs that do not verify, become self-loops. Therefore, the resulting instance has a unique solution, a sink that identifies a solution to the original SVL instance.

The successor circuit for EOL now simply simulates the space-optimal strategy. If the next move in the strategy involves placing a pebble, the successor EOL.S computes the label corresponding to this vertex by invoking SVL.S and appends it to the current label. On the other hand, if the next move is a removal, it also simply removes the label corresponding to this vertex. The functioning of the predecessor circuit is analogous. The space-optimal strategy demonstrates that by storing only l intermediate states we can implement EOL.S and EOL.P that traverse the exponential SVL line from 0^m to $v_L = \mathsf{S}^L(0^m)$. However, this comes at a slight cost: the length of the standard path in this EOL instance is $L^{\log(3)}$ and the size of its labels is $2m \cdot \log L$. In particular, every vertex corresponding to an intermediate state of the strategy is followed by the subsequent state, and the final step of the strategy is a self-loop under EOL.S. For a more formal description of the reduction see [62]. □

2.3 Hardness in CLS

Recall that **CLS** consists of all problems that are Karp-reducible to Continuous-Local-Optimum (see [40] for the formal definition). To establish average-case hardness in **CLS**, however, we rely on End-of-Metered-Line (EOML) [62], a problem that is known to lie in **CLS** but not known to be complete.

Definition 15 ([62])**.** End-of-Metered-Line (EOML)

- *Instance.*
  1. A successor circuit $\mathsf{S} : \{0,1\}^m \to \{0,1\}^m$
  2. A predecessor circuit $\mathsf{P} : \{0,1\}^m \to \{0,1\}^m$
  3. A meter circuit $\mathsf{M} : \{0,1\}^m \to [0, 2^m]$
- *Guarantee.*
  1. 0^m is unbalanced: $\mathsf{P}(0^m) = 0^m$ and $\mathsf{S}(0^m) \neq 0^m$
  2. 0^m is the first vertex: $\mathsf{M}(0^m) = 1$
- *Solution.* 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 source:** $v \neq 0^m$ 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 to deviation in the correct behaviour of M also acts as a solution and thus puts End-of-Metered-Line in **TFNP**.

2.3.1 Reducing SVL to EOML

The approach of Bitansky et al., that was explained in the previous section, was extended by Hubáček and Yogev [62] to **CLS** under the same assumptions. They observed that in addition to simulating the successor and predecessor circuits, the simulation of SVL using reversible black pebbling yields a natural odometer M, an algorithm that simply returns the index of the pebbling configuration. Therefore, the same reduction basically yields an EOML instance.

Lemma 3 ([62])**.** Sink-of-Verifiable-Line *is Karp-reducible to* End-of-Metered-Line.

Proof. The construction of the successor and predecessor circuits for the EOML instance $\mathsf{EOML} = (\mathsf{S}, \mathsf{P}, \mathsf{M})$ is the same as in Lemma 2 – what is missing is the description of the odometer circuit M. However, it can be easily implemented as follows. For every valid pebbling configuration (i.e., a vertex that is not a self-loop), the meter $\mathsf{EOML.M}$ simply

outputs the number of steps taken in the pebbling strategy so far plus one (which can be computed efficiently just from the configuration itself) and the self-loops are given value 0. Thus, the resulting End-of-Metered-Line instance $(\mathsf{S}, \mathsf{P}, \mathsf{M})$, like the End-of-Line instance in Lemma 2, corresponds to a graph with a single line traversing the sequence of all the configurations visited by the space-optimal revsersible black pebbling strategy. □

2.4 Relaxing the SVL Problem

The definition of the SVL problem that we saw in Definition 6 (§2.2) is very rigid in the sense that its verifier circuit V behaves "perfectly". The instances that result from such a verifier consist of a *single* standard path that starts at the source vertex and ending at the sink (see Figure 1.8). The verifier attests to every vertex on this path and rejects the ones that lie off it. In particular, this means a verifier cannot accept two different vertices for the same index.

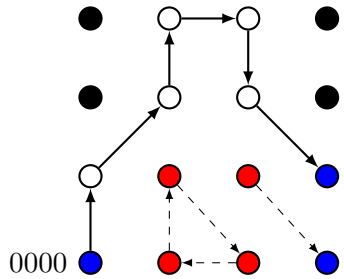


Figure 2.2: Sample Relaxed-Sink-of-Verifiable-Line instance with 16 vertices (i.e., $m = 4$) with a standard path of length $L = 6$. The output of the successor circuit is indicated by the directed edges: i.e., an edge (u, v) implies that $\mathsf{S}(\boldsymbol{v}) = v$. The source (resp., sink) vertices are in red (resp., blue). The vertices that lie off the path but are accepted by the verifier, i.e. the relaxed vertices, are in red and the paths they form are the dashed ones. Thus, it is possible that the blue sink with the dashed incoming edge is also accepted as the L-th vertex on the line, i.e. as a solution. The vertices rejected by the verifier are in black.

The construction of SVL from IO (presented in §2.2.1) did have such a perfect verifier thanks to the power of obfuscation. However, we saw in the overview of our constructions (§1.3.2) that in some cases these vertices comprise of (non-interactive) proofs. To meet the stringent requirements of the SVL verifier would require designing (non-interactive) proof systems that are both perfectly sound (i.e., false proofs are always rejected) and unambiguous (i.e., every statement has a unique proof). Sometimes, however, such proofs are hard to construct or outright impossible. Instead we relax the promise involved in SVL to accommodate imperfect proof systems via a Relaxed-Sink-of-Verifiable-Line problem.

Definition 16. Relaxed-Sink-of-Verifiable-Line (RSVL)

- *Instance.*
  1. A successor circuit $\mathsf{S} : \{0,1\}^m \to \{0,1\}^m$

2. A verifier circuit $\mathsf{V} : \{0,1\}^m \times [1,2^m] \to \{0,1\}$
3. Length $L \in [1,2^m]$

- *Promise.* For every $v \in \{0,1\}^m$ and $i \in [1,2^m]$, $\mathsf{V}(v,i) = 1$ *if*

  1. $i \leq L$; and
  2. $v = \mathsf{S}^i(0^m)$.

- *Solution.* 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 $(v,i) \in \{0,1\}^m \times [0,2^m]$ such that $v \neq \mathsf{S}^i(0^m)$ and $\mathsf{V}(v,i) = 1$.

The main difference from Definition 6 is that the promise about the behaviour of the verifier circuit V is relaxed so that it can also accept vertices off the standard path: the promise is relaxed from an *if and only if* to an *if*. However, any vertex off the main line accepted by V is an additional solution, a *false positive*. Consequently, for the SVL instance to remain hard, these off-the-path vertices need to be computationally-hard to find. In Lemma 4 we show that despite the relaxed promise, RSVL reduces to EOML and therefore constructing hard distributions of EOML is reduced to constructing hard distributions of RSVL.

Lemma 4. Relaxed-Sink-of-Verifiable-Line *is (randomised) Karp-reducible to* End-of-Metered-Line.

Proof. The proof of the lemma follows by inspection of the reductions from SVL to EOL (Lemma 2) and SVL to EOML (Lemma 3). Let $\mathsf{RSVL} = (\mathsf{S},\mathsf{V},L)$ be an RSVL instance. Let's consider the EOML instance $\mathsf{EOML} = (\mathsf{S},\mathsf{P},\mathsf{M})$ obtained by applying the reduction in Lemma 3 to this RSVL instance. (Recall that the length of the standard path in this EOML instance is $L^{\log(3)}$ and the size of its labels is $2m \cdot \log L$.) The main issue is that due to the relaxed guarantee, we might have introduced additional solutions besides the sink corresponding to $v_L = \mathsf{RSVL.S}^L(0^m)$. We argue that any such solution will correspond to a false positive in the RSVL instance. To this end, we claim that EOML has the following properties.

1. Every vertex $\boldsymbol{v}$ is labelled by a tuple of $l = \log(L)$ pairs of the form $(v,i) \in \{0,1\}^m \times [1,L]$.
2. Every vertex $\boldsymbol{v}$ that is not a self-loop (i.e., with $\mathsf{EOML.S}(\boldsymbol{v}) \neq \boldsymbol{v}$ or $\mathsf{EOML.P}(\boldsymbol{v}) \neq \boldsymbol{v}$) contains in its label only pairs $(v,i) \in \{0,1\}^m \times [1,L]$ such that $\mathsf{RSVL.V}(v,i) = 1$ and is given a non-zero value by $\mathsf{EOML.M}$.
3. A vertex $\boldsymbol{v}$ lies on the main directed line starting at the standard source corresponding to the initial pebbling configuration if and only if it contains in its label only pairs $(v,i) \in \{0,1\}^m \times [1,L]$ such that $v = \mathsf{RSVL.S}^i(0^m)$.

Items 1 and 2 are immediate from the description of the reduction. Item 3 follows since any vertex $\boldsymbol{v}$ which is not a self-loop corresponds to a valid pebbling configuration. Thus, if the label of $\boldsymbol{v}$ contains only pairs $(v,i) \in \{0,1\}^m \times [1,L]$ such that $v = \mathsf{RSVL.S}^i(0^m)$

then the successor (resp., predecessor) of $\boldsymbol{v}$ on the main directed EOML line is the successive (resp., preceding) pebbling configuration. Note that the converse implication is straightforward.

Consider any vertex $\boldsymbol{v}$ that is a solution (of any type from Definition 15) to EOML. If the label of $\boldsymbol{v}$ contains a pair of the form (v, L) then we have found a solution to the relaxed SVL instance. By Item 2, $\mathsf{RSVL.V}(v, L) = 1$ and either v is the sink of the RSVL instance (when $v = \mathsf{RSVL.S}^L(0^m)$) or (v, L) is a false positive (when $v \neq \mathsf{RSVL.S}^L(0^m)$).

Otherwise, we show that the solution $\boldsymbol{v}$ must contain a false positive in its label. First, notice that there are no other solutions on the main directed line besides the sink containing (v_L, L) in its label (this sink falls into the previous case we already handled). Therefore, $\boldsymbol{v}$ lies off the main standard path and, by Items 2 and 3 above, its label must contain a pair (v, i) such that $\mathsf{RSVL.V}(v, i) = 1$ but $v \neq \mathsf{RSVL.S}^i(0^m)$, i.e., a false positive. Since there are $\log(L)$ such pairs in the label, we can select the false positive (v, i) with a noticeable probability simply by picking one of the pairs in the label uniformly at random (rendering it a *randomised* Karp reduction). □

Chapter 3
PPAD-Hardness via Iterated-Squaring

In the previous chapter, we saw how the task of constructing hard distribution of End-of-Line reduces to constructing a hard distribution on the Relaxed-Sink-of-Verifiable-Line (RSVL) problem. In this chapter, we formally present our first construction of hard distribution of RSVL, denoted RSVL_1. The hardness of the construction, if you recall, is in the random-oracle model under an assumption on Iterated-Squaring (IS). The exact hardness assumption along with its relevant background is described in §3.1. Since our construction is inspired by Pietrzak's interactive protocol for the *decision* problem corresponding to IS [88], we describe it in detail in §3.2. The construction of the RSVL instance is finally given in §3.3, and there we also explain the notion of *recursive proof-merging* which constitutes the main technical novelty in this thesis.

3.1 Hardness Assumption

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 §3.1.2, we describe the weaker assumption (Assumption 2) sufficient for the hardness of our construction.

3.1.1 The RSW Time-Lock Puzzle

Rivest, Shamir and Wagner [94] introduced the notion of time-lock puzzles. Such a puzzle is specified by a sampling algorithm which, on input a security parameter $\lambda \in \mathbb{N}$ and a time parameter T, outputs a puzzle instance I and the corresponding solution S. The solution S can be computed given only the puzzle I making T simple sequential steps. The security property requires that even an adversary with $\mathbf{poly}(\lambda)$ parallelism cannot compute the solution much faster than the honest (sequential) algorithm. They also propose a simple and elegant construction: on input (λ, T), a puzzle is sampled by choosing two random $\lambda_{\mathrm{RSA}}/2$-bit primes p, q (see Remark 4), which define a λ_{RSA}-bit modulus $N := p \cdot q$, together with any $x \in \mathbb{Z}_N^*$. The puzzle and solution are then defined as

$$I := (N, x, T),\ S = f(N, x, T) := x^{2^T} \bmod N.$$

Observe that this is precisely the search problem Iterated-Squaring (IS) from Definition 5. The solution to the puzzle, or equivalently to the IS instance, can be efficiently computed by the puzzle-sampling algorithm in two steps using the knowledge of the group order $\varphi(N) = (p-1)(q-1)$ as

$$e := 2^T \bmod \varphi(N),\ S := x^e \bmod N. \tag{3.1}$$

It follows that IS is at most as hard as Factoring.

Theorem 4 (Folklore)**.** ITERATED-SQUARING *is Karp-reducible to* FACTORING*.*

We do not know whether the converse holds, i.e., whether FACTORING reduces to IS for any $T = T(\lambda)$. It is conjectured in [94] that the fastest way to solve IS without the knowledge of the group order is through repeated squaring:

$$x \to x^2 \to x^{2^2} \to x^{2^3} \to \ldots \to x^{2^T} \pmod{N}. \tag{3.2}$$

In particular, parallelism (beyond what can be used to speed up a single squaring) does not allow to compute the solution any faster. In other words, IS is conjectured to be an *inherently-sequential* computational problem. 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 time". For instance, if the algorithm is given as a circuit, then the running time would depend on its size, whereas the sequential time only on its depth.

Assumption 1 (Sequential hardness of IS [94])**.** For a security parameter $\lambda \in \mathbb{N}$, let N be the product of two random $\lambda_{\text{RSA}}/2$-bit primes and $x \leftarrow \mathbb{Z}_N^*$ be sampled uniformly at random. Consider any $T \in \mathbb{N}$ and any algorithm A such that T and running time of A is significantly smaller than what is required to factor N. Any A that solves the IS instance (N, x, T) with a non-negligible probability requires *sequential time* not much less than T times the sequential time required for a single squaring in $\mathbb{Z}_N^*$.

Remark 4 (λ vs. λ_{RSA})**.** A construction with security parameter λ is supposed to guarantee λ bits of security i.e., it should ideally take an adversary $O(2^\lambda)$ computation to break the construction using the best known attacks. Therefore, λ_{RSA} is computed from λ by taking into account the best-known factoring algorithms (which currently is the General Number Field Sieve (GNFS) [73]). For instance, for 128-bit security the size of the modulus currently recommended is 2048 bits. However since the GNFS is a sub-exponential algorithm, asymptotically speaking, one could assume that $\lambda_{\text{RSA}} \in \mathbf{poly}(\lambda)$ and we will do this throughout the chapter.

3.1.2 Our Number-Theoretic Assumption

The hardness result in this chapter is based on a weaker assumption where we just require that for some superpolynomial T, a variant of IS cannot be solved in polynomial time. The exact algebraic setting for our assumption, as formally stated in Assumption 2, differs slightly from that in Assumption 1. We highlight the differences below and justify at the end of the section (Proposition 1) why these changes do not really affect the strength of the assumption.

1. First, the group that we assume our hardness is the group of *signed quadratic residues* QR_N^+, described in §3.1.2.1, which carries slightly more structure than $\mathbb{Z}_N^*$. This extra structure helps guarantee *unambiguous soundness* in Pietrzak's protocol: see §3.2.2 for the details.

2. Second, we require the primes p, q that define the modulus N to be *safe* primes. This is to ensure that QR_N^+ contains no sub-group of small order, a property that is exploited to prove statistical soundness of the proof system (Lemma 6).[1]

[1]One can prove soundness of the protocol also when a standard RSA modulus is used (i.e., p and q

3.1.2.1 Signed Quadratic Residues

For two safe primes p and q, and $N := p \cdot q$ the signed quadratic residues [46, 59] is defined as the group

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

where $|x|$ is the absolute value when representing the elements of $\mathbb{Z}_N^*$ as

$$\left\{-\frac{N-1}{2}, \ldots, \frac{N-1}{2}\right\}. \tag{3.3}$$

Since $-1 \in \mathbb{Z}_N^*$ is a quadratic non-residue with Jacobi symbol $+1$, the function $|\cdot|$ 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 in eq.(3.3)) belongs also to QR_N+, ensure that $x \geq 0$ *and* that its Jacobi symbol is $+1$ using [38, Algorithm 1.4.10].

Generators of QR_N^+. Before moving on the assumption, 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 §3.3 (Claim 6.2 in Theorem 6). Let's denote by $QR_N^\star \subset QR_N^+$ the set of generators of QR_N^+:

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

If $N := p \cdot q = (2p'+1)(2q'+1)$ is the product of $\lambda_{\text{RSA}}/2$-bit safe primes, then we have

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

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

$$\Pr_{x \leftarrow QR_N^+}[x \in QR_N^\star] = 1 - \frac{p'+q'-1}{p' \cdot q'} \geq 1 - \frac{1}{2^{\lambda_{\text{RSA}}/2}} \ . \tag{3.4}$$

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

---

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 [17, 105]. So the hardness of our RSVL instance would rely on hardness of repeated squaring in QR_N^+ as stated in Assumption 2, and additionally on the hardness of finding some element z where $z^e = x$ for some e of polynomial size.

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

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 function is an automorphism[3] of QR_N^+ (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

$$\left\{x, x^2, x^{2^2}, x^{2^3}, \ldots, x^{2^{(p'-1)(q'-1)-1}} = x\right\} \subseteq QR_N^\star. \tag{3.5}$$

3.1.2.2 The Assumption

The hardness assumption that underlies the RSVL instance proposed in this chapter is stated below. It pertains to the Iterated-Squaring problem adapted to the setting of signed quadratic residues. The formal definition of the problem, which we call Iterated-Squaring$^+$ (IS$^+$) to avoid conflation with IS, is given below. The description of the hardness assumption follows.

Definition 17. Iterated-Squaring$^+$ (IS$^+$)[4]

- *Instance.*
  1. Modulus $N \in \mathbb{N}$
  2. Group element $x \in QR_N^+$
  3. Time parameter $T \in \mathbb{N}$
- *Solution.* $f^+(N, x, T) := x^{2^T} = \underbrace{x \circ \ldots \circ x}_{2^T \text{ times}}$

As explained in §3.1.2.1, checking whether a string that parses as (N, x, T) is a valid instance reduces to checking that x has a Jacobi symbol of $+1$ modulo N.

Assumption 2 (Average-case hardness of IS$^+$)**.** For a security parameter $\lambda \in \mathbb{N}$, let $N := p \cdot q$ be the product of two random $\lambda_{\text{RSA}}/2$-bit *safe* primes p, q and $x \leftarrow QR_N^+$ be sampled uniformly at random. There exists some $T \in \left[\omega(\mathbf{poly}(\lambda)), 2^\lambda\right]$, such that no $\mathbf{poly}(\lambda)$-time algorithm can solve the Iterated-Squaring$^+$ instance (N, x, T) with non-negligible probability.

Note that we allow the time parameter T to be any super-polynomial value, but restrict it to be representable by $\mathbf{poly}(\lambda)$-many bits. Even though it may seem that computing the solution can only get harder as T increases, this is not actually true. For instance, if T is the product of all $\lambda_{\text{RSA}}/2$-bit primes, then computing $x^{2^T} \bmod N$ is actually trivial as

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

The final equality holds as $T \bmod \varphi(\varphi(N)) = 0$. (However, such a T takes $\omega(\mathbf{poly}(\lambda))$ bits if represented normally as an integer.) Next we argue that Assumption 2 is at most as strong as Assumption 1 in this time parameter regime. Note that Assumption 2 is vacuous for $T \in O(\mathbf{poly}(\lambda))$.

---

[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 [15] – it is an automorphism.

[4]We 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.

Proposition 1. *Under Conjecture* 6.1 *from [104], Assumption 2 is at most as strong as Assumption 1 for* $T \in \left[\omega(\mathbf{poly}(\lambda)), 2^{\lambda}\right]$ *under (randomised) Karp reductions.*

Proof. We split the proof into two claims. We first argue in Claim 4.1 that a switch to safe primes does not affect Assumption 1 as the density of safe primes among all primes is sufficiently high under the conjecture. Then, in Claim 4.2, we show that Algorithm 2 (which is in the setting of $(QR_N^+, \circ)$) is at most as strong as this assumption (which is in $(\mathbb{Z}_N^*, \cdot)$) with respect to randomised reductions.

Claim 4.1. *Under Conjecture* 6.1 *from [104], Assumption 1* with safe primes *is at most as strong as Assumption 1.*

Proof (Sketch). Conjecture 6.1 from [104] states that for some constant c, there are $c \cdot 2^n/n^2$ safe n-bit primes. If this conjecture holds, a random n-bit prime is safe with probability $\approx c/n^2$. Even under a weaker requirement that there are at least $2^n/\mathbf{poly}(n)$ safe n-bit primes for some polynomial in $\mathbf{poly}(n)$, Assumption 1 is at least as strong as an assumption where we additionally require p and q to be safe: if a $\Theta(1/\mathbf{poly}(n))$ 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 $\Theta(1/\mathbf{poly}(n)^2)$. □

Claim 4.2. *Assumption 2 is at most as strong as Assumption 1* with safe primes.

Proof (Sketch). 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)).

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 $(QR_N, \cdot)$.

□

□

3.2 Pietrzak's Proof System

The key component of our construction is Pietrzak's interactive protocol [88] for the decision problem corresponding to ITERATED-SQUARING$^+$, i.e. showing that a tuple $I = (N, x, T, y)$ satisfies $y = x^{2^T}$ over QR_N^+. There, the motivation was to construct a cryptographic primitive called *verifiable delay function* [16, 17] (see the discussion in Chapter 5). The protocol we use in this work differs in some minor aspects from the one in [88]. These changes we introduce make the proof system less efficient but enable a cleaner description — see Remark 7 for further details. But first, we formally introduce the different notions of proof systems. In particular, we define *unambiguous* proof systems from [93], which formalises the *uniqueness* property that is crucial for our constructions.

3.2.1 Proof Systems

Interactive protocols. An interactive protocol (see Figure 1.7.(a)) consists of a pair (P, V) of interactive Turing machines that are run on a common input I. The first machine is called the prover and is denoted by P, and the second machine, which is probabilistic, is called the verifier and is denoted by V. In an ρ-round (i.e., $(2\rho-1)$-message) interactive protocol, in each round $i \in [1, \rho]$, first P sends a message $\alpha_i \in \Sigma^a$ to V and then V sends a message $\beta_i \in \Sigma^b$ to P, where Σ is a finite alphabet. At the end of the interaction, V runs a (deterministic) Turing machine on input $(I, (\beta_1, \ldots, \beta_\rho), (\alpha_1, \ldots, \alpha_\rho))$. The interactive protocol is *public-coin* if β_i is a uniformly distributed random string in Σ^b. The *communication complexity* of an interactive protocol is the total number of bits transmitted, namely, $(\rho \cdot (b + a) \cdot \log(|\Sigma|))$.

Interactive proofs (IPs). The classical notion of an interactive proof for a language L is due to Goldwasser, Micali and Rackoff [56].

Definition 18. An interactive protocol (P, V) is a δ-*sound* interactive proof (IP) for L if:

- **Completeness:** For every $I \in \mathrm{L}$, if V interacts with P on common input I, then V accepts with probability 1.
- **Soundness:** For every $I \notin \mathrm{L}$ and every (computationally-unbounded) cheating prover strategy $\widetilde{\mathsf{P}}$, the verifier V accepts when interacting with $\widetilde{\mathsf{P}}$ with probability less $\delta(|I|)$, where $\delta = \delta(n)$ is called the *soundness error* of the proof system.

Remark 5. Just like in Remark 1, we also consider the above definition for languages where the domain is restricted to a set of instances. This applies likewise to the definitions that follow.

Unambiguous IPs. Reingold, Rothblum and Rothblum [93] introduced a variant of interactive proofs, called *unambiguous* interactive proofs, in which the *honest* prover strategy is defined for *every* I (i.e., also for $I \notin \mathrm{L}$) and the verifier is required to reject when interacting with any cheating prover that deviates from the prescribed honest prover strategy at any point of the interaction. Therefore, one can think such interactive protocols as having a "unique" execution.

More formally, if (P, V) is an interactive protocol, and $\widetilde{\mathsf{P}}$ is some arbitrary (cheating) strategy, we say that $\widetilde{\mathsf{P}}$ *deviates* from the protocol at round i^* if the message sent by $\widetilde{\mathsf{P}}$ in round i^* differs from the message that P would have sent given the transcript of the protocol thus far. In other words, if the verifier sent the messages $\beta_1, \ldots, \beta_{i^*-1}$ in rounds $1, \ldots, i^* - 1$ respectively, we say that $\widetilde{\mathsf{P}}$ deviates from the protocol at round i^* if

$$\widetilde{\mathsf{P}}\left(I, i^*, (\beta_1, \ldots, \beta_{i-1})\right) \neq \mathsf{P}\left(I, i^*, (\beta_1, \ldots, \beta_{i-1})\right).$$

We consider a slightly different formulation, where the unambiguity is required to hold *only* for $I \in \mathrm{L}$. Therefore for $I \notin \mathrm{L}$, we need to reinstate the standard soundness condition.

Definition 19. An interactive protocol (P, V), in which we call P the *prescribed prover*, is a (δ, ϵ)-unambiguosly sound IP for L if the following three properties hold:

- **Completeness:** For every $I \in \mathrm{L}$, if V interacts with P on common input I, then V accepts with probability 1.
- **Soundness:** For every $I \notin \mathrm{L}$ and every (computationally-unbounded) cheating prover strategy $\widetilde{\mathsf{P}}$, the verifier V accepts when interacting with $\widetilde{\mathsf{P}}$ with probability less than $\delta(|I|)$, where $\delta = \delta(n)$ is called the *soundness error* of the proof system.
- **Unambiguity:** For every $I \in \mathrm{L}$, every (computationally-unbounded) cheating prover strategy $\widetilde{\mathsf{P}}$, every round $i^* \in [1, \rho]$, and for every $\beta_1, \ldots, \beta_{i^*-1}$, if $\widetilde{\mathsf{P}}$ first deviates from the protocol in round i^* (given the messages $\beta_1, \ldots, \beta_{i^*-1}$ in rounds $1, \ldots, i^* - 1$ respectively), then at the end of the protocol V accepts with probability at most $\epsilon(|I|)$, where the probability is over V's coin tosses in rounds $i^*, \ldots, \rho$.

An IP is simply δ-unambiguosly sound if $\epsilon = \delta$.

Non-interactive proof systems. A non-interactive proof system involves the prover sending a single message to the verifier. To give this proof system additional power[5], we assume that both prover and verifier have access to a common reference string (CRS). When the CRS is simply a uniformly random string from some domain $\mathcal{R}$, it is referred to as a *common random string*. We focus on *adaptive* proof systems where a cheating prover gets to see the CRS *before* forging a proof for a statement of its choice. As for the case of interactive proofs, we consider unambiguous non-interactive proof systems (instead of the standard "sound" non-interactive proof systems).

Definition 20. A pair of machines (P, V), where P is the prescribed prover, is a (δ, ϵ)-unambiguosly *adaptively* sound non-interactive proof system for a language L if V is probabilistic polynomial-time and the following three properties hold:

- **Completeness:** For every $I \in \mathrm{L}$,

$$\Pr_{R \leftarrow \mathcal{R}}\left[\mathsf{V}(I, \mathsf{P}(I, R), R) = 1\right] = 1,$$

  where the probability is over the random choice of the CRS $R \in \mathcal{R}$.

[5]It was shown in [82] that a non-interactive protocol without a CRS exists only for trivial languages, i.e. those in **BPP**.

- **Soundness:** For every (computationally-unbounded) cheating prover strategy $\widetilde{\mathsf{P}}$,

$$\Pr_{\substack{R\leftarrow\mathcal{R}\\(I,\widetilde{\pi})\leftarrow\widetilde{\mathsf{P}}(R)}}[\mathsf{V}(I,\widetilde{\pi},R)=1\wedge I\notin \mathrm{L}]\leq\delta(|I|).$$

- **Unambiguity:** For every (computationally-unbounded) cheating prover strategy $\widetilde{\mathsf{P}}$,

$$\Pr_{\substack{R\leftarrow\mathcal{R}\\(I,\widetilde{\pi})\leftarrow\widetilde{\mathsf{P}}(R)\\\pi\leftarrow\mathsf{P}(I,R)}}[\mathsf{V}(I,\widetilde{\pi},R)=1\wedge\widetilde{\pi}\neq\pi\wedge I\in \mathrm{L}]\leq\epsilon(|I|).$$

A non-interactive protocol is simply δ-unambiguosly sound if $\epsilon=\delta$.

Remark 6. A non-interactive protocol is called an *argument system* if the soundness holds only against efficient cheating prover strategy $\widetilde{\mathsf{P}}$.

3.2.2 Pietrzak's Protocol

An informal descsription of the interactive Pietrzak Protocol PP for the decision problem corresponding to IS was given in §1.3.2.2. The high level ideas for the algebraic setting of IS^+ is similar and is hence omitted. We define the protocol formally in Algorithm 3.1 and highlight some of the key differences from the original protocol from [88] in the remark below.

Remark 7 (Comparison with [88])**.** 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 prover described above uses the minimum space necessary, even though additional space can significantly improve the efficiency of the computation of the proof.

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

3. 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 [88].

3.2.2.1 Removing Interaction

The interactive protocol PP described in Algorithm 3.1 is public-coin and the proof system has an exponentially-small soundness error, which means it could be turned non-interactive using the Fiat-Shamir methodology [44]. However, the Fiat-Shamir Transform is known to be sound in the random-oracle model when applied to *constant-round*

Algorithm 3.1 Interactive Pietrzak Protocol PP [88]

procedure $\mathsf{P}(N, x, T, y)$ ▷ Prover
input
1. IS$^+$ instance $(N, x, T) \in \mathbb{N} \times QR_N^+ \times \mathbb{N}$ where $T = 2^t$
2. Solution to the IS$^+$ instance $y = x^{2^T} \in QR_N^+$

  Initialise $(x_1, y_1) := (x, y)$
  for $i \in [1, t]$ **do** ▷ Halving subprotocol
    $\mu_i := x_i^{2^{T/2^i}} \in QR_N^+$ ▷ Compute the midpoint
    Send μ_i to V ▷ $\mathsf{P} \to \mathsf{V}$
    Receive challenge $r_i \in \{0,1\}^\lambda$ from V ▷ $\mathsf{P} \leftarrow \mathsf{V}$
    $x_{i+1} := x_i^{r_i} \circ \mu_i$, $y_{i+1} := \mu_i^{r_i} \circ y_i$ ▷ Compute random linear combination
  end for
  HALT ▷ The prover halts without any output
end procedure

procedure $\mathsf{V}(N, x, T, y)$ ▷ Verifier
input same as P
output a bit indicating ACCEPT or REJECT
  Initialise $(x_1, y_1) := (x, y)$
  for $i \in [1, t]$ **do** ▷ Halving subprotocol
    Receive $\mu_i \in QR_N^+$ from P ▷ $\mathsf{P} \to \mathsf{V}$
    Choose a random element $r_i \leftarrow \{0,1\}^\lambda$ ▷ Next challenge
    Send r_i to P ▷ $\mathsf{P} \leftarrow \mathsf{V}$
    $x_{i+1} := x_i^{r_i} \circ \mu_i$, $y_{i+1} := \mu_i^{r_i} \circ y_i$ ▷ Compute random linear combination
  end for
  if $y_{t+1} = x_{t+1}^2$ **then return** 1
  else return 0
  end if
end procedure

interactive protocols.[6] Nonetheless, it was shown in [88] that PP can be compiled into a non-interactive protocol that is sound in the random-oracle model. The formal description of this non-interactive protocol, denoted $\overline{\mathsf{PP}}$, is given in Algorithm 3.2. Notice that the transform used in $\overline{\mathsf{PP}}$ is slightly different from the standard Fiat-Shamir Transform we described in §1.3.1.2: the hash function in $\overline{\mathsf{PP}}$ is applied only to the statement and the prover's message for that round (and not the whole transcript till that round). However, this does not affect the soundness of the resulting non-interactive protocol as shown in the next section.

Instantiating Fiat-Shamir. A number of recent works have aimed at relaxing the assumptions under which the Fiat-Shamir Transform can be proved sound [68, 66, 29]. In particular, [29] 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. We see such an instantiation for our second construction RSVL_2 in §4.4, and point out that the same machininery could be applied to the Pietrzak's protocol PP as well.

3.2.3 Unambiguous Soundness

We show that $\overline{\mathsf{PP}}$ is unambiguosly sound in the random-oracle model and the idea is to show that an adversary (i.e., a cheating prover) never finds a "bad query" as defined in Definition 21. 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 5); conditioned on the adversary not making a bad query, we prove that unambiguous soundness is hard to break in the statistical sense (Lemma 6). It follows that $\overline{\mathsf{PP}}$ is non-interactive proof system that is statistically unambiguously sound in the random oracle model (Theorem 5). Before moving on to the proof, we fix a notation that will be used further in this section: we reserve

$$\pi(x \xrightarrow{T} y) \leftarrow \overline{\mathsf{PP}}.\mathsf{P}^H(N, x, T, x^{2^T})$$

denote prescribed proofs for true statements in $\overline{\mathsf{PP}}$, and $\tilde{\pi}(x \xrightarrow{T} y)$ denote any string that parses as a possible proof.

Definition 21 (Bad query)**.** A query is a tuple (μ, x, y, T) where $\mu, x, y \in QR_N^+$ and $T \in \mathbb{N}$. Let

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

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

[6] In [44] the assumption that the transform is sound was made for *constant-round* protocols. It was later shown in [89] that this transformation is sound when H is instantiated with a random oracle. However, there are examples of non-constant round interactive protocols that are unsound when the Fiat-Shamir methodology is applied, even in the random-oracle model. One simple counter-example is to consider the sequential composition, say n times, of a constant-round protocol that has constant soundness error. By standard amplification techniques, the resultant protocol can be shown to have a negligible soundness error — inverse-exponential in n to be precise — but the Fiat-Shamir Transformed protocol is totally insecure as argued next. Since the soundness error is constant a malicious prover can basically sample (in constant number of attempts in expectation) "favourable" messages for each round and with high probability come up with a cheating proof.

Algorithm 3.2 Non-Interactive Pietrzak Protocol $\overline{\mathsf{PP}}$

procedure $\overline{\mathsf{PP}}.\mathsf{P}^H(N, x, T, y)$ ▷ Prover
oracle $H : \{0,1\}^* \to \{0,1\}^\lambda$ ▷ Random oracle
input

1. IS$^+$ instance $(N, x, T) \in \mathbb{N} \times QR_N^+ \times \mathbb{N}$ where $T = 2^t$
2. Solution to the IS$^+$ instance $y = x^{2^T} \in QR_N^+$

output Proof $\pi(x \xrightarrow{T} y) \in \mathbb{N} \times QR_N^+ \times \mathbb{N} \times QR_N^+ \times (QR_N^+)^t$

Initialise $(x_1, y_1) := (x, y)$
for $i \in [1, t]$ **do**
  $\mu_i := x_i^{2^{T/2^i}} \in QR_N^+$ ▷ Compute the midpoint
  $r_i := H(\mu_i, x_i, y_i, T/2^{i-1})$ ▷ Compute the challenge
  $x_{i+1} := x_i^{r_i} \circ \mu_i,\ y_{i+1} := \mu_i^{r_i} \circ y_i$ ▷ Compute random linear combination
end for
return $\pi(x \xrightarrow{T} y) = (N, x, T, y, \boldsymbol{\mu} := \mu_1, \ldots, \mu_t)$
end procedure

procedure $\overline{\mathsf{PP}}.\mathsf{V}^H(\tilde{\pi}(x \xrightarrow{T} y))$ ▷ Verifier
oracle $H : \{0,1\}^* \to \{0,1\}^\lambda$ ▷ Random oracle
input Proof $\tilde{\pi}(x \xrightarrow{T} y)$ parsed as

$$(N, x, T = 2^t, y, \boldsymbol{\mu} = \mu_1, \ldots, \mu_t) \in \mathbb{N} \times QR_N^+ \times \mathbb{N} \times QR_N^+ \times (QR_N^+)^t$$

output a bit indicating ACCEPT or REJECT

Initialise $(x_1, y_1) := (x, y)$
for $i \in [1, t]$ **do**
  $r_i := H(\mu_i, x_i, y_i, T/2^{i-1})$ ▷ Recompute the challenge
  $x_{i+1} := x_i^{r_i} \circ \mu_i,\ y_{i+1} := \mu_i^{r_i} \circ y_i$ ▷ Recompute random linear combination
end for
if $x_{t+1}^2 = y_{t+1}$ **then return** 1
else return 0
end if
end procedure

(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 5 (Bad queries are hard to find)**.** *For any $N = p \cdot q$ where $p = 2p' + 1, q = 2q' + 1$ are $(\lambda_{RSA}/2)$-bit safe primes, the following holds: any adversary that makes at most Q queries to the random oracle $H : \{0,1\}^* \to \{0,1\}^\lambda$ will make a bad query with probability at most $3 \cdot Q/2^\lambda$*

The proof of Lemma 5 is a straightforward adaptation of [88, Lemma 1] to the setting of QR_N^+. We provide it in §A.1 in the appendix for the sake of self-containment. As a corollary of Lemma 5 we get a strong soundness guarantee for the proof system (Lemma 6). 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 prescribed proofs for true statements. Since correctness is guaranteed by construction, Theorem 5 follows.

Lemma 6 (Unambiguous Soundness of $\overline{\mathsf{PP}}$ in the random oracle model)**.** *For any $N = p \cdot q$ where $p = 2p' + 1, q = 2q' + 1$ are $(\lambda_{RSA}/2)$-bit safe, no adversary that makes at most Q queries to the random oracle $H : \{0,1\}^* \to \{0,1\}^\lambda$ (but is otherwise computationally-unbounded) will find a proof $\tilde{\pi}(x \xrightarrow{T} y)$ where*

- *$x \in QR_N^\star$ (we let the adversary choose T and x, but require x to be in $QR_N^\star$).*
- *$\overline{\mathsf{PP}}.\mathsf{V}^H(\tilde{\pi}(x \xrightarrow{T} y)) = 1$ (proof verifies)*
- *$\pi(x \xrightarrow{T} x^{2^T}) \neq \tilde{\pi}(x \xrightarrow{T} y)$ (proof is different from the prescribed proof for a true statement)*

except with probability $\leq 3 \cdot Q/2^\lambda$.

Proof. Let *break* denote the event that an adversary that makes at most Q queries to the random oracle finds a proof $\tilde{\pi}(x \xrightarrow{T} y)$ such that

$$\overline{\mathsf{PP}}.\mathsf{V}^H(\tilde{\pi}(x \xrightarrow{T} y)) = 1 \text{ and } \pi(x \xrightarrow{T} x^{2^T}) \neq \tilde{\pi}(x \xrightarrow{T} y).$$

If *bad* denotes the event that a bad query is made, the probability of *break* can be bounded as follows:

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

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

Theorem 5. *In the random oracle model, $\overline{\mathsf{PP}}$ is a* statistically *unambiguosly sound non-interactive proof system for the language corresponding to* IS^+.

3.2.4 Efficiency

Finally, we point out three properties concerning the efficiency of $\overline{\mathsf{PP}}$. 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 RSVL_1 that follows next in §3.3.

Property 1 (Cost of computing proofs)**.** *The computational cost incurred to compute* $\pi(x \xrightarrow{T} y) \leftarrow \overline{\mathsf{PP}}.\mathsf{P}^H(N, x, T, y)$ *is* $T + \mathbf{poly}(\lambda)$ *multiplications in* $(QR_N, \circ)$*, whereas the space required is* $\mathbf{poly}(\lambda)$ *.*

To see this, note that the cost of computing $\overline{\mathsf{PP}}.\mathsf{P}^H(N, x, T, y)$ is dominated by computing the μ_i's, which requires $T/2$ squarings for μ_1, $T/4$ for μ_2 and so on and so forth, for a total of $T - 1$ squarings.

Property 2 (Size of the proof)**.** *The size of a proof* $\pi(x \xrightarrow{T} y)$ *is* $O(\lambda_{RSA} \cdot \log(T))$ *bits.*

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

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

$$r := H(\mu = \mu_1, I, y, T),\ x' := x^r \circ \mu,\ y' := \mu^r \circ y \tag{3.6}$$

and then, making $T/2 + \mathbf{poly}(\lambda)$ multiplications (by Property 1), compute its proof

$$\pi(x' \xrightarrow{T/2} y') = (N, x', T/2, y', \mu'_1, \ldots, \mu'_t) := \overline{\mathsf{PP}}.\mathsf{P}^H(N, x', T/2, y').$$

From the proof for the merged statement, we can reconstruct the proof for $\pi(x \xrightarrow{T} y)$ as

$$(N, T, x, y, \mu_1, \mu'_1, \ldots, \mu'_t). \tag{3.7}$$

3.3 The Reduction

To construct a hard distribution of RSVL instances, we rely on the hardness of solving Iterated-Squaring$^+$ 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 standard path using the verifier V — in order to construct such a V, we intend to use Pietrzak's non-interactive proof system $\overline{\mathsf{PP}}$ for certifying $y = x^{2^T}$ just described in §3.2. We sketch in §3.3.1 why some simple approaches do not work. The reader, however, can skip these and directly jump to §3.3.2 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; $\Sigma^{\leq t}$ denotes $\cup_{j\in[0,t]}\Sigma^j$ (with the empty string denoted by ε). For a string $a \in \Sigma^t$ and $j \in [0,t]$, $a[j]$ 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 naturally encodes its position — i.e., a node at level $l \in [0,t]$ 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 3.2).
- Finally, we reserve "nodes" to refer only to the vertices of a tree, to avoid conflation with the vertices of the RSVL instance.

3.3.1 Intuition

We consider SVL or RSVL instances where the IS^+ instance (N, x, T) is first sampled as in Assumption 2 with $T = 2^t$.

Attempt 1: inefficient verifier circuit. The first idea is to sample an SVL instance as $(\mathsf{S}, \mathsf{V}, T, x)$ where

$$\begin{aligned} \mathsf{V}((y,i),j) = 1 \quad &\iff \quad j = i, i \leq T \ \text{ and } \ y = x^{2^i} \\ \mathsf{S}((y,i)) \quad &:= \quad \begin{cases} (y,i) & \text{if } i \geq T \\ (y^2, i+1) & \text{otherwise} \end{cases} \end{aligned}$$

The only way to solve this instance is to find the sink of the standard path (x^{2^T}, T). However as this label contains the solution to IS^+, computing it should be hard under Assumption 2. Unfortunately without knowing the group order $\varphi(N)$ we cannot realize V efficiently as computing x^{2^i} requires i squarings.

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

$$\begin{aligned} \mathsf{V}(\tilde{\pi}(x_0 \xrightarrow{i} y), j) = 1 \quad &\iff \quad j = i, i \leq T \ \text{ and } \ \overline{\mathsf{PP}}.\mathsf{V}^H(\tilde{\pi}(x_0 \xrightarrow{i} y)) = 1 \\ \mathsf{S}(\tilde{\pi}(x_0 \xrightarrow{i} y)) \quad &:= \quad \begin{cases} \tilde{\pi}(x_0 \xrightarrow{i} y) & \text{if } \mathsf{V}(\tilde{\pi}(x_0 \xrightarrow{i} y), i) = 0 \text{ or } i = T \\ \pi(x_0 \xrightarrow{i+1} x_{i+1}) & \text{else if } \tilde{\pi}(x_0 \xrightarrow{i} y) = \pi(x_0 \xrightarrow{i} x_i) \\ \text{unspecified} & \text{otherwise.} \end{cases} \end{aligned}$$

Using soundness as stated in Lemma 6, we can argue that an adversary making a $\mathbf{poly}(\lambda)$ number of oracle queries will not be able to find a wrong accepting proof $\tilde{\pi}(x_0 \xrightarrow{i} y)$, i.e.,

$$\tilde{\pi}(x_0 \xrightarrow{i} y) \neq \pi(x_0 \xrightarrow{i} x_i) : \overline{\mathsf{PP}}.\mathsf{V}^H(\tilde{\pi}(x_0 \xrightarrow{i} y)) = 1$$

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

$\mathsf{S}^T(\pi(x_0 \xrightarrow{0} x_0)) = \pi(x_0 \xrightarrow{T} x_T)$ (i.e., a solution of type (i) as per Definition 16), which under Assumption 2 is hard.

Unfortunately now it's not clear how to implement the successor circuit S efficiently, as computing a proof $\pi(x_0 \xrightarrow{i+1} x_{i+1})$ seems to require around i exponentiations even when given a proof for the previous state $\pi(x_0 \xrightarrow{i} x_i)$.[7]

Attempt 3: assuming efficient merging. Assume it is possible, say using a procedure M, to merge proofs $\pi(x \xrightarrow{\ell} y)$ and $\pi(y \xrightarrow{\ell} z)$ into a single proof $\pi(x \xrightarrow{2\ell} z)$ in just $\mathbf{poly}(\lambda)$ steps (rather than $\ell + \mathbf{poly}(\lambda)$ steps required by Property 3). This allows us to define a very simple hard RSVL instance using the *recursive* approach of Valiant [103]: reduce the computation of a proof for time parameter 2ℓ to the computation of two proofs for time parameter ℓ, and then use M to merge. The resulting algorithm F is given in Algorithm 3.3.

Algorithm 3.3 Recursive description of the RSVL instance with efficient merge.

procedure $\mathsf{F}^H(N, x, T)$
oracle $H : \{0,1\}^* \to \{0,1\}^\lambda$ ▷ Random oracle
input IS$^+$ instance $(N, x, T) \in \mathbb{N} \times QR_N^+ \times \mathbb{N}$, where $T = 2^t$
output: $\pi(x \xrightarrow{T} z)$
  if $T = 1$ **then return** $\pi(x \xrightarrow{1} x^2) \leftarrow \overline{\mathsf{PP}}.\mathsf{P}^H(N, x, 1, x^2)$ ▷ Base case
  else
    $\pi(x \xrightarrow{T/2} y) \leftarrow \mathsf{F}^H(N, x, T/2)$ ▷ First recursive call
    $\pi(y \xrightarrow{T/2} z) \leftarrow \mathsf{F}^H(N, y, T/2)$ ▷ Second recursive call
    return $\pi(x \xrightarrow{T} z) \leftarrow \mathsf{M}^H(N, \pi(x \xrightarrow{T/2} y), \pi(y \xrightarrow{T/2} z))$ ▷ Invoke efficient merging
  end if
end procedure

The description of the successor and verifier circuits for the corresponding RSVL instance can now be obtained by simulating $\mathsf{F}^H(N, x, T)$ using stack traces. We will see in §3.3.2 how this can be exactly (and succinctly) accomplished using the tree that captures the execution of F (see Figure 3.1.(a)), and limit below to an informal overview.

Let $x_i := x^{2^i}$ as before. Starting at $\emptyset$ and ending at $\pi(x \xrightarrow{T} x_T)$, the RSVL instance has a standard path of length T. The first few vertices on this path are:

$$\emptyset \to \pi(x_0 \xrightarrow{1} x_1) \to \pi(x_0 \xrightarrow{2} x_2) \to \pi(x_0 \xrightarrow{2} x_2) \| \pi(x_2 \xrightarrow{1} x_3) \to \pi(x_0 \xrightarrow{4} x_4) \to$$
$$\pi(x_0 \xrightarrow{4} x_4) \| \pi(x_4 \xrightarrow{1} x_5) \to \pi(x_0 \xrightarrow{4} x_4) \| \pi(x_4 \xrightarrow{2} x_6) \to \cdots \to \pi(x \xrightarrow{T} x_T).$$

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

[7] As a way around the above problem, instead of assuming that S outputs the proof $\pi(x_0 \xrightarrow{i+1} x_{i+1})$ 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 cannot 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 cannot conclude that solving this instance breaks Assumption 2 or soundness).

valid sequence — looks at the last proof in this sequence, denoted $\pi(a \xrightarrow{\ell} b)$, adds the base proof $\pi(b \xrightarrow{1} b^2)$ 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 $\pi(x_1 \xrightarrow{1} x_2)$ to the previous label $\pi(x_0 \xrightarrow{1} x_1)$ and then doing the merge; the fifth label is obtained by adding $\pi(x_3 \xrightarrow{1} x_4)$ to

$$\pi(x_0 \xrightarrow{2} x_2) \| \pi(x_2 \xrightarrow{1} x_3)$$

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 $\mathbf{poly}(\lambda, \log(T))$. Since $\pi(x \xrightarrow{T} x_T)$ contains the value of $x_T = x^{2^T}$, finding the sink $\pi(x \xrightarrow{T} x_T)$ 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 unambiguous soundness of the underlying proof system.

Attempt 4: exploiting somewhat-efficient merging. As noted in the introduction, the last approach is similar to Valiant's construction of general-purpose IVC [103]. In particular, it can be considered to be its instantiation for a *specific* computation, that of solving IS^+. Unfortunately we do not know how to implement efficient merging for Pietrzak's proof system (without resorting to heavy machinery like in [103]). Hence it is at this point that we deviate from his construction by *trading off* the efficient merging procedure for somewhat-efficient merging that Pietrzak's proof system does allow thanks to Property 3 (i.e., $\pi(x \xrightarrow{\ell} y)$ and $\pi(y \xrightarrow{\ell} z)$ can be merged into $\pi(x \xrightarrow{2\ell} z)$ at the cost of computing a proof $\pi(x' \xrightarrow{\ell} y')$ which still requires $i + \mathbf{poly}(\lambda)$ multiplications). However, as we will explain in the next section, this property already suffices for a reduction. Our main observation in the thesis is that even in some cases where the merging is not efficient, his ideas might still apply.

3.3.2 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 extends the ideas in Algorithm 3.3). 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.

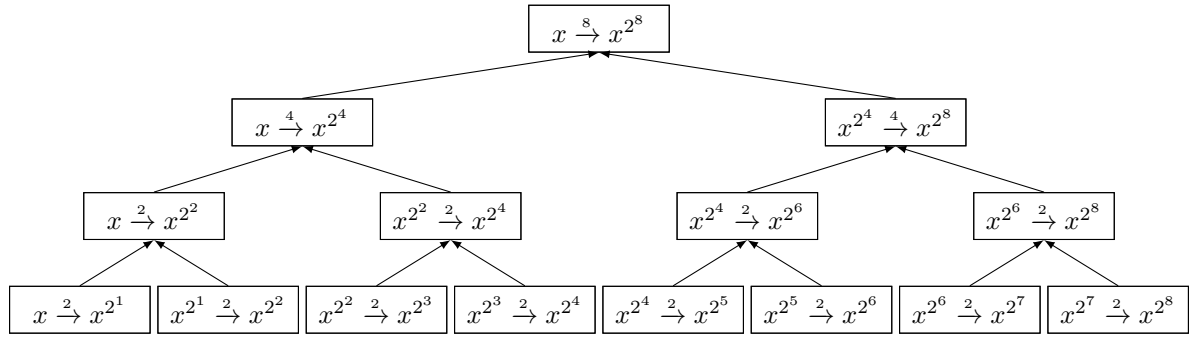
3.3.2.1 Recursive Proof-Merging

The main idea behind our construction is to merge the proofs recursively, exploiting Property 3: given $\pi(x \xrightarrow{\ell} y)$ and $\pi(y \xrightarrow{\ell} z)$, we efficiently reduce the computation of $\pi(x \xrightarrow{2\ell} z)$ to the computation of the proof $\pi(x' \xrightarrow{\ell} y')$ for x', y' as in the merged statement given in eq.(3.6). 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*.

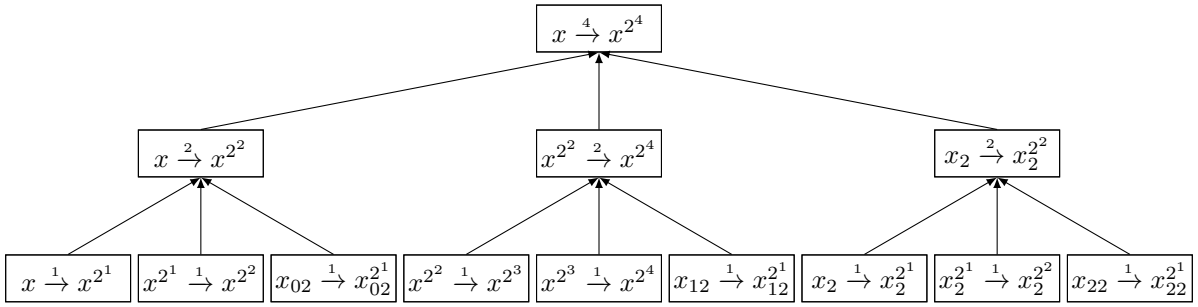
The resulting recursive algorithm F is given in Algorithm 3.4. Note that the way F is structured, the whole computation is being carried out in a verifiable manner: the midpoint and the endpoint are both accompanied by proofs that they are the correct power of x and *only* certified values are being used in the subsequent calls.

Algorithm 3.4 Recursive description of the RSVL instance.

procedure $\mathsf{F}^H(N, x, T)$
  oracle $H : \{0,1\}^* \to \{0,1\}^\lambda$ ▷ Random oracle
  input IS^+ instance $(N, x, T) \in \mathbb{N} \times QR_N^+ \times \mathbb{N}$, where $T = 2^t$
  output $\pi(x \xrightarrow{T} y)$
  if $T = 1$ **then return** $\pi(x \xrightarrow{1} x^2) \leftarrow \overline{\mathsf{PP}}.\mathsf{P}^H(N, x, 1, x^2)$ ▷ Base case
  else
    $\pi(x \xrightarrow{T/2} \mu) \leftarrow \mathsf{F}^H(N, x, T/2)$ ▷ First recursive call
    $\pi(\mu \xrightarrow{T/2} y) \leftarrow \mathsf{F}^H(N, \mu, T/2)$ ▷ Second recursive call
    $r := H(\mu, x, y, T/2)$ and $x' := x^r \circ \mu$ ▷ Reduce $\pi(x \xrightarrow{T} y)$ to $\pi(x' \xrightarrow{T/2} y')$
    $\pi(x' \xrightarrow{T/2} y') \leftarrow \mathsf{F}^H(N, x', T/2)$ ▷ Third recursive call
    Parse $\pi(x' \xrightarrow{T/2} y')$ as $(T/2, x', y', \mu'_1, \ldots, \mu'_{\log(T/2)})$
    return $\pi(x \xrightarrow{T} y) := (T, x, y, \mu, \mu'_1, \ldots, \mu'_{\log(T/2)})$ ▷ Reconstruct $\pi(x \xrightarrow{T} y)$
  end if
end procedure



(a)



(b)

Figure 3.1: (a) The complete binary tree that corresponds to $\mathsf{F}^H(N, x, 8)$ from Algorithm 3.3. (b) The complete ternary tree that corresponds to $\mathsf{F}^H(N, x, 4)$ from Algorithm 3.4. The value of the x_2, x_{02}, x_{12} and x_{22} can be computed using eq.(3.6).

3.3.2.2 Unwinding the Recursion

We obtain the description of successor and verifier circuits for our RSVL instance $\mathsf{RSVL}_1 = (\mathsf{S}, \mathsf{V}, L, x_0)$ by simulating F using stack traces. To this end, we view the execution of $\mathsf{F}^H(N, x, T = 2^t)$ as a complete *ternary* tree τ of depth 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: see Figure 3.1.(b). 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 $\pi(x \xrightarrow{T} x_T)$ (where, if you recall, $x_i := x^{2^i}$), its three children

$$\pi(x \xrightarrow{T/2} x_{T/2}), \pi(x_{T/2} \xrightarrow{T/2} x_T) \text{ and } \pi(x' \xrightarrow{T/2} x'^{2^{T/2}}),$$

where x' is computed as in eq.(3.6), and so on until the leaves which are labelled using base proofs. For example, the tree corresponding to $\mathsf{F}^H(N, x, 4)$ is depicted in Figure 3.1.(b).

RSVL_1 consists of a standard path of length $L = 3^{\log(T)}$ starting at $\emptyset$ and ending at $\pi(x \xrightarrow{T} x_T)$. This corresponds to a *depth-first traversal* of τ. The intermediate vertices can be described using τ thanks to this correspondence: for $i \in \{0,1,2\}^t$, the i-th vertex on the standard path consists of set the of proofs in the stack of $\mathsf{F}^H(N, x, T)$ when its execution begins the recursion at i — *the stack trace at* i, for short.

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 3.4 (and Figure 3.1.(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 $trace(i)$, the standard path in RSVL_1 is defined as

$$\emptyset = \boldsymbol{v}_{0^t} \to \boldsymbol{v}_{0^{t-1}1} \to \boldsymbol{v}_{0^{t-1}2} \to \boldsymbol{v}_{10^{t-1}} \to \cdots \to \boldsymbol{v}_{2^{t-1}1} \to \boldsymbol{v}_{2^t} \to \boldsymbol{v}_\varepsilon = \pi(x \xrightarrow{T} x_T),$$

where $\boldsymbol{v}_i$ denotes $(\pi_j)_{j \in trace(i)}$. Consequently, the label for a vertex consists of at most $2\log(T)$ proofs of the underlying proof system, and we assume that labels with fewer proofs are padded accordingly.

For example, consider the toy RSVL instance from Figure 3.2. For the node 122 (bold red), the path to the root is dashed in red, and thus its inclusive ancestors are $(122, 12, 1, \varepsilon)$. Its trace $trace(122) = (0, 10, 11, 120, 121)$ is in red, and the correct label for the 122-th vertex is thus $\boldsymbol{v}_{122} = (\pi_0, \pi_{10}, \pi_{11}, \pi_{120}, \pi_{121})$ where, for example, $\pi_0 = \pi(x_0 \xrightarrow{4} x_4)$ and $\pi_{10} = \pi(x_4 \xrightarrow{2} x_6)$.

With the standard path defined as above, the successor and verifier functions follow quite logically. The verifier, given as input a vertex $\boldsymbol{v}$ and an index i, ensures that $\boldsymbol{v}$ 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. Both the intuitive and formal descriptions of the circuits V and S are provided in the next section.

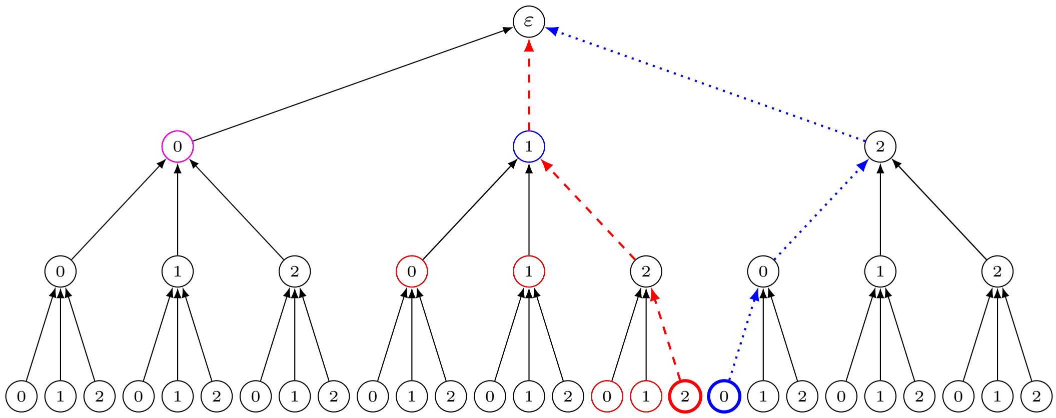


Figure 3.2: A schematic diagram of τ for $\mathsf{F}^H(N, x, 8)$. The resulting RSVL instance is of length 27. The vertex $i = 122$ (resp., $i + 1 = 200$) is highlighted in thick red (resp., blue). The path from $i = 122$ to the root is dashed in red and the vertices in $trace(i)$ are in red. Similarly, the path from $i + 1 = 200$ to the root is dotted in blue and the vertices in $trace(i + 1)$ are in blue. The vertex 0, as it appears in both the traces, is coloured in magenta.

Remark 8 (Comparison with [42])**.** In a concurrent and independent work, Ephraim et al. [42] construct "continuous" verifiable delay functions (see the discussion in Chapter 5) and show how they can be used to construct RSVL instances. Their construction from Iterated-Squaring is similar to ours except that they use a $k + 1$-ary tree (restricted to a particular depth) 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 RSVL instances, and (ii) constant-round Fiat-Shamir Transform to separate $\mathbf{P} \cap \mathbf{PPAD}$ from $\mathbf{NC}$ (which, e.g., implies that there exist easy instances of Nash that are non-parallelisable). Our construction RSVL_1 can be thought of as their construction with k set to 2.

3.3.2.3 The RSVL Instance

Recall that we denote the RSVL instance described next by $\mathsf{RSVL}_1 = (\mathsf{S}, \mathsf{V}, L, \emptyset)$, where $L = 3^{\log(T)}$ for a time parameter $T = 2^t$. Its formal definition is given in Algorithm 3.5, which is complemented by the informal explanation below.

The verifier circuit, $\mathsf{RSVL}_1.\mathsf{V}$**.** On input an index i and a vertex $\boldsymbol{v}$, parsed as a sequence of proofs $(\tilde{\pi}_j)_{j \in trace(i)}$, $\mathsf{RSVL}_1.\mathsf{V}$ ensures that $\boldsymbol{v}$ is a valid stack trace at i by performing a series of checks on the sequence as described below. Recall from Algorithm 3.2 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.

1. $\mathsf{RSVL}_1.\mathsf{V}$ first ensures that each $\tilde{\pi}_j$ is valid by invoking the verifier algorithm $\overline{\mathsf{PP}}.\mathsf{V}$ of the underlying proof system.

2. Second, it 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 3.2 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., $\ell_{121} = 1$).

3. Finally, provided that each proof satisfies the first two conditions, $\mathsf{RSVL}_1.\mathsf{V}$ 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.(3.6). 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 $\boldsymbol{v}_{122} = (\pi_0, \pi_{10}, \pi_{11}, \pi_{120}, \pi_{121})$ is valid if

$$\pi_0 \leftrightarrow \pi_{10} \leftrightarrow, \pi_{11} \rightsquigarrow \pi_{120} \leftrightarrow \pi_{121}, \tag{3.8}$$

where the '$\rightsquigarrow$' 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.(3.6).

The successor circuit, $\mathsf{RSVL}_1.\mathsf{S}$. Given as input a vertex $\boldsymbol{v}$, $\mathsf{RSVL}_1.\mathsf{S}$ first uses a function[8] $index(\cdot)$ to extract the index i that is implicitly embedded in the sequence of proofs in $\boldsymbol{v}$. Next, it confirms whether or not $\boldsymbol{v}$ is the valid i-th vertex on the standard path, i.e. $\boldsymbol{v}_i := (\pi_j)_{j \in trace(i)}$, by invoking the verifier V. In case $\boldsymbol{v}$ is invalid, $\mathsf{RSVL}_1.\mathsf{S}$ forms a self-loop at $\boldsymbol{v}$; otherwise, $\boldsymbol{v}$ is the valid stack trace at i and $\mathsf{RSVL}_1.\mathsf{S}$ utilises it to compute the stack trace at $i + 1$.

$\mathsf{RSVL}_1.\mathsf{S}$ first simulates the next recursion in the pipeline by adding the base proof π_i to $\boldsymbol{v} := (\tilde{\pi}_j)_{j \in trace(i)}$ (cf. lines 21 to 26 in Algorithm 3.5 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 — $\mathsf{RSVL}_1.\mathsf{S}$ has to reconstruct the resulting proof. We denote the node in τ where this recursive call originates by $source(i)$, and it can be obtained by truncating the trailing 2s of i. We refer to Algorithm 3.5 (lines 29 and 30) for the exact procedure used for reconstructing the proof for $source(i)$, but once it possesses this proof, $\mathsf{RSVL}_1.\mathsf{S}$ has all the components of the next vertex on the standard path, the stack trace at $i + 1$.

For example, let us consider the successor circuit applied to $\boldsymbol{v}_{122} := (\pi_0, \pi_{10}, \pi_{11}, \pi_{120}, \pi_{121})$, the $i = 122$-th vertex in the RSVL instance given in Figure 3.2. $\mathsf{RSVL}_1.\mathsf{S}$ first computes π_{122}, and since this concludes the recursive call $\mathsf{F}^H(N, x^4, 4)$ at the (source) vertex 1, $\mathsf{RSVL}_1.\mathsf{S}$ reconstructs the corresponding proof π_1 by merging twice: first it merges π_{120}, π_{121} and π_{122} (using eq.(3.7)) 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 as $\boldsymbol{v}_{200} := (\pi_0, \pi_1)$, and since

[8] To be precise, the function $index(\cdot)$ on input a sequence of proof $\boldsymbol{v}$ computes the index i as follows: it counts the number of proofs of length k in $\boldsymbol{v}$ 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 $\perp$.

Algorithm 3.5 The RSVL instance RSVL_1

procedure $\mathsf{RSVL}_1.\mathsf{V}^H_{N,x,T}(\boldsymbol{v}, i)$ ▷ Verifier circuit
hardwired IS^+ instance $(N, x, T) \in \mathbb{N} \times QR^+_N \times \mathbb{N}$, where $T = 2^t$
oracle $H : \{0,1\}^* \to \{0,1\}^\lambda$ ▷ Random oracle
input
1. Index $i \in \{0,1,2\}^t$
2. Label $\boldsymbol{v}$ parsed as $(\tilde{\pi}_j)_{j \in trace(i)}$

output a bit indicating ACCEPT or REJECT
  if $i > 3^{\log(T)}$ **then return** 0
  Set $x_{next} = x$ ▷ Set starting point of the chain
  for $j \in trace(i)$ **do** ▷ Verify the sequence topologically from left to right
    Parse $\tilde{\pi}_j =: (N, x_j, y_j, \ell_j, \boldsymbol{\mu}_j)$
    if $\overline{\mathsf{PP}}.\mathsf{V}^H(\tilde{\pi}_j) = 0$ or $\ell_j \neq T/2^{|j|}$ **then return** 0 ▷ Checks 1 and 2
    else if $x_j \neq x_{next}$ **then return** 0 ▷ Check 3
    else ▷ Compute the next point on the chain
      if $j[|j|] = 1$ **then** ▷ case (i): second recursion
        Set $\mu := x_j$ and compute $r = H(\mu, x_{j-1}, y_j, \ell_j)$
        Set $x_{next} = x^r_{j-1} \circ \mu$ ▷ Compute the merged statement
      else Set $x_{next} := y_j$ ▷ case (ii): first or third recursion
      end if
    end if
  end for
  return 1 ▷ Valid stack trace at i
end procedure

procedure $\mathsf{RSVL}_1.\mathsf{S}^H_{N,x,T}(\boldsymbol{v})$ ▷ Successor circuit
hardwired IS^+ instance $(N, x, T) \in \mathbb{N} \times QR^+_N \times \mathbb{N}$, where $T = 2^t$
oracle $H : \{0,1\}^* \to \{0,1\}^\lambda$ ▷ Random oracle
input Label $\boldsymbol{v}$ parsed as $(\tilde{\pi}_j)_{j \in trace(i)}$ where $i := index(\boldsymbol{v})$
output Next label $\boldsymbol{v}_{i+1}$
  if $\mathsf{V}(\boldsymbol{v}, i) = 0$ or $i \geq 3^{\log(T)}$ **then return** $\boldsymbol{v}$ ▷ Invalid trace at i or $i \geq L$: self-loop
  for each $j \in trace(i)$ **do** Parse $\tilde{\pi}_j =: (N, x_j, y_j, \ell_j, \boldsymbol{\mu}_j)$
  if $i[t] = 2$ **then** ▷ Compute start of next base proof
    Set $\mu = y_{i-1}$ and compute $r = H(\mu, x_{i-1}, y_i, 1)$
    Compute $x_{next} = x^r_{i-1} \circ \mu$ ▷ case (i): compute the merged statement
  else
    Let l denote the last index in $trace(i)$ and set $x_{next} = y_l$ ▷ case (ii)
  end if
  Set $\tilde{\pi}_i := \pi_i \leftarrow \overline{\mathsf{PP}}.\mathsf{P}^H(N, x_{next}, 1, x^2_{next})$ ▷ Next base proof
  if $i[t] = 2$ **then** ▷ Merge
    Let $s := source(i)$
    Set $\tilde{\pi}_s := (N, 2\ell_{s0}, x_{s0}, y_{s1}, y_{s0}, y_{s20}, y_{s220}, \ldots, y_{i-2})$ ▷ Reconstruct source proof
  end if
  return $\boldsymbol{v}_{i+1} := (\tilde{\pi}_j)_{j \in trace(i+1)}$ ▷ Return stack trace at $i+1$
end procedure

$trace(200) = (0, 1)$ the newly assembled proof indeed is the stack trace at $i + 1$. However applying $\mathsf{RSVL}_1.\mathsf{S}$ again, as no merging is involved, requires simply adding the base proof π_{200} to the input label (π_0, π_1). That is, the label for the 201-th vertex is $\boldsymbol{v}_{201} := (\pi_0, \pi_1, \pi_{200})$.

3.3.3 Analysis

In this section we state and prove the main theorem in this chapter.

Theorem 6. *For a security parameter $\lambda \in \mathbb{N}$, let (N, x, T) be sampled as in Assumption 2 and*

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

be defined as in Algorithm 3.5, where $H : \{0,1\}^* \to \{0,1\}^\lambda$ is a random oracle and

$$m := m(T, \lambda) = 2\log(T) \cdot (\log(T) + 4) \cdot \lambda_{RSA}.$$

Then $\mathsf{RSVL}_1 := (\mathsf{S}, \mathsf{V}, 3^{\log(T)}, \emptyset)$ constitutes a hard distribution of RSVL *instances.*

Remark 9. Note that Theorem 6 can be interpreted as a *Turing reduction* from IS^+ (of form described above) to RSVL. To come up with a (randomised) Karp reduction would require replacing the random oracle with some standard model hash function. We take a step in this direction in the next chapter (in §4.4).

Proof. First, in Claim 6.1 we show that RSVL_1 is efficient: i.e., S and V are both polynomial-sized circuits. Then, to establish hardness, we show that any adversary that runs in time $\mathbf{poly}(\lambda)$, making up to $Q = Q(\lambda) \in \mathbf{poly}(\lambda)$ queries to the random oracle H, has a negligible probability of success.

Recall that by Definition 16 the adversary can solve an RSVL instance in two ways: find either (i) the real sink, which in our case contains the solution x^{2^T} to IS^+; *or* (ii) a false positive i.e., a pair $(\boldsymbol{v}, i)$ s.t. $\mathsf{V}(\boldsymbol{v}, i) = 1$ while $\mathsf{S}^i(\emptyset) \neq \boldsymbol{v}$.

Let $p(\lambda)$ denote the probability that a $\mathbf{poly}(\lambda)$-time adversary on input (N, x, T) as above finds a type (i) solution: under Assumption 2, $p(\lambda)$ is negligible in λ (even if we put no bound on Q). In Claim 6.2 below we will show that finding a type (ii) solution requires breaking the unambiguous soundness of $\overline{\mathsf{PP}}$ and therefore, by Lemma 6, its probability is at most $4 \cdot Q/2^\lambda$. Therefore, the total probability of the adversary breaking the hardness of RSVL_1 is

$$\Pr[type(i) \vee type(ii)] = \Pr[type(i)] + \Pr[type(ii)] \leq \frac{4 \cdot Q}{2^\lambda} + p(\lambda) \in \mathbf{negl}(\lambda), \tag{3.9}$$

completing the proof.

Claim 6.1. S *and* V *are both efficient, i.e. have size* $\mathbf{poly}(\log(T), \lambda)$ *which is* $\mathbf{poly}(\lambda)$ *for* $T \in [\lambda^{\omega(1)}, 2^\lambda]$.

Proof. As a first step, we show that the size of the input vertices $m(T, \lambda)$ is $\mathbf{poly}(\log(T), \lambda)$. As noted in §3.3.2, 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, we infer from Algorithm 3.2 that

$$m(T, \lambda) \leq 2\log(T) \cdot ((\log(T) + 4) \cdot \lambda),$$

i.e. $m(T, \lambda) \in \mathbf{poly}(\log(T), \lambda)$ as claimed.[9]

Next, let us consider the verifier circuit V given in Algorithm 3.5. Note that the size of V is dominated by the call to $\overline{\mathsf{PP}}.\mathsf{V}$ (line 5) and the group operation $\circ$ for QR_N^+ (line 10) inside the loop (line 3). Since the output of $trace(\cdot)$ consists of at most $2\log(T)$ elements, and $\overline{\mathsf{PP}}.\mathsf{V}$ and $\circ$ are both efficient, the size of V is roughly $2\log(T) \cdot \mathbf{poly}(\log(T), \lambda)$ which is still $\mathbf{poly}(\log(T), \lambda)$.

A similar argument holds for the successor circuit S. □

Claim 6.2. *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 $(\boldsymbol{v}, i)$ s.t. $\mathsf{V}(\boldsymbol{v}, i) = 1$ but $\mathsf{S}^i(\emptyset) \neq \boldsymbol{v}$, is upper bounded by $4 \cdot Q/2^\lambda$.*

Proof. For the event *bad* as defined in Definition 21, the probability that an adversary produces a solution of type (ii) is

$$\begin{aligned}
\Pr[type(ii)] &= \Pr[type(ii) \wedge (bad \vee \neg bad)] \\
&= \Pr[type(ii)|bad] \cdot \Pr[bad] + \Pr[type(ii) \wedge \neg bad] \\
&\leq \Pr[type(ii)|bad] \cdot \Pr[bad] + 1/2^{\lambda_{\mathrm{RSA}}/2} && (3.10) \\
&\leq 3 \cdot Q/2^\lambda + 1/2^\lambda && (3.11) \\
&\leq 4 \cdot Q/2^\lambda.
\end{aligned}$$

The upper bound in eq.(3.11) above directly follows Lemma 5, and we argue below that eq.(3.10) is a consequence of Lemma 6. The properties of $QR_N^\star$ (i.e. the generators of QR_N^+) that were discussed in §3.1.2 will be crucial as it allows repeated application of Lemma 6.

Let's suppose that the adversary outputs a solution $(\boldsymbol{v}, i)$ of type (ii), i.e., $\mathsf{V}(\boldsymbol{v}, i) = 1$ but $\mathsf{S}^i(\emptyset) \neq \boldsymbol{v}$, *without* having made a bad query. Since V accepts, $\boldsymbol{v}$ is of the form $(\tilde{\pi}_j)_{j \in trace(i)}$, and this sequence is guaranteed to be a valid chain starting at x as described in §3.3.2.3. We argue that, provided $x \in QR_N^\star$, the adversary *could not* have output the type (ii) solution $(\boldsymbol{v}, i)$ since such a vertex $\boldsymbol{v}$ would equal $\mathsf{S}^i(\emptyset)$ and hence lie on the standard path leading to a contradiction. Since a random $x \in QR_N^+$ also belongs to $QR_N^\star$ with a overwhelming probability of $1 - 1/2^{\lambda_{\mathrm{RSA}}/2}$ (using eq.(3.4)), the upper bound in eq.(3.10) follows.

Provided $x \in QR_N^\star$ and that the adversary never makes a bad query, let us see why the type (ii) solution it outputs lies on the standard path. Assume that

$$\boldsymbol{v} =: (\tilde{\pi}_j)_{j \in trace(i)} = \{\tilde{\pi}_{j_1}, \tilde{\pi}_{j_2}, \ldots, \tilde{\pi}_{j_\ell}\}.$$

Note that $\tilde{\pi}_{j_1}$ is of the form $\tilde{\pi}(x_1 \xrightarrow{\ell_1} y_1)$, where $x_1 = x$ 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 prescribed proof, as a consequence of the unambiguous soundness of the proof system (Lemma 6), we get $\tilde{\pi}(x \xrightarrow{\ell_1} y_1) = \pi(x \xrightarrow{\ell_1} y_1)$.

[9] This can also be established by analysing Algorithm 3.4. 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 $m(1) \leq 4\lambda$. 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 $m(T) < 8\log^2 T \cdot \lambda \in \mathbf{poly}(\log(T), \lambda)$.

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.(3.8)). In the first case, $x_{j_2} = y_1$ and by the property of the generators given in eq.(3.5), we have $y_1 \in QR_N^\star$. Since $y_1 \in QR_N^\star$, we can again apply Lemma 6 and therefore $\tilde{\pi}_{j_2} = \pi_{j_2}$. As for the second case, let $\tilde{\pi}_{j_2} =: \tilde{\pi}(x_2 \xrightarrow{\ell_2} y_2)$. Since we assume that the adversary did not make bad queries, it is guaranteed that $x_2 \in QR_N^\star$ and, by Lemma 6, we get $\tilde{\pi}_{j_2} = \pi_{j_2}$.

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

□

Chapter 4
PPAD-Hardness and Fiat-Shamir

In this chapter, we present the second construction of hard-on-average distribution on the Relaxed-Sink-of-Verifiable-Line problem, denoted RSVL_2. This construction can be thought of as a strengthening of the result from the previous section. In particular, it can be viewed as an alternative instantiation of the ideas there with two main differences concerning the underlying assumptions. First, the underlying computational problem is switched to from Iterated-Squaring to #SAT and therefore we rely here on the (worst-case) hardness of #**P**, which is weaker than the concrete number theoretic Assumption 2. Secondly, the underlying interactive protocol needs to be switched accordingly to the classical Sumcheck Protocol for #SAT $\in$ **IP** [75]. We describe this protocol, both its original form and the variations required for our construction, in §4.2.

The construction RSVL_2 itself relies also on the recursive proof-merging technique. It is presented in detail in §4.3, albeit a bit different from the previous construction RSVL_1. As for its hardness, we additionally rely on the assumption that Fiat-Shamir Transform is (unambiguously) sound for the Sumcheck Protocol. We show that this is true in the random-oracle model. Moreover in §4.4, using ideas from [27], we show that it also holds under some strong assumptions on fully-homomorphic encryption.

We start off with an informal overview of the chapter.

4.1 Overview

4.1.1 Main Results

We base **PPAD**-hardness on the soundness of the Fiat-Shamir Transform, which we described in §1.3.1. The particular protocol to which we apply the Fiat-Shamir Transform is the *Sumcheck Protocol* by Lund et al. [75], which is an n-round interactive proof for the number of satisfying assignments to a given SAT instance over n variables. We show that solving the End-of-Line problem is no easier than breaking the *soundness* of the non-interactive argument obtained by applying the Fiat-Shamir Transform to the sumcheck protocol. Note that we do not explicitly require (adaptive) *unambiguous* soundness. As we shown later in Proposition 2 (§4.1.2) for the case of Sumcheck Protocol unambiguous soundness reduces to plain soundness.

Theorem 7 (informal)**.** *Solving the* End-of-Line *problem is at least as hard as breaking the (adaptive) soundness of the Fiat-Shamir Transform applied to the Sumcheck Protocol.*

We prove this theorem by constructing an efficiently-sampleable distibution of End-of-Line instances, where solving this distribution requires either breaking the soundness of the Fiat-Shamir Transform, applied to the Sumcheck Protocol or solving a #**P** complete problem (and thus any problem in #**P**). Since breaking the soundness of Fiat-Shamir is

reducible to #SAT (in fact to SAT) it follows that efficiently solving the above distribution is at least as hard as breaking Fiat-Shamir. (We note that, like in the previous chapter, all our theorems apply to the class **CLS** as well.).

On the soundness of Fiat-Shamir. The Fiat-Shamir heuristic is widely used in practice, and constructing hash functions for which the transform is sound is a central and long-standing open question in cryptography. Empirical evidence in support of the soundness of Fiat-Shamir is the fact that it has been successfully "field tested" for over three decades, though it should be mentioned that the context in which the transform was originally proposed focused on constant-round protocols, whereas the Sumcheck Protocol has n rounds.

From a foundational perspective, Goldwasser and Kalai [55] demonstrated theoretical barriers towards the instantiation of Fiat-Shamir in the case of certain arguments (i.e., computationally sound proof systems: see Definition 20). Secure instantiations for proofs (i.e., statistically sound proof systems) such as the Sumcheck Protocol, are an active area of recent research. Several recent works have shown that, under strong cryptographic assumptions, the heuristic is sound when it is applied to certain interactive proofs [68, 29, 27, 31, 87]. For our specific purposes it is sufficient that there exists a specific hash family for which the transform is sound *for the Sumcheck Protocol*. Thus, the family can be "tailored" to the protocol. As far as we know, the application of Fiat-Shamir to sumchecks has only been considered recently, most notably in the "sumcheck style" protocol by Pietrzak [88] from the previous chapter and in very recent work of Canetti et al. [27].

4.1.1.1 Instantiating Fiat-Shamir

We obtain two instantiations of the Fiat-Shamir Transform to the Sumcheck Protocol: first with a random oracle and second with the recent construction of Canetti et al. [27].

Random oracle instantiation. We give supporting evidence that the Fiat-Shamir Transform may retain soundness of the Sumcheck Protocol by proving that this indeed is the case when the hash function in the transform is modeled as a Random Oracle. What we show is in fact stronger, namely that the transformed protocol satisfies *unambiguous soundness*, which is what our reduction actually requires (see §4.1.2 for further discussion). One important consequence is the following.

Theorem 8. *Relative to a random oracle, solving* End-of-Line *is at least as hard as solving* #SAT *(and in particular at least as hard as inverting any one-way function).*

FHE-based instantiation. Canetti et al. [27] construct a hash family for which, under the assumption that any one of a broad class of fully homomorphic encryption (FHE) schemes has almost optimal security against polynomial-time adversaries, the Fiat-Shamir Transform is sound when it is applied to (certain instantiations of) the Sumcheck Protocol. Adapting their results to our setting gives rise to a hard distribution in the class **CLS**.

Theorem 9 (Informal Statement, see Theorem 14)**.** *Assuming that any one of the LWE-based fully homomorphic encryption schemes in the literature (such as [21, 20, 19, 49,*

22]) has optimal security against quasi-polynomial-size key-recovery attacks, and assuming further that the #SAT *problem over* **polylog** *variables is (worst-case) hard for polynomial time algorithms, there exists an efficiently-sampleable hard distribution of* End-of-Line *instances.*

Here and below, by optimal security against quasi-polynomial-size attacks, we mean that every quasi-polynomial-size circuit family breaks the assumption with probability at most $\mathbf{quasipoly}(\lambda)/2^{\lambda}$.

To obtain this result, we need a hash function for which the Fiat-Shamir Transform is sound when it is applied to a Sumcheck Protocol for a hard language. Specifically, we consider the Sumcheck Protocol for a formula with poly-logarithmically many variables. By the random self-reducability of **#P** [74, 54], the assumption that the #SAT problem for formulas with **polylog** variables is (worst-case) hard for polynomial time algorithms, gives rise to a hard #SAT distribution over such formulas.[1]

The results of Canetti et al. [27] do not immediately give a hash function as needed. This is because they consider applying the Fiat-Shamir Transform to *doubly-efficient* interactive proofs, where the honest prover runs in polynomial time.[2] We, on the other hand, need to apply the transform to a sumcheck over a quasi-polynomial number of assignments, where the honest prover runs in quasi-polynomial time. Adapting their results to our setting, we show that assuming almost-optimal security of the FHE scheme for *quasi-polynomial* adversaries implies that the transform is sound for the Sumcheck Protocol we consider. See §4.4 for an exposition on this result and a formal statement of Theorem 9.

Sampling hard instances of EOL. By reducing appropriately chosen one-way functions to #SAT, our result opens up the possibility of sampling hard instances of End-of-Line, whose size is significantly smaller than the ones potentially obtained by reducing from indistinguishability obfuscation (IO) and related assumptions. First, the random oracle can be instantiated heuristically with a concrete practical hash function (e.g., SHA3). The reduction from #SAT is best instantiated with a small SAT instance in which each variable appears in a small constant number of clauses. Hard distributions of such SAT instances arise for example from Goldreich's candidate one-way function [53]. This opens up the possibility, given an efficient reduction from End-of-Line to Nash, of sampling reasonably-sized distributions of games for which solving Nash is heuristically hard and against which existing heuristics (such as the Lemke-Howson algorithm) can be tested.

4.1.2 Techniques

Our main technical contribution is a stateful *incrementally-verifiable* procedure that, given a SAT instance over n variables, counts the number of satisfying assignments. The counting is performed via an exponential sequence of polynomial-time computation steps,

[1] Specifically, there is a distribution over #SAT instances with **polylog** variables and a polynomial-time reduction from solving this distribution with non-negligible probability to solving the problem in the worst case (with high probability). Such a result follows similarly to the worst-case to rare-case reductions in the recent work of Goldreich and Rothblum [54].

[2] In fact, they need a stronger *efficient sampleability* property. A sum-check for a $\mathbf{poly}(m)$-sized function over n variables is sampleable in time $\mathbf{poly}(m, 2^n)$.

where we maintain an intermediate state between consecutive steps. Incremental verifiability means that each intermediate state includes proof of its correctness, and the proof can be updated and verified in time $\mathbf{poly}(n)$. The proofs are based on a non-interactive Sumcheck Protocol obtained using the Fiat-Shamir methodology. The main technical challenge is efficient incremental updates to these proofs. We use this incrementally-verifiable counting procedure to construct, given a #SAT instance, an instance of the Relaxed-Sink-of-Verifiable-Line (RSVL) problem. We show that finding a solution to the RSVL instance requires either breaking the unambiguous soundness of the non-interactive sumcheck, or solving the original #SAT instance. This constitutes the first step of our blueprint. For the second step, i.e. to obtain **PPAD** or **CLS** hardness, we simply invoke Lemma 4. We proceed with a high level explanation of the first step.

Sums and sumcheck proofs. Our incrementally-verifiable construction computes sums of low-degree polynomials over exponential numbers of terms. Fix a finite field $\mathbb{F}$ and an n-variate polynomial $f : \mathbb{F}^n \to \mathbb{F}$ over the field $\mathbb{F}$, where f has degree at most d in each variable (think of d as a constant). We are interested in computing sums of the form:

$$\sum_{\boldsymbol{z} \in \{0,1\}^n} f(\boldsymbol{z}).$$

We are also interested in *sumcheck* proofs, proving that y is the correct value of such a sum. More generally, we consider the tasks of computing, proving and verifying sums where a prefix of the variables are fixed to values $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_j) \in \mathbb{F}^j$. We refer to these as *prefix sums*, or the sum with prefix $\boldsymbol{\beta}$. A sumcheck proof can prove statements of the form:

$$\sum_{\boldsymbol{z} \in \{0,1\}^{n-j}} f(\boldsymbol{\beta}, \boldsymbol{z}) = y,$$

which we refer to as a statement *of size* 2^{n-j}.

Given a SAT formula ϕ over n variables, a claim about the number of satisfying assignments can be expressed as a sumcheck claim over an appropriately chosen field using multilinear extensions [75]. The polynomial f is derived from ϕ, and the individual degree can be as low as 4. For this, we first transform ϕ into a 3SAT-4 formula (a 3CNF where each variable appears in at most 4 clauses), using the (Karp) reduction from counting satisfiable assignments of general CNFs to counting satisfying assignments of 3SAT-4 formulae [102]. A standard arithmetization yields an appropriate polynomial f over the field. In what follows, we use the numbers $\{0, 1, \ldots, d\}$ to refer to the "first" $(d + 1)$ field elements.

Incrementally-verifiable counting. Our incrementally-verifiable counting procedure is given as input the field $\mathbb{F}$ and an n-variate polynomial f over this field (described as an arithmetic circuit of size $\mathbf{poly}(n)$ and degree d). We also consider giving the procedure, as part of its input, a prefix $\boldsymbol{\beta} \in \mathbb{F}^j$. The goal of the procedure is computing the value y of the sum with prefix $\boldsymbol{\beta}$, and a sumcheck proof for this value.

This computation is performed in a sequence of incremental steps. Towards this, we specify two $\mathbf{poly}(n)$-time algorithms: S and V. The procedure S performs single steps, receiving as input the current state, and computing the next state. The *completeness*

requirement is that applying S sequentially for $L = L(n)$ steps, starting at a fixed known initial state s_0, leads to a final state s_L comprised of the correct value y of the sum with prefix $\boldsymbol{\beta}$, as well as a proof π of y's correctness. We use s_t to denote the t-th state along the path from s_0 to s_L. In our construction, each intermediate state s_t includes its index $t \in [1, L]$. Since we are computing the value of an exponential sum, we expect the number of steps L to be exponential. We use $m = m(n)$ to denote a bound on the size of the state (the memory used by this process), and $P = P(n)$ to denote a bound on the size (and verification time) of the final proof π.

Soundness is guaranteed using the verification procedure V, which takes as input a state and accepts or rejects. The *unambiguous soundness* requirement is that it should be intractable for an adversary who is given the input to compute a state s' with index t s.t. $s' \neq s_t$ but V accepts s'. We note that this is a strong soundness requirement, and we use the strength of this guarantee to reduce to the RSVL problem.

An incrementally-verifiable counting procedure as described above directly gives rise to an instance of the RSVL problem, where any solution either specifies the correct count, or describes a state $s' \neq s_t$ that V accepts. We first overview the verifiable counter construction, and close by elaborating on the RSVL problem and on the reduction to it.

A recursive construction. Suppose that $(\mathsf{S}_{n-j}, \mathsf{V}_{n-j})$ can compute sums of size 2^{n-j} in an incrementally-verifiable manner. Suppose further that this computation takes L steps, uses m memory, and has a final proof of size P. We want to recursively construct an incrementally-verifiable procedure $(\mathsf{S}_{n-j+1}, \mathsf{V}_{n-j+1})$ for computing sums of size 2^{n-j+1}, which takes $O(L)$ steps, uses $m + O(P) + \mathbf{poly}(n)$ memory, and has a final proof of size $P + \mathbf{poly}(n)$. If we could do so, then unwinding the recursion would give a procedure for computing sums of size 2^n with $2^{O(n)}$ steps, $\mathbf{poly}(n)$ space and $\mathbf{poly}(n)$ proof size (at the base of the recursion, the trivial procedure $(\mathsf{S}_0, \mathsf{V}_0)$ for computing sums of size 1 takes a single step, uses $\mathbf{poly}(n)$ memory and has an "empty" proof). In this overview we ignore the time needed to verify the proof, but we note that it is closely tied to the size P of the final proof. We note that a similar recursive structure underlies Valiant's incrementally-verifiable computation procedure [103].

To construct $(\mathsf{S}_{n-j+1}, \mathsf{V}_{n-j+1})$, given a prefix $\boldsymbol{\beta} \in \mathbb{F}^{j-1}$, the naive idea is to use S_{n-j} to sequentially compute two sums of size 2^{n-j}. We refer to the process of running S_{n-j} for L steps to compute a prefix sum as *a full execution* of (S_{n-j}). In the naive approach, a full execution of S_{n-j+1} is comprised of two sequential full executions of S_{n-j}: a first execution for computing the sum for prefix $(\boldsymbol{\beta}, 0) \in \mathbb{F}^j$, and a second execution computing the sum for prefix $(\boldsymbol{\beta}, 1) \in \mathbb{F}^j$. The first full execution yields a sum y_0 and a proof π_0. These are carried through in the second full execution, which yields a sum y_1 and a proof π_1. The final result is $y = (y_0 + y_1)$, and a naive proof for this result is the concatenated proof (y_0, π_0, y_1, π_1). We can construct $(\mathsf{S}_{n-j+1}, \mathsf{V}_{n-j+1})$ to implement and verify the above execution, and it follows that if the base procedure was unambiguously sound, then the new procedure will also be unambiguously sound. The number of steps and the memory grow exactly as we would like them to: in particular, the space complexity of the new procedure is indeed roughly $m + P$, since we only need to "remember" the proof and end result from the first execution while performing the second execution. The main issue is that the proof length and verification time have doubled. If we repeat this recursion many times, they will become super-polynomial.

A natural approach is to try and *merge* the proofs: given (y_0, π_0) and (y_1, π_1), to

construct a proof π for the total count $y = (y_0 + y_1)$. Ideally, the merge would be performed in $\mathbf{poly}(n)$ time, and π would be of similar length to π_0 and π_1. This was the approach used in [103], who used strong extractability assumptions to achieve efficient proof-merging (we recall that this construction does not have unambiguous proofs, see above). Our approach is different: we use a (long) *incrementally-verifiable proof-merging procedure*, which is constructed recursively (and is unambiguously sound). Proof merging cannot be performed in $\mathbf{poly}(n)$ time, indeed it requires $O(L)$ steps, but this is fine so long as the merge itself is incrementally-verifiable and does not use too much memory or proof size. To obtain an incrementally-verifiable proof-merging procedure, we show that the proof system we use supports a reduction from proof-merging to incrementally-verifiable counting. In particular, given the counts $\{y_\gamma\}_{\gamma=0}^{d}$ for the $(d+1)$ prefix sums with prefixes $\{(\boldsymbol{\beta}, \gamma)\}_{\gamma=0}^{d}$ (sums of size 2^{n-j}), computing a proof π for the count $y = (y_0+y_1)$ of the sum with prefix $\boldsymbol{\beta}$ (a sum of size 2^{n-j+1}) reduces to computing a single additional prefix sum of size 2^{n-j}. This merge procedure relies heavily on the structure of the non-interactive sumcheck proof system, we give an overview below.

Given the merge procedure, we can detail the recursive construction of $(\mathsf{S}_{n-j+1}, \mathsf{V}_{n-j+1})$. Given a prefix $\boldsymbol{\beta} \in \mathbb{F}^{j-1}$, a full execution of S_{n-j+1} runs $(d+1)$ full executions of S_{n-j}, computing the prefix sums (and proofs) for sums with prefixes $\{(\boldsymbol{\beta}, \gamma)\}_{\gamma=0}^{d}$ (these are sums of size 2^{n-j}). Let $\{(y_\gamma, \pi_\gamma)\}_{\gamma=0}^{d}$ be the (respective) prefix sums and proofs. We then run a final full execution of S_{n-j} to compute a "merged" proof π for the sum with prefix $\boldsymbol{\beta}$ (a sum of size 2^{n-j+1}). Once the merged proof is completed we can "forget" the intermediate values $\{(y_\gamma, \pi_\gamma)\}_{\gamma=0}^{d}$. We end the entire process with a proof that is not much larger than the proofs for sums of size 2^{n-j}. Computing the merged proof boils down to computing an additional sum of size 2^{n-j} with a prefix $(\boldsymbol{\beta}, \sigma)$ for a single field element $\sigma \in \mathbb{F}$. Thus, the number of steps for a full execution of S_{n-j+1} is $O(d \cdot L)$ (recall that d is constant), and the memory used is $m + O(d \cdot P)$. Unwinding the recursion, we obtain an incrementally-verifiable counting procedure which takes $2^{O(n)}$ steps, with $\mathbf{poly}(n)$ memory and proof size.

We proceed to detail the proof system we use, and then describe the reduction from proof-merging to incrementally-verifiable counting.

The non-interactive sumcheck. In the interactive Sumcheck Protocol, an untrusted (and not necessarily efficient) prover wants to convince a verifier that:

$$\sum_{\boldsymbol{z} \in \{0,1\}^n} f(\boldsymbol{z}) = y.$$

The protocol proceeds in n rounds. In the first round, the prover sends a univariate polynomial $\widetilde{g}_1$ obtained by leaving the first variable in f free, and summing over all 2^{n-1} assignments to the remaining variables. Note this univariate polynomial is of degree d, and thus it can be specified by sending its $\mathbb{F}$-*representation*, i.e., its valuations over the first $(d+1)$ field elements, and the prover sends these valuations $\boldsymbol{\alpha}_1 = \{\alpha_{1,\gamma} = \widetilde{g}_1(\gamma)\}_{\gamma=0}^{d}$ as its message. On receiving the $\mathbb{F}$-representation, the verifier interpolates to recover $\widetilde{g}_1$, and checks that this polynomial is consistent with the prover's past claims, i.e., that $\widetilde{g}_1(0) + \widetilde{g}_1(1) = y$ (otherwise the verifier rejects immediately). The verifier then picks a random field element β_1 and sends it to the prover.

More generally, the first i rounds fix polynomials $\widetilde{g}_1, \ldots, \widetilde{g}_i$ and a prefix of field elements $\boldsymbol{\beta} := (\beta_1, \ldots, \beta_j)$. In the $(i+1)$-th round the parties run the same two-message protocol

described above to verify the i-th claim:

$$\sum_{\boldsymbol{z}\in\{0,1\}^{n-i}} f(\boldsymbol{\beta},\boldsymbol{z}) = \widetilde{g}_i(\beta_i).$$

Note that this i-th claim is about the sum with the prefix $\boldsymbol{\beta} = (\beta_1,\ldots,\beta_j)$, which is of size 2^{n-i}. After n rounds, the verifier can simply verify the n-th claim on its own using a single evaluation of f on the point $(\beta_1,\ldots,\beta_n)$. Soundness of the protocol follows from the fact that if the i-th claim is false, then for any $\widetilde{g}_{i+1}$ sent by a cheating prover, w.h.p. over the choice of β_{i+1} the $(i+1)$-th claim will also be false (because of the Schwartz-Zippel Lemma). Unambiguity means that even if the i-th claim is true, then if a cheating prover sends any polynomial $\widetilde{g}_{i+1}$ that is not equal to the "prescribed" polynomial $g_{i+1}(x)$ that would have been sent by the honest prover, then w.h.p. over the choice of β_{i+1} the $(i+1)$-th claim will be false (even though the i-th claim was true!). More generally, we can use the same protocol to verify sums with a fixed prefix $\boldsymbol{\beta}\in\mathbb{F}^j$ for any $j\in\{0,\ldots,n\}$. This requires only $(n-j)$ rounds of interaction.

To make this protocol non-interactive, we use the Fiat-Shamir methodology. Given a hash function H, the prescribed proof for an instance $(\mathbb{F},y,f)$ specifies the prescribed prover's messages $(\boldsymbol{\alpha}_1,\ldots,\boldsymbol{\alpha}_n)$ in a particular execution of the Sumcheck Protocol. The particular execution is specified by computing for each i the verfier's challenge $\beta_i = H(\mathbb{F},y,f,\boldsymbol{\alpha}_1,\beta_1,\ldots,\beta_{i-1},\boldsymbol{\alpha}_i)$. To verify such a proof, the verifier: (i) computes each β_i (using the hash function, as above), and (ii) checks that the sumcheck verifier would accept the input $(\mathbb{F},y,f)$ given the transcript $(\boldsymbol{\alpha}_1,\beta_1,\ldots,\boldsymbol{\alpha}_n,\beta_n)$. We assume that this non-interactive protocol is adaptively unambiguously sound: given the hash function H, no polynomial-time prover can find a false statement $(\mathbb{F},\widetilde{y},f)$ and an accepting proof for that statement. Nor can a cheating prover find a true statement $(\mathbb{F},y,f)$ and an accepting proof that differs from the prescribed one. Similarly to the interactive sumcheck, we can also use the non-interactive sumcheck to verify sums with a fixed prefix $\boldsymbol{\beta}\in\mathbb{F}^j$. In fact, if we define the language appropriately, adaptive soundness of this protocol directly implies adaptive *unambiguous* soundness. We elaborate on this below, see Proposition 2.

Merging proofs by computing a (small) sum. Recall our setting: for a fixed prefix $\boldsymbol{\beta}\in\mathbb{F}^{j-1}$, we have computed the sums with prefixes $\{(\boldsymbol{\beta},\gamma)\in\mathbb{F}^j\}_{\gamma=0}^d$, which have values $\{y_\gamma\}_{\gamma=0}^d$, together with corresponding proofs $\{\pi_\gamma\}_{\gamma=0}^d$ for those values. We want now to compute the proof π for the sum with prefix $\boldsymbol{\beta}$. This proof corresponds to a larger sum, and so it should contain an additional (collapsed) round of interaction. What should the prover's first message in the protocol for this statement be? The first message is comprised of the values of the polynomial g_j, where $g_j(\gamma)$ equals the sum with prefix $(\boldsymbol{\beta},\gamma)$. Thus, the first message is simply the values $\boldsymbol{\alpha}_1 = \{y_\gamma\}_{\gamma=0}^d$. Once we know the prover's first message $\boldsymbol{\alpha}_1$, we can compute the verifier's random challenge σ by applying the Fiat-Shamir hash function to the instance and to $\boldsymbol{\alpha}_1$. To complete a transcript for the non-interactive Sumcheck Protocol (and a proof for the sum with prefix $\boldsymbol{\beta}$), we now need a proof for the next claim, i.e., a proof that:

$$\sum_{\boldsymbol{z}\in\{0,1\}^{n-j}} f(\boldsymbol{\beta},\sigma,\boldsymbol{z}) = g_1(\sigma).$$

In particular, all we need is to compute a sum of size 2^{n-j} with prefix $(\boldsymbol{\beta},\sigma)$, and a proof for this sum. Once the value and proof for this larger sum are computed, we can "forget" the

values and proofs $\{y_\gamma, \pi_\gamma\}_{\gamma=0}^{d}$ that were computed for the prefixes. This completes the reduction from incrementally-verifiable proof-merging to incrementally-verifiable counting.

More generally, we have shown that the sum and proof for a statement of size 2^{n-j+1} can be obtained by computing $(d+1)$ proofs for statements of size 2^{n-j} (with a common prefix of length 2^{j-1}, followed by each one of the first $(d+1)$ field elements), and an additional proof for a final statement of size 2^{n-j} (with the same common prefix, but a different j-th element that depends on the Fiat-Shamir hash function). Note that while sums with boolean prefixes correspond to the counting the number of satisfying assignments in subcubes of the hypercube $\{0,1\}^n$, the sums with prefixes that include elements outside $\{0,1\}$ have no such correspondence and in particular the summands can be arbitrary field elements.

4.1.2.1 From Soundness to Unambiguous Soundness

Recall from Definition 19 that an interactive proof system is *unambiguously sound* if the following holds. The proof system specifies a *prescribed* strategy for the honest prover to follow on YES instances. If, given a YES instance, the prover first deviates from its prescribed strategy in some round, then no matter what strategy it follows in subsequent rounds, the verifier will reject with high probability over its subsequent coin tosses. Note that this is a type of soundness requirement for YES instances. Similarly, we say that a non-interactive argument system is unambiguously (adaptively) sound if there is a prescribed proof for every YES instance, and no efficient cheating prover can come up with a pair $(x, \widetilde{\pi})$ that is accepted by the verifier unless x is a YES instance and $\widetilde{\pi}$ is its prescribed proof.

The Sumcheck Protocol is known to be unambiguously sound [93]. For our results, we need to assume that when the Fiat-Shamir Transform is applied to it, the resulting non-interactive argument is adaptively unambiguously sound. We find the assumption that unambiguous soundness is preserved by the Fiat-Shamir Transform to be a natural one. We present supporting evidence for this assumption by demonstrating that it is true in the random oracle model (Theorem 11). We also show that for a particular instantiation of the Fiat-Shamir Transform (which suffices for **PPAD**-hardness), adaptive unambiguous soundness reduces to standard adaptive soundness. To be more precise, we show that adaptive soundness of the non-interactive Sumcheck Protocol applied to a particular language L_{SC}, defined below, implies adaptive unambiguous soundness for the same language (Proposition 2). Moreover, the protocol's adaptive unambiguous soundness for L_{SC} suffices for (unambiguous) soundness of our incrementally-verifiable computation scheme.

The language L_{SC} is defined over tuples that include an instance to the Sumcheck Protocol, a fixed prefix $(\beta_1, \ldots, \beta_j) \in \mathbb{F}$, and a fixed partial transcript of the Sumcheck Protocol. For $0 \leq j \leq i \leq n$ an instance is of the form

$$I_{SC} := (\mathbb{F}, y, f, (\beta_1, \ldots, \beta_j), \widetilde{\boldsymbol{\alpha}}_{j+1}, \beta_{j+1}, \ldots, \widetilde{\boldsymbol{\alpha}}_i).$$

The language is defined as follows:

1. When $i = j = 0$, this is simply a standard input for the Sumcheck Protocol, and the instance is in the language if and only if indeed the sum of f over all 2^n inputs equals y.

2. For $i = j \geq 1$, the instance is in the language if and only if the sum over all 2^{n-j} assignments with prefix $(\beta_1, \ldots, \beta_j)$ equals y.

3. Otherwise, the instance is in the language if and only if the final prover message $\widetilde{\boldsymbol{\alpha}}_i$ is consistent with the prescribed (honest) prover's message, given the fixed prefix $(\beta_1, \ldots, \beta_j)$ and the verifier's messages $\beta_{j+1}, \ldots, \beta_{i-1}$ (there is no condition on y or on the prior prover messages, only the last one matters).[3] It follows that the verifier ends up in an accepting state if the Sumcheck Protocol (with prefix $(\beta_1, \ldots, \beta_j)$) is carried out from i-th round onwards (even if the sum of f over all 2^n inputs does not equal y.).

With this language in mind, we can view each round of the sumcheck as reducing from a claim that a instance is in L_{SC}, to a claim that a longer instance is in L_{SC}. Soundness means that if the initial claim is false, then w.h.p. the new claim is also false. Unambiguity means that even if the initial instance was in the language, if the prover doesn't follow the prescribed strategy, then w.h.p. over the verifier's choice of β_i, the new instance is not in the language.

For our incrementally-verifiable computation scheme, it suffices to assume that the non-interactive sumcheck is an adaptively unambiguously sound non-interactive argument system for the language L_{SC} (see the full construction in §4.2). The following claim shows that in fact it suffices to assume adaptive soundness, which itself implies unambiguity.

Proposition 2. *If the (non-interactive) Sumcheck Protocol is an adaptively sound argument system for the language* L_{SC}*, then it is also an adaptively unambiguously sound argument system for* L_{SC}*.*

Proof. Assume for contradiction that there exists an adversary A that, given a hash function H, can find with noticeable probability an instance $I_{SC} \in \mathrm{L}_{SC}$, whose prescribed proof is π, and an accepting proof $\widetilde{\pi} \neq \pi$. Let

$$I_{SC} = (\mathbb{F}, y, f, (\beta_1, \ldots, \beta_j), \boldsymbol{\alpha}_{j+1}, \beta_{j+1}, \ldots, \boldsymbol{\alpha}_i),$$

$\widetilde{\pi} = (\widetilde{\boldsymbol{\alpha}}_{i+1}, \ldots, \widetilde{\boldsymbol{\alpha}}_n)$ and $\pi = (\boldsymbol{\alpha}_{i+1}, \ldots, \boldsymbol{\alpha}_n)$. Let $\ell^* \in [1, n-i]$ be the *smallest* index such that $\widetilde{\boldsymbol{\alpha}}_{i+\ell^*} \neq \boldsymbol{\alpha}_{i+\ell^*}$. Note that such a ℓ^* must exist because $\widetilde{\pi} \neq \pi$. For $k \in [i, \ell^* - 1]$, if

$$\widetilde{\beta}_k = H(I_{SC}, \widetilde{\beta}_i, \widetilde{\boldsymbol{\alpha}}_{i+1}, \widetilde{\beta}_{i+1}, \ldots, \widetilde{\boldsymbol{\alpha}}_k),$$

it follows that the instance

$$I^*_{SC} = \left(I_{SC}, \widetilde{\beta}_i, \widetilde{\boldsymbol{\alpha}}_{i+1}, \widetilde{\beta}_{i+1}, \ldots, \widetilde{\boldsymbol{\alpha}}_{i+\ell^*}\right) = \left(\mathbb{F}, y, f, \boldsymbol{\alpha}_1, \beta_1, \ldots, \boldsymbol{\alpha}_i, \widetilde{\beta}_i, \widetilde{\boldsymbol{\alpha}}_{i+1}, \ldots, \widetilde{\boldsymbol{\alpha}}_{i+\ell^*}\right)$$

is a NO instance for L_{SC} since $\widetilde{\boldsymbol{\alpha}}_{i+\ell^*}$ differs from the prescribed value. The proof for this new instance is $\pi^* := (\widetilde{\boldsymbol{\alpha}}_{\ell+1}, \ldots, \widetilde{\boldsymbol{\alpha}}_n)$. By construction, if $\widetilde{\pi}$ is an accepting proof for I_{SC}, then also π^* will be an accepting proof for I^*_{SC}. We can exploit A to break the adaptive soundness of the same argument system by simply guessing ℓ^*, then computing (I^*_{SC}, π^*) as described above and returning it. Given A returns a valid proof, the reduction succeeds with probability at least $1/n$. □

[3]Here, when we refer to the prescribed prover, we are ignoring the fact that the actual claim being proved (i.e. the value of y) might be false, as the prescribed prover in the sum check protocol does not need to use the value y to compute its messages.

4.2 The Sumcheck Protocols

The Sumcheck Protocol was introduced by Lund et al. [75] to show that **#P** is contained in **IP**. In this section, we first recall the original interactive protocol SC and then describe its generalisation GSC (§4.2.2) that is useful in our construction RSVL_2. Intuitively, GSC is an interactive protocol for the language L_{SC} and therefore takes a prefix and a partial transcript of the original Sumcheck Protocol SC as its instance. This is followed by the description of non-interactive protocol $\overline{\mathsf{GSC}}$ obtained by applying the Fiat-Shamir Transform to GSC (§4.2.3).

4.2.1 Sumcheck Protocol

Fix a finite field $\mathbb{F}$ and a subset $\mathbb{H} \subseteq \mathbb{F}$ (usually $\mathbb{H} = \{0, 1\}$). In the original Sumcheck Protocol, a (not necessarily efficient) prover takes as input an n-variate polynomial $f : \mathbb{F}^n \to \mathbb{F}$ of degree at most d in each variable (think of d as a constant significantly smaller than $|\mathbb{F}|$). The prover's goal is to convince a verifier that

$$\sum_{\boldsymbol{z} \in \mathbb{H}^n} f(\boldsymbol{z}) = y,$$

for some value $y \in \mathbb{F}$. The verifier only has oracle access to f, and is given the constant $y \in \mathbb{F}$. The verifier is allowed a single oracle query to f, and runs in time $\mathbf{poly}(n, d, \log(|\mathbb{F}|))$. In Algorithm 4.1, we review the standard Sumcheck Protocol from [75], denoted by

$$\mathsf{SC} = \left(\mathsf{P}(y, f), \mathsf{V}^f(y)\right).$$

P is an interactive Turing machine, and $\mathsf{V}^{(\cdot)}$ is a probabilistic interactive Turing machine with oracle access to $f : \mathbb{F}^n \to \mathbb{F}$. The prover $\mathsf{P}(y, f)$ runs in time $\mathbf{poly}(|\mathbb{F}|^n)$.[4] The verifier $\mathsf{V}^f(y)$ runs in time $\mathbf{poly}(n, \log(|\mathbb{F}|), d)$ and queries the oracle f at a single point. The communication complexity is $\mathbf{poly}(n, \log(|\mathbb{F}|), d)$, and the total number of bits sent from the verifier to the prover is $O(n \cdot \log(|\mathbb{F}|))$. Moreover, this protocol is public-coin; i.e., all the messages sent by the verifier are truly random and consist of the verifier's random coin tosses.

Remark 10 (Representing polynomials)**.** For a finite field $\mathbb{F}$, let $\{0, \ldots, d\}$ denote its first $d + 1$ elements. Given a univariate polynomial g of degree d over $\mathbb{F}$, we say that

$$\boldsymbol{\alpha} := \{\alpha_\gamma = g(\gamma)\}_{\gamma=0}^{d}$$

is an $\mathbb{F}$-*representation* of g. Note that this representation is unique since any two degree d polynomials over $\mathbb{F}$ can agree on at most d values (i.e., by Schwartz-Zippel Lemma). Therefore, given $\boldsymbol{\alpha}$, it is possible to reconstruct g by using standard interpolation techniques (e.g., Lagrange interpolation).[5]

[4] Here we assume the prover's input is a description of the polynomial f, from which f can be computed (on any input) in time $\mathbf{poly}(n)$.

[5] The choice of $\{0, \ldots, d\}$ to represent polynomials is arbitrary and, in principle, one could use any $d + 1$ distinct set of field elements.

Algorithm 4.1 Sumcheck Protocol SC [75]

procedure $\mathsf{P}(y, f)$ ▷ Prover
parameters
1. description of a finite field $\mathbb{F}$ and its subset $\mathbb{H} \subset \mathbb{F}$
2. dimension $n \in \mathbb{N}$ and individual degree $d \in \mathbb{N}$

input
1. field element $y \in \mathbb{F}$
2. n-variate polynomial $f : \mathbb{F}^n \to \mathbb{F}$ of individual degree at most d

for $i \in [1, n]$ **do** ▷ At the start of round i, P knows $\beta_1, \ldots, \beta_{i-1}$
Compute the degree-d univariate polynomial

$$g_i(x) := \sum_{z_{i+1},\ldots,z_n \in \mathbb{H}} f(\beta_1, \ldots, \beta_{i-1}, x, z_{i+1}, \ldots, z_n)$$

Send $g_i(x)$ to V via its $\mathbb{F}$-representation $\boldsymbol{\alpha}_i = \{\alpha_{i,\gamma} = g_i(\gamma)\}_{\gamma=0}^{d}$ ▷ $\mathsf{P} \to \mathsf{V}$
Receive challenge β_i from V ▷ $\mathsf{P} \leftarrow \mathsf{V}$
end for
HALT ▷ The prover halts without any output
end procedure

procedure $\mathsf{V}^f(y)$ ▷ Verifier
parameters same as P
oracle n-variate polynomial $f : \mathbb{F}^n \to \mathbb{F}$ of individual degree at most d
input field element $y \in \mathbb{F}$
output a bit indicating ACCEPT or REJECT
Initialise $y_0 = y$
for $i \in [1, n]$ **do** ▷ At the start of round i, V knows y_{i-1}
Receive $d + 1$ field elements $\widetilde{\boldsymbol{\alpha}}_i = \{\widetilde{\alpha}_{i,\gamma}\}_{\gamma=0}^{d}$ from P ▷ $\mathsf{P} \to \mathsf{V}$
Compute the degree-d polynomial $\widetilde{g}_i$ by interpolating $\widetilde{\boldsymbol{\alpha}}_i$
if $\sum_{x \in \mathbb{H}} \widetilde{g}_i(x) \neq y_{i-1}$ **then return** 0
else
Choose a random element $\beta_i \leftarrow \mathbb{F}$ and set $y_i := \widetilde{g}_i(\beta_i)$ ▷ Next challenge
Send β_i to P ▷ $\mathsf{P} \leftarrow \mathsf{V}$
end if
end for
if $y_n = f(\beta_1, \ldots, \beta_n)$ **then return** 1 ▷ V uses a single oracle call to f
else return 0
end if
end procedure

4.2.2 Generalised Sumcheck Protocol

Recall the definition of the language L_{SC} from §4.1.2.1. The Sumcheck Protocol SC described above can be considered to be a proof system for "plain" L_{SC} — i.e., L_{SC} without a prefix and partial transcript (i.e., $i = j = 0$). In this section, we describe the generalised Sumcheck Protocol GSC for general L_{SC}, i.e. where i and j can be both greater than zero. The protocol is given in Algorithm 4.2, and a few remarks are in order.

1. Firstly, for our construction of RSVL in §4.3 we only require the protocol GSC with prefixes (i.e., $i = j > 0$) — the protocol in its fully generality (i.e., with $0 < j < i \leq n$) is used only in Proposition 2.

2. Secondly, the verifier in GSC is given the polynomial f, unlike oracle-access as in SC. Thus, the verifier's run-time can be $\mathbf{poly}(n, d, \log(|\mathbb{F}|), |f|)$, where $|f|$ refers to the size of the polynomial f under some appropriate representation (e.g., arithmetic circuits).

We show in Theorem 10 that this protocol is an unambiguously-sound interactive proof system for L_{SC}.

Theorem 10. *Consider the language L_{SC} with instances of the form $I_{SC} := (\mathbb{F}, y, f, \boldsymbol{\beta}, \boldsymbol{\tau})$, where y is a element in the field $\mathbb{F}$, $f : \mathbb{F}^n \to \mathbb{F}$ is an n-variate polynomial of degree at most $d < |\mathbb{F}|$ in each variable, $\boldsymbol{\beta} \in \mathbb{F}^j$ is any prefix, and $\boldsymbol{\tau} \in (\mathbb{F}^{d+1} \times \mathbb{F})^{i-j}$ is any fixed partial transcript. The Generalised Sumcheck Protocol GSC, described in Algorithm 4.2, is a $(d(n-i)/|\mathbb{F}|)$-unambiguously sound interactive proof system for L_{SC}. That is, for every I_{SC} it satisfies the following three properties.*

- ***Completeness:*** *If $I_{SC} \in \mathrm{L}_{SC}$,*

$$\Pr_{\beta_{i+1},\ldots,\beta_n \leftarrow \mathbb{F}^{n-i}} [(\mathsf{P}(y, f, \boldsymbol{\beta}, \boldsymbol{\tau}), \mathsf{V}(y, f, \boldsymbol{\beta}, \boldsymbol{\tau})) = 1] = 1.$$

- ***Soundness:*** *If $I_{SC} \notin \mathrm{L}_{SC}$ then for every (computationally unbounded) interactive Turing machine $\widetilde{\mathsf{P}}$,*

$$\Pr_{\beta_{i+1},\ldots,\beta_n \leftarrow \mathbb{F}^{n-i}} \left[\left(\widetilde{\mathsf{P}}(y, f, \boldsymbol{\beta}, \boldsymbol{\tau}), \mathsf{V}(y, f, \boldsymbol{\beta}, \boldsymbol{\tau}) \right) = 1 \right] \leq \frac{d(n-i)}{|\mathbb{F}|}. \tag{4.1}$$

- ***Unambiguity:*** *If $I_{SC} \in \mathrm{L}_{SC}$ then for every (computationally unbounded) interactive Turing machine $\widetilde{\mathsf{P}}$ that deviates from the protocol in some round in $[i+1, n]$, V accepts with probability at most $d(n-i)/|\mathbb{F}|$, where the probability is over the coins of the verifier..*

Proof. We show below that the three properties hold for the plain case (i.e., $i = j = 0$) and in the end sketch how the argument can be easily extended to the general case. Completeness follows from the protocol description. We argue soundness[6] and unambiguity in Claim 10.1 and Claim 10.2 below. In both the claims, we proceed via an induction on the number of variables n. We also make extensive use of the univariate formulation of the Schwartz-Zippel Lemma [99, 106], which states that the probability with which a

[6]The presentation of the proof here closely follows that of a lecture note by Thaler.

Algorithm 4.2 Generalised Sumcheck Protocol GSC

procedure $\mathsf{P}(y, f, \boldsymbol{\beta}, \boldsymbol{\tau})$ ▷ Prover
parameters
1. description of a finite field $\mathbb{F}$ and its subset $\mathbb{H} \subset \mathbb{F}$
2. dimension $n \in \mathbb{N}$ and individual degree $d \in \mathbb{N}$

input
1. field element $y \in \mathbb{F}$
2. n-variate polynomial $f : \mathbb{F}^n \to \mathbb{F}$ of individual degree at most d
3. prefix $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_j) \in \mathbb{F}^j$
4. partial transcript $\boldsymbol{\tau}$ parsed as $\boldsymbol{\alpha}_{j+1}, \beta_{j+1}, \ldots, \boldsymbol{\alpha}_i, \beta_i \in \left(\mathbb{F}^{d+1} \times \mathbb{F}\right)^{i-j}$

for $k \in [i+1, n]$ **do** ▷ At the start of round k, P knows $\beta_1, \ldots, \beta_{k-1}$
Compute the degree-d univariate polynomial

$$g_k(x) := \sum_{z_{k+1}, \ldots, z_n \in \mathbb{H}} f(\beta_1, \ldots, \beta_{k-1}, x, z_{k+1}, \ldots, z_n)$$

Send $g_k(x)$ to V via its $\mathbb{F}$-representation ▷ P → V

$$\boldsymbol{\alpha}_k = \{\alpha_{k,\gamma} = g_k(\gamma)\}_{\gamma=0}^{d}$$

Receive challenge β_i from V ▷ P ← V
end for
HALT ▷ The prover halts without any output
end procedure

procedure $\mathsf{V}(y, f, \boldsymbol{\beta}, \boldsymbol{\tau})$ ▷ Verifier
parameters and **input** same as P
output a bit indicating ACCEPT or REJECT
if $i = j$ **then** $y_i := y$ ▷ Transcript empty; carry out SC with prefix $\boldsymbol{\beta}$
else $y_i := \widetilde{g}_i(\beta_i)$, where $\widetilde{g}_i$ the degree-d interpolation of $\boldsymbol{\alpha}_i$
end if
for $k \in [i+1, n]$ **do** ▷ At the start of round k, V knows y_{k-1}
Receive $d+1$ field elements $\widetilde{\boldsymbol{\alpha}}_k = \{\widetilde{\alpha}_{k,\gamma}\}_{\gamma=0}^{d}$ from P ▷ P → V
Compute the degree-d polynomial $\widetilde{g}_k$ by interpolating $\widetilde{\boldsymbol{\alpha}}_k$
if $\sum_{x \in \mathbb{H}} \widetilde{g}_k(x) \neq y_{k-1}$ **then return** 0
else
Choose a random element $\beta_k \leftarrow \mathbb{F}$ and set $y_k := \widetilde{g}_k(\beta_k)$ ▷ Next challenge
Send β_k to P ▷ P ← V
end if
end for
if $y_n = f(\beta_1, \ldots, \beta_n)$ **then return** 1 ▷ V uses a single call to f
else return 0
end if
end procedure

degree-d univariate polynomial f over a field $\mathbb{F}$ evaluates to zero at a randomly-chosen point in $\mathbb{F}$ is at most $d/|\mathbb{F}|$:

$$\Pr_{x \leftarrow \mathbb{F}}[f(x) = 0] \leq \frac{d}{|\mathbb{F}|}.$$

As a consequence, the probability that two distinct degree-d polynomials f and g over $\mathbb{F}$ agree on a random point is also at most $d/|\mathbb{F}|$:

$$\Pr_{x \leftarrow \mathbb{F}}[f(x) = g(x)] \leq \frac{d}{|\mathbb{F}|}.$$

For two univariate polynomials f and g over $\mathbb{F}$, by $f \equiv g$ we denote that the two polynomials are equivalent in the sense that they agree on all points in $\mathbb{F}$, i.e., $\forall x \in \mathbb{F} : f(x) = g(x)$; similary, $f \not\equiv g$ denotes that f and g are not equivalent.

Claim 10.1. *If $\sum_{\boldsymbol{z} \in \mathbb{H}^n} f(\boldsymbol{z}) \neq y$, then V rejects with probability at least $1 - d \cdot n/|\mathbb{F}|$.*

Proof. **Base case:** In the base case of $n = 1$, there is only one round of interaction and $\widetilde{\mathsf{P}}$ sends an $\mathbb{F}$-representation of some polynomial g_1^*. If $g_1^* \not\equiv g_1$, then by Schwartz-Zippel Lemma the probability that $g_1^*(\beta_1) = g_1(\beta_1)$ is at most $d/|\mathbb{F}|$ over the choice of the challenge $\beta_1 \in \mathbb{F}$. Therefore, V rejects with probability at least $1 - d/|\mathbb{F}|$.

Induction hypothesis: We assume that the soundness guarantee in eq.(4.1) holds for all $(n-1)$-variate polynomials. That is, if $\sum_{z_2,\ldots,z_n \in \mathbb{H}^{n-1}} f(z_2, \ldots, z_n) \neq y$ then V rejects with probability at least $1 - (n-1) \cdot d/|\mathbb{F}|$.

Induction: Suppose that $\widetilde{\mathsf{P}}$ sends an $\mathbb{F}$-representation of some polynomial $g_1^* \not\equiv g_1$ in the first round. Again, by Schwartz-Zippel Lemma, $g_1^*(\beta_1) \neq g_1(\beta_1)$ with probability at least $1 - d/|\mathbb{F}|$ over the choice of β_1. Conditioned on this event, $\widetilde{\mathsf{P}}$ is left to prove the following false claim in the second round:

$$g_1^*(\beta_1) = \sum_{z_2,\ldots,z_n \in \mathbb{H}^{n-1}} f(\beta_1, z_2, \ldots, z_n).$$

Since this is a claim involving $(n-1)$-variate polynomials, by our induction hypothesis, V rejects in one of the subsequent rounds with probability at least $1 - (n-1) \cdot d/|\mathbb{F}|$. Let's denote this event by V_{n-1} and by V_n the overall event that V rejects. It follows that

$$\begin{aligned}
\Pr_{\beta_1,\ldots,\beta_n \leftarrow \mathbb{F}^n}[V_n] &= \Pr_{\beta_1,\ldots,\beta_n \leftarrow \mathbb{F}^n}[g_1^*(\beta_1) \neq g_1(\beta_1) \wedge V_{n-1}] \\
&= \Pr_{\beta_2,\ldots,\beta_n \leftarrow \mathbb{F}^{n-1}}[V_{n-1} | g_1^*(\beta_1) \neq g_1(\beta_1)] \cdot \Pr_{\beta_1 \leftarrow \mathbb{F}}[g_1^*(\beta_1) \neq g_1(\beta_1)] \\
&\geq \left(1 - \frac{d \cdot (n-1)}{|\mathbb{F}|}\right) \cdot \left(1 - \frac{d}{|\mathbb{F}|}\right) \\
&= 1 - \frac{d \cdot (n-1)}{|\mathbb{F}|} - \frac{d}{|\mathbb{F}|} - \frac{d^2 \cdot (n-1)}{|\mathbb{F}|^2} \geq 1 - \frac{d \cdot n}{|\mathbb{F}|}.
\end{aligned}$$

□

Claim 10.2. *Suppose that $\sum_{\boldsymbol{z} \in \mathbb{H}^n} f(\boldsymbol{z}) = y$. If $\widetilde{\mathsf{P}}$ deviates from the prescribed protocol then V rejects with probability at least $1 - d \cdot n/|\mathbb{F}|$.*

Proof. **Base case:** In the base case of $n = 1$, there is only one round of interaction and $\widetilde{\mathsf{P}}$ sends an $\mathbb{F}$-representation of some polynomial g_1^*. If $\widetilde{\mathsf{P}}$ deviates and sends $g_1^* \not\equiv g_1$, then by Schwartz-Zippel Lemma the probability that $g_1^*(\beta_1) = g_1(\beta_1)$ is at most $d/|\mathbb{F}|$ over the choice of the challenge $\beta_1 \in \mathbb{F}$. Therefore, V rejects with probability at least $1 - d/|\mathbb{F}|$.

Induction hypothesis: We assume that ambiguity holds for all $(n-1)$-variate polynomials. That is, if $\sum_{z_2,\ldots,z_n \in \mathbb{H}^{n-1}} f(z_2, \ldots, z_n) = y$ and if $\widetilde{\mathsf{P}}$ deviates from the prescribed strategy in some round in $[1, n-1]$ then V rejects with probability at least $1 - (n-1) \cdot d/|\mathbb{F}|$.

Induction: Suppose that $\widetilde{\mathsf{P}}$ sends an $\mathbb{F}$-representation of some polynomial g_1^* in the first round. There are two (complementary) cases: either $g_1^* \equiv g_1$ or $g_1^* \not\equiv g_1$. Clearly, the first case reduces to the induction hypothesis. In the second case, the probability that V rejects outright in the first round is at least $(1 - d/|\mathbb{F}|)$ by the Schwartz-Zippel Lemma. Putting together, we have

$$\begin{aligned}
\Pr_{\beta_1,\ldots,\beta_n \leftarrow \mathbb{F}^n}[V_n] &= \Pr_{\beta_1,\ldots,\beta_n \leftarrow \mathbb{F}^n}[V_n \wedge (g_1^* \equiv g_1 \vee g_1^* \not\equiv g_1)] \\
&= \Pr_{\beta_1,\ldots,\beta_n \leftarrow \mathbb{F}^n}[V_n | g_1^* \equiv g_1] \cdot \Pr[g_1^* \equiv g_1] + \Pr_{\beta_1,\ldots,\beta_n \leftarrow \mathbb{F}^n}[V_n | g_1^* \not\equiv g_1] \cdot \Pr[g_1^* \not\equiv g_1] \\
&\geq \left(1 - \frac{d \cdot (n-1)}{|\mathbb{F}|}\right) \cdot \Pr[g_1^* \equiv g_1] + \left(1 - \frac{d}{|\mathbb{F}|}\right) \cdot \Pr[g_1^* \not\equiv g_1] \\
&\geq 1 - \frac{d \cdot (n-1)}{|\mathbb{F}|} \cdot \Pr[g_1^* \equiv g_1] - \frac{d}{|\mathbb{F}|} \cdot \Pr[g_1^* \not\equiv g_1] \\
&\geq 1 - \frac{d \cdot (n-1)}{|\mathbb{F}|} - \frac{d}{|\mathbb{F}|} = 1 - \frac{d \cdot n}{|\mathbb{F}|}
\end{aligned}$$

where V_{n-1} and V_n are defined as in Claim 10.1. □

Extending to general L_{SC}. It is easy to see that the claims above can be easily extended to accommodate a prefix $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_j)$. To see why the same applies when considering transcripts, we need to take a closer look at the definition of L_{SC} from §4.1.2.1. Recall that an instance

$$I_{SC} := (\mathbb{F}, y, f, (\beta_1, \ldots, \beta_j), \widetilde{\boldsymbol{\alpha}}_{j+1}, \beta_{j+1}, \ldots, \widetilde{\boldsymbol{\alpha}}_i)$$

belongs to L_{SC} if and only if the final prover message $\widetilde{\boldsymbol{\alpha}}_i$ is consistent with the prescribed (honest) prover's message g_i for the prefix $\beta_1, \ldots, \beta_{i-1}$. To argue soundness, note that if $\widetilde{\boldsymbol{\alpha}}_i$ is inconsistent then, by Schwartz-Zippel Lemma, $\widetilde{\mathsf{P}}$ ends up proving a wrong sum in round $i+1$ with high probability. An analogous argument applies for unambiguity. □

4.2.3 Non-Interactive Sumcheck Protocol

We consider the non-interactive version of the Sumcheck Protocol obtained by applying the Fiat-Shamir Transform to the protocol GSC from Algorithm 4.2. The protocol, denoted $\overline{\mathsf{GSC}}$, is described in Algorithm 4.3. The verifier's "challenges" β_i in $\overline{\mathsf{GSC}}$ are obtained by applying a hash function $H : \{0,1\}^* \to \mathbb{F}$ to the transcript thus far, which is comprised of the instance and the prover's messages up to that round. The prover $\overline{\mathsf{GSC}}.\mathsf{P}$ runs in time $\mathbf{poly}(|\mathbb{F}|^n)$. The verifier $\overline{\mathsf{GSC}}.\mathsf{V}$ runs in time $\mathbf{poly}(n, |f|)$ and space $O(n \cdot \log(|\mathbb{F}|))$.

Algorithm 4.3 Non-Interactive Sumcheck Protocol $\overline{\mathsf{GSC}}$

procedure $\mathsf{P}_H(y, f, \boldsymbol{\beta}, \boldsymbol{\tau})$ ▷ Prover
parameters

1. description of a finite field $\mathbb{F}$ and its subset $\mathbb{H} \subset \mathbb{F}$
2. dimension $n \in \mathbb{N}$ and individual degree $d \in \mathbb{N}$
3. hash function $H : \{0,1\}^* \to \mathbb{F}$

input

1. field element $y \in \mathbb{F}$
2. n-variate polynomial $f : \mathbb{F}^n \to \mathbb{F}$ of individual degree at most d
3. prefix $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_j) \in \mathbb{F}^j$
4. partial transcript $\boldsymbol{\tau}$ parsed as $\boldsymbol{\alpha}_{j+1}, \beta_{j+1}, \ldots, \boldsymbol{\alpha}_i, \beta_i \in \left(\mathbb{F}^{d+1} \times \mathbb{F}\right)^{i-j}$

output Proof π

for $k \in [i+1, n]$ **do**
  Compute the degree-d univariate polynomial

$$g_k(x) := \sum_{z_{k+1},\ldots,z_n \in \mathbb{H}} f(\beta_1, \ldots, \beta_{k-1}, x, z_{k+1}, \ldots, z_n)$$

  Compute its $\mathbb{F}$-representation $\boldsymbol{\alpha}_k = \{\alpha_{k,\gamma} = g_k(\gamma)\}_{\gamma=0}^{d}$
  Compute the challenge ▷ Fiat-Shamir

$$\beta_i = H(\mathbb{F}, y, f, \boldsymbol{\beta}, \boldsymbol{\tau}, \boldsymbol{\alpha}_{i+1}, \beta_{i+1}, \ldots, \beta_{k-1}, \boldsymbol{\alpha}_k)$$

end for
Return $\pi := \{\boldsymbol{\alpha}_{i+1}, \ldots, \boldsymbol{\alpha}_n\}$
end procedure

procedure $\mathsf{V}_H(y, f, \boldsymbol{\beta}, \boldsymbol{\tau}, \pi)$ ▷ Verifier
parameters same as P
input

1. $y, f, \boldsymbol{\beta}, \boldsymbol{\tau}$ as in P
2. Proof π parsed as $\{\widetilde{\boldsymbol{\alpha}}_{i+1}, \ldots, \widetilde{\boldsymbol{\alpha}}_n\} \in (\mathbb{F}^{d+1})^{i-j}$

output a bit indicating ACCEPT or REJECT

if $i = j$ **then** $y_i := y$ ▷ Transcript empty; carry out SC with prefix $\boldsymbol{\beta}$
else $y_i := \widetilde{g}_i(\beta_i)$, where $\widetilde{g}_i$ the degree-d interpolation of $\boldsymbol{\alpha}_i$
end if
for $k \in [i+1, n]$ **do** ▷ At the start of round k, V knows y_{k-1}
  Compute the degree-d polynomial $\widetilde{g}_k$ by interpolating $\widetilde{\boldsymbol{\alpha}}_k$
  if $\sum_{x \in \mathbb{H}} \widetilde{g}_k(x) \neq y_{k-1}$ **then return** 0
  else
    Compute $\beta_i = H(\mathbb{F}, y, f, \boldsymbol{\beta}, \boldsymbol{\tau}, \boldsymbol{\alpha}_{i+1}, \beta_{i+1}, \ldots, \beta_{k-1}, \boldsymbol{\alpha}_k)$ and set $y_k := \widetilde{g}_k(\beta_k)$
  end if
end for
if $y_n = f(\beta_1, \ldots, \beta_n)$ **then return** 1 ▷ V uses a single call to f
else return 0
end if
end procedure

4.2.4 Our Assumption on Fiat-Shamir

The assumption that underlies our main theorem pertains to the soundness of the compiled protocol $\overline{\mathsf{GSC}}$ from Algorithm 4.3. In particular, we assume that the Fiat-Shamir Transform is *unambiguously (adaptively) sound* when applied to the parent interactive Sumcheck Protocol GSC and, as a result, that $\overline{\mathsf{GSC}}$ is an unambiguously (adaptively) sound non-interactive argument system for the language L_{SC}. More formally:

Assumption 3. The Fiat-Shamir Transform is *unambiguously* sound for the Generalised Sumcheck Protocol GSC from Theorem 4.2. In other words, there exists a family of hash functions $\mathcal{H} : \{0,1\}^* \to \mathbb{F}$ such that when Fiat-Shamir Transform is instantiated with (random) $H \leftarrow \mathcal{H}$, the non-interactive Sumcheck Protocol $\overline{\mathsf{GSC}}$ from Theorem 4.3 is (δ, ϵ)-unambiguously (adaptively) sound argument system for the language L_{SC} for some δ and ϵ that are negligible in n, the number of variables.

For the particular instatiation of Fiat-Shamir Transform described in Algorithm 4.3, by Proposition 2, the above assumption can be relaxed to simply requiring (adaptive) soundness. As already pointed out, this suffices for our application in §4.3. In Theorem 11 below, we show that the assumption holds *relative to random oracles*. In fact we directly show the stronger property of unambiguous soundness.

Theorem 11. *Fix a finite field $\mathbb{F}$. Let $\overline{\mathsf{GSC}} := \left(\mathsf{P}^H, \mathsf{V}^H\right)$ denote the protocol from Algorithm 4.3 where the hash function H is instatiated with a random function from the set of all functions $\mathcal{H} : \{0,1\}^* \to \mathbb{F}$. Then $\overline{\mathsf{GSC}}$ is a $(Qd/|\mathbb{F}|)$-unambiguously sound non-interactive proof system for the language L_{SC}, where $Q = Q(n)$ denotes the number of queries made to the random oracle. That is, it satisfies the following three properties.*

- ***Completeness:*** *For every $I_{SC} = (\mathbb{F}, y, f, \boldsymbol{\beta}, \boldsymbol{\tau}) \in \mathrm{L}_{SC}$, where $y \in \mathbb{F}$ is an element in the field $\mathbb{F}$, $f : \mathbb{F}^n \to \mathbb{F}$ is an n-variate polynomial of degree at most $d < |\mathbb{F}|$ in each variable, $\boldsymbol{\beta} \in \mathbb{F}^j$ is a prefix, and $\boldsymbol{\tau} \in (\mathbb{F}^{d+1} \times \mathbb{F})^{i-j}$ is a partial transcript,*

$$\Pr_{H \leftarrow \mathcal{H}}\left[\mathsf{V}^H\left((y, f, \boldsymbol{\beta}, \boldsymbol{\tau}), \mathsf{P}^H(y, f, \boldsymbol{\beta}, \boldsymbol{\tau})\right) = 1\right] = 1.$$

- ***Soundness:*** *For every (computationally unbounded) cheating prover strategy $\widetilde{\mathsf{P}}$ that makes at most Q queries to H,*

$$\Pr_{\substack{H \leftarrow \mathcal{H} \\ ((y,f,\boldsymbol{\beta},\boldsymbol{\tau}),\widetilde{\pi}) \leftarrow \widetilde{\mathsf{P}}^H}}\left[\mathsf{V}^H(y, f, \boldsymbol{\beta}, \boldsymbol{\tau}, \widetilde{\pi}) = 1 \wedge (\mathbb{F}, y, f, \boldsymbol{\beta}, \boldsymbol{\tau}) \notin \mathrm{L}_{SC}\right] \leq \frac{Qd}{|\mathbb{F}|}.$$

- ***Unambiguity:*** *For every (computationally unbounded) cheating prover strategy $\widetilde{\mathsf{P}}$ that makes at most Q queries to H,*

$$\Pr_{\substack{H \leftarrow \mathcal{H} \\ ((y,f,\boldsymbol{\beta},\boldsymbol{\tau}),\widetilde{\pi}) \leftarrow \widetilde{\mathsf{P}}^H \\ \pi \leftarrow \mathsf{P}^H(y,f,\boldsymbol{\beta},\boldsymbol{\tau})}}\left[\mathsf{V}^H((y, f, \boldsymbol{\beta}, \boldsymbol{\tau}), \widetilde{\pi}) = 1 \wedge \widetilde{\pi} \neq \pi \wedge (\mathbb{F}, y, f, \boldsymbol{\beta}, \boldsymbol{\tau}) \in \mathrm{L}_{SC}\right] \leq \frac{Qd}{|\mathbb{F}|}.$$

Proof. A query is a tuple of the form

$$(\mathbb{F}, y, f, \boldsymbol{\beta}, \widetilde{\boldsymbol{\alpha}}_{j+1}, \widetilde{\beta}_{j+1}, \ldots, \widetilde{\beta}_{\ell-1}, \widetilde{\boldsymbol{\alpha}}_{\ell}),$$

where f is a polynomial over the field $\mathbb{F}$, $y \in \mathbb{F}$, $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_j)$ is a prefix, and $\widetilde{\boldsymbol{\alpha}}_k \in \mathbb{F}^{d+1}$ and $\widetilde{\beta}_k \in \mathbb{F}$ for $k \in [j+1, \ell]$ where $\ell \in [1, n-j]$. For a statement $(y, f, \boldsymbol{\beta})$ (either in or not in the language L_{SC}), we consider the output $\boldsymbol{\alpha}_{j+1}, \ldots, \boldsymbol{\alpha}_n$ of the prescribed prover P when invoked on $(y, f, \boldsymbol{\beta})$ and the associated hash values $\beta_{j+1}, \ldots, \beta_{\ell-1}$. Let

$$\beta_\ell := H(\mathbb{F}, y, f, \boldsymbol{\beta}, \boldsymbol{\alpha}_{j+1}, \beta_{j+1}, \ldots, \beta_{\ell-1}, \boldsymbol{\alpha}_\ell) \quad \text{and} \quad \widetilde{\beta}_\ell := H(\mathbb{F}, y, f, \boldsymbol{\beta}, \widetilde{\boldsymbol{\alpha}}_{j+1}, \widetilde{\beta}_{j+1}, \ldots, \widetilde{\boldsymbol{\alpha}}_\ell)).$$

Also, let $g_\ell(x)$ (resp., $\widetilde{g}_\ell(x)$) denote the unique degree-d polynomial obtained by interpolating the field elements in $\boldsymbol{\alpha}_\ell$ (resp., $\widetilde{\boldsymbol{\alpha}}_\ell$). We say that the query

$$(y, f, \boldsymbol{\beta}, \widetilde{\boldsymbol{\alpha}}_{j+1}, \widetilde{\beta}_{j+1}, \ldots, \widetilde{\beta}_{\ell-1}, \widetilde{\boldsymbol{\alpha}}_\ell)$$

is *bad* if

1. $\widetilde{\boldsymbol{\alpha}}_\ell \neq \boldsymbol{\alpha}_\ell$ (which implies that the polynomial $g_\ell(x) \neq \widetilde{g}_\ell(x)$) and
2. $\widetilde{g}_\ell(\widetilde{\beta}_\ell) = g_\ell(\beta_\ell)$.

Since β_ℓ and $\widetilde{\beta}_\ell$ are outputs of a random oracle and the polynomials $g_\ell(x)$, $\widetilde{g}_\ell(x)$ are different, the probability of a particular query being bad is at most $d/|\mathbb{F}|$ by the Schwartz-Zippel lemma. Therefore by a union bound over all the queries, the probability that the adversary made a bad query during its execution is at most $Qd/|\mathbb{F}|$.

Note that in the absence of bad queries, denoted $\neg bad$, an adversary cannot break either soundness or unambiguity of the non-interactive Sumcheck Protocol. It follows that the probability the adversary breaks the soundness or unambiguity is also at most $Qd/|\mathbb{F}|$. More formally, let *break* denote the event that an adversary that makes at most Q queries to the random oracle finds a proof that breaks either the soundness or unambiguity of the non-interactive such check protocol. The probability of *break* can be bounded as follows:

$$\begin{aligned}\Pr[break] &= \Pr[break \wedge (bad \vee \neg bad)] \\ &\leq \Pr[break \wedge bad] + \Pr[break \wedge \neg bad] \\ &= \Pr[break | bad] \cdot \Pr[bad] \leq Qd/|\mathbb{F}|.\end{aligned}$$ □

Remark 11. The proof follows the fact that the random oracle is "collision-intractable" for "sparse" relations, which in the case of Theorem 11 is the relation defined by the transcript. Later, in §4.4, we will see an alternative instantation of collision-intractable hash function for this relation from assumptions on lattices.

4.3 The Reduction

In this section, we present the RSVL instance RSVL_2 constructed using the non-interactive Sumcheck Protocol $\overline{\mathsf{GSC}}$ from §4.2.3 as a building block. The proposed instance counts — *incrementally and verifiably* — the number of satisfying assignments (i.e., the sum) of an n-variate polynomial f with individual degree at most d. To be specific, the standard path in the RSVL instance starts at a fixed initial state $\boldsymbol{s}_0$ (the source) and ends at a final state $\boldsymbol{s}_L$ (the sink) comprised of the sum

$$y = \sum_{\boldsymbol{z} \in \{0,1\}^n} f(\boldsymbol{z}),$$

as well as a proof π of y's correctness. The i-th (intermediate) state along the path from $\boldsymbol{s}_0$ to $\boldsymbol{s}_L$, which we denote by $\boldsymbol{s}_i$, consists of appropriately-chosen prefix sums and associated proofs. (To be precise, each state also includes an index $t \in [0, d+1]^{\leq n}$ that is determined by its counter i.) The successor S performs single steps, receiving as input the current state $\boldsymbol{s}_i$, and computing the next state $\boldsymbol{s}_{i+1}$. The verification procedure V, which takes as input a state $\boldsymbol{s}$ and a counter i and accepts if $\boldsymbol{s}$ is the i-th state.

Since the sink will contain the overall sum y with a proof, any adversary that attempts to solve the RSVL instance by finding a type (i) solution (see, Definition 16) must compute the sum for f, the correctness of which can be verified using the proof. On the other hand, it is intractable for an adversary to find a type (ii) solution (i.e., a false positive $(\boldsymbol{s}', i)$ such that $\boldsymbol{s}' \neq \boldsymbol{s}_i$ but V accepts $\boldsymbol{s}'$ as the i-th vertex on the RSVL line) because of the unambiguous soundness of the non-interactive sumcheck proof system. The above is formally stated in the following theorem.

Theorem 12 (Main Theorem)**.** *For a parameter $n \in \mathbb{N}$, fix a finite field $\mathbb{F}$ of sufficiently large size p (say $O(2^n)$). Let f be an n-variate polynomial over $\mathbb{F}$ of individual degree at most d. Pick a hash function H uniformly at random from a family $\mathcal{H}$. Let*

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

be constructed as in Algorithm 4.6 with

$$m = m(n,d,p) = (d+1)n\log(p) \quad \text{and} \quad L = L(n,d) = \sum_{j \in [0,n]} (d+2)^j.$$

Given an adversary A that solves instances of $\mathsf{RSVL}_2 = (\mathsf{S}, \mathsf{V}, L, (0^n, f(0^n), \emptyset))$ in polynomial time $T_{\mathsf{A}} = T_{\mathsf{A}}(n)$ and a non-negligible probability $\epsilon = \epsilon(n)$, it is possible to either

- *find the sum $\sum_{\boldsymbol{z} \in \{0,1\}^n} f(\boldsymbol{z})$ in time $O(T_{\mathsf{A}})$ with probability ϵ, or*
- *break the unambiguous soundness of the non-interactive Sumcheck Protocol (Assumption 3) with probability $\epsilon/(d+1) \cdot n$.*

The proof of the theorem is given in §4.3.3. As explained in §4.1.1, in order to define a hard distribution of RSVL instances, the polynomial f in RSVL_2 is, for example, the arithmetization of a one-way function. The main technical component of our reduction are the successor circuit S and the verifier circuit V, described in §4.3.1 and §4.3.2. S an V, together, implement the incrementally-verifiable counter for statements of size 2^n. They are defined using a sequence of circuits

$$(\mathsf{S}_n, \mathsf{V}_n), \ldots, (\mathsf{S}_0, \mathsf{V}_0),$$

where $(\mathsf{S}_{n-j+1}, \mathsf{V}_{n-j+1})$ is an incrementally-verifiable counter for statements of size 2^{n-j+1} and is implemented *recursively* using $(\mathsf{S}_{n-j}, \mathsf{V}_{n-j})$. At the base of the recursion, $(\mathsf{S}_0, \mathsf{V}_0)$ computes sums of size 1 and is therefore trivial: it takes a single step, uses $\mathbf{poly}(n)$ memory and has an "empty" proof. The circuits (S, V) simply invoke $(\mathsf{S}_n, \mathsf{V}_n)$.

We implement these procedures using circuits and to ensure that the size of these circuits does not blow up, we have to exploit the recursive structure of Sumcheck Protocol. In our construction, if $(\mathsf{S}_{n-j}, \mathsf{V}_{n-j})$ takes L steps, uses m bits of memory, and generates a final proof of size P bits, then $(\mathsf{S}_{n-j+1}, \mathsf{V}_{n-j+1})$ takes $O(dL)$ steps, uses $m + O(dP) +$

$\mathbf{poly}(n)$ memory, and has a final proof of size $P + \mathbf{poly}(n)$. On unwinding the recursion, it can be shown that (S, V) runs for $2^{O(n)}$ steps, uses $\mathbf{poly}(n)$ space and has proof size of $\mathbf{poly}(n)$. But most importantly S and V are polynomial-sized circuits, and therefore each step can be carried out in $\mathbf{poly}(n)$ time. In other words, we get an RSVL instance describing a directed graph with $2^{O(n)}$ vertices each with a label of size $\mathbf{poly}(n)$, where the successor and verifier functions have efficient descriptions.

4.3.1 The Recursive Construction

The circuits $(\mathsf{S}_{n-j+1}, \mathsf{V}_{n-j+1})$ in our incrementally-verifiable counting procedure have hard-wired into them

1. f, an n-variate polynomial over a field $\mathbb{F}$ of individual degree at most d (described as an arithmetic circuit of size $\mathbf{poly}(n)$ and degree d), and
2. H, the description of a hash function from the family $\mathcal{H}$.

It takes as its input a prefix $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_{j-1}) \in \mathbb{F}^{j-1}$ (and also the transcript $\boldsymbol{\tau}$ as explained below). The goal of the procedure is computing the value y of the sum with prefix $\boldsymbol{\beta}$, along with a sumcheck proof for this value.

In order to describe how $(\mathsf{S}_{n-j+1}, \mathsf{V}_{n-j+1})$ is implemented using $(\mathsf{S}_{n-j}, \mathsf{V}_{n-j})$, we need to take a closer look at the non-interactive Sumcheck Protocol $\overline{\mathsf{GSC}}$ given in Algorithm 4.3. For this application, its suffices to consider the protocol with the transcript being empty (i.e., $j = i$). Suppose that the prover $\overline{\mathsf{GSC}}.\mathsf{P}$ has been invoked on the $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_{j-1})$-prefix sum

$$\sum_{\boldsymbol{z} \in \{0,1\}^{n-j+1}} f(\boldsymbol{\beta}, \boldsymbol{z}) = y_{j-1}, \tag{4.2}$$

which is a statement of size 2^{n-j+1}. At the end of the first iteration, $\overline{\mathsf{GSC}}.\mathsf{P}$ reduces this sumcheck to checking a smaller $(\boldsymbol{\beta}, \sigma)$-prefix sum $y_j = g_j(\sigma)$ of size 2^{n-j}, where $g_j(x)$ is the univariate polynomial

$$\sum_{\boldsymbol{z} \in \{0,1\}^{n-j}} f(\boldsymbol{\beta}, x, \boldsymbol{z})$$

specified by the field elements $\alpha_{j,0} = g_j(0), \ldots, \alpha_{j,d} = g_j(d)$, and σ is the "challenge", i.e., a hash value depending on $\alpha_{j,0}, \ldots, \alpha_{j,d}$.

Now, suppose we are given incrementally-verifiable procedures $(\mathsf{S}_{n-j}, \mathsf{V}_{n-j})$ to compute sums of size 2^{n-j}, which takes $L(n-j)$ steps, uses $m(n-j)$ memory, and has a final proof of size $P(n-j)$ bits. Our construction of $(\mathsf{S}_{n-j+1}, \mathsf{V}_{n-j+1})$ takes $(d+2) \cdot L(n-j)$ steps, uses $m(n-j) + (d+1) \cdot P(n-j)$ memory, and has a final proof of size $P(n-j) + (d+1)\log(p)$ bits (where, if you recall, p denotes the size of the field $\mathbb{F}$). On unwinding the above recursive expressions for L, m and P, we conclude that $(\mathsf{S}_n, \mathsf{V}_n)$ is procedure for computing sums of size 2^n with $2^{O(n)}$ steps and $\mathbf{poly}(n)$ space, and the final proof is of size $\mathbf{poly}(n)$.

To achieve this construction, we exploit the structure of Sumcheck Protocol. Note that the polynomial $g_j(x)$ can itself be recursively computed with proof. To be more precise, for each $\gamma \in [0, d]$, we sequentially run $(\mathsf{S}_{n-j}, \mathsf{V}_{n-j})$ to compute the valuations $\alpha_{j+1,\gamma}$ with a proof certifying the sum

$$\sum_{\boldsymbol{z} \in \{0,1\}^{n-j-1}} f(\boldsymbol{\beta}, \gamma, \boldsymbol{z}) = \alpha_{j+1,\gamma} = g_j(\gamma) \tag{4.3}$$

Once we possess $\alpha_{j+1,0}, \ldots, \alpha_{j+1,d}$ after the $(d+1)$ sequential applications of $(\mathsf{S}_{n-j}, \mathsf{V}_{n-j})$, the challenge σ can be computed and subsequently the prefix-sum

$$\sum_{\boldsymbol{z} \in \{0,1\}^{n-j-1}} f(\boldsymbol{\beta}, \sigma, \boldsymbol{z}) = y_j = g_j(\sigma) \tag{4.4}$$

for the next round can also be computed using $(\mathsf{S}_{n-j}, \mathsf{V}_{n-j})$ in an incrementally-verifiable manner. In other words, we have reduced computing the proof for the $\boldsymbol{\beta}$-prefix sum given in eq.(4.2) to (i) $(d+1)$ *new* sumchecks given in eq.(4.3) concerning the computation of polynomial $g_j(x)$, and (ii) the second iteration of the *original* sumcheck given in eq.(4.4), which serves as an incrementally-verifiable proof-merging procedure. Moreover, all the $(d+2)$ sumchecks above involve work proportional to the computation of sumchecks of size 2^{n-j-1}, and therefore they can be computed using $(\mathsf{S}_{n-j}, \mathsf{V}_{n-j})$.

The working of the procedure $(\mathsf{S}_{n-j+1}, \mathsf{V}_{n-j+1})$ on an input a prefix $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_{j-1})$ can therefore be described on a high level as follows.

1. Compute the polynomial $g_j(x)$, represented by the field elements $\{\alpha_{j,\gamma} = g_j(\gamma)\}_{\gamma=0}^{d}$, incrementally and verifiably by invoking $(\mathsf{S}_{n-j}, \mathsf{V}_{n-j})$ on $(\boldsymbol{\beta}, \gamma)$
2. Compute the $\boldsymbol{\beta}$-prefix sum by adding $(\boldsymbol{\beta}, 0)$- and $(\boldsymbol{\beta}, 1)$-prefix sums $\alpha_{j,0}$ and $\alpha_{j,1}$
3. Calculate the "challenge" σ and compute the partial proof for the original sumcheck: compute the proof for the $(\boldsymbol{\beta}, \sigma)$-prefix sum $g_j(\sigma)$ using $(\mathsf{S}_{n-j}, \mathsf{V}_{n-j})$
4. Obtain the "merged proof" for the $\boldsymbol{\beta}$-prefix sum by appending $\{\alpha_{j,\gamma}\}_{\gamma=0}^{d}$ to the proof for $(\boldsymbol{\beta}, \sigma)$-prefix sum
5. Return the $\boldsymbol{\beta}$-prefix sum with proof

Keeping the above recursive procedure in mind, we proceed to detail the recursive successor and verifier circuits S_{n-j+1} and V_{n-j+1}.

The recursive successor. Recall that $(\mathsf{S}_{n-j+1}, \mathsf{V}_{n-j+1})$, on input a prefix $\boldsymbol{\beta} = (\beta_0, \ldots, \beta_{j-1})$, calls the procedure $(\mathsf{S}_{n-j}, \mathsf{V}_{n-j})$ sequentially $d+2$ times. This results in a sequence of states $\boldsymbol{s}_0, \ldots, \boldsymbol{s}_{L(n-j+1)}$, where $\boldsymbol{s}_{L(n-j+1)}$ is comprised of the sum

$$y = \sum_{\boldsymbol{z} \in \{0,1\}^{n-j+1}} f(\boldsymbol{\beta}, \boldsymbol{z})$$

as well as a proof π of y's correctness. Since the invocations of $(\mathsf{S}_{n-j}, \mathsf{V}_{n-j})$ are sequential, an intermediate state $\boldsymbol{s}_i$ along the path from $\boldsymbol{s}_0$ to $\boldsymbol{s}_{L(n-j+1)}$, is comprised of at most $(d+2)$ "sub-states", one for each invocation of $(\mathsf{S}_{n-j}, \mathsf{V}_{n-j})$.

In more details, each state $\boldsymbol{s}_i$ is associated with an index $t \in [0, d+1]^{\leq (n-j+1)}$ which is determined by the counter i. Loosely speaking, the index t of the i-th state $\boldsymbol{s}_i$ is the i-th vertex in the perfect $(d+2)$-dary tree that the standard depth-first search visits for the last time (see the discussion in §4.3.2 for more details). If $t = (t_1, \ldots, t_\ell)$ (where $\ell \in [0, n-j+1]$) then $\boldsymbol{s}_i$ consists of t_1 sub-states, where

- the first $t_1 - 1$ are *final*, i.e., correspond to full executions of $(\mathsf{S}_{n-j}, \mathsf{V}_{n-j})$, and therefore consist of a single tuple of the form (y, π), and

- the t_1-th sub-state is either final and consists of a single tuple (y, π) as in the previous case *or* is itself *intermediate* (i.e., not corresponding to a full execution of $(\mathsf{S}_{n-j}, \mathsf{V}_{n-j})$) and therefore consists of a sequence of tuples of the form (y, π).

The successor circuit S_{n-j+1}, on input a prefix $\boldsymbol{\beta}$ and the current state $\boldsymbol{s}$ with index t, computes the next state. Depending on the conditions of sub-states in $\boldsymbol{s}$, it takes one of the following actions:

- Case A: The state $\boldsymbol{s}$ consists of $d+2$ final sub-states of $(\mathsf{S}_{n-j}, \mathsf{V}_{n-j})$. Such sub-states contain the information necessary to compute the sum for the prefix $\boldsymbol{\beta}$ and assemble its proof (by merging). As a result, the next state is the final state of $(\mathsf{S}_{n-j+1}, \mathsf{V}_{n-j+1})$.
- Case B: The state $\boldsymbol{s}$ consists of $t_1 < d+2$ final sub-states of $(\mathsf{S}_{n-j}, \mathsf{V}_{n-j})$. In this case, S_{n-j+1} initiates the next (i.e., t_1+1-th) execution of $(\mathsf{S}_{n-j}, \mathsf{V}_{n-j})$.
- Case C: The t_1-th sub-state $\boldsymbol{s}'$ is intermediate. Here, S_{n-j+1} simply calls the successor S_{n-j} to increment $\boldsymbol{s}'$ — and as a result the state $\boldsymbol{s}$ — by one step.

The resulting construction of S_{n-j+1} is formally described in Algorithm 4.4. There we have also addressed a minor detail (which also applies to V_{n-j+1}) that we have up to now brushed under the rug: in order to compute the challenges σ, the counters need some additional information. To this end, S_{n-j+1} receives as an auxiliary argument the protocol transcript that serves as the input to H, denoted by $\boldsymbol{\tau}$. From the description of $\overline{\mathsf{GSC}}$ in Algorithm 4.3, $\boldsymbol{\tau}$ should contain the following information:

1. the *original* statement $(\mathbb{F}, f, y, \boldsymbol{\beta})$, left empty if the sum y has not been computed yet, and
2. a *partial* proof π for $\boldsymbol{\beta}$-prefix sum, which consists of all the values α_i and β_i that have been computed up to the current iteration (as specified in the description of $\overline{\mathsf{GSC}}.\mathsf{P}$).

The recursive verifier. Given as input the prefix $\boldsymbol{\beta}$, the state $\boldsymbol{s}$ and an index t, the verifier V_{n-j+1} ensures that $\boldsymbol{s}$ equals the intermediate state $\boldsymbol{s}_i$ for $(\mathsf{S}_{n-j+1}, \mathsf{V}_{n-j+1})$, where i is the counter that is associated with t.

If the state $\boldsymbol{s}$ is final for $(\mathsf{S}_{n-j+1}, \mathsf{V}_{n-j+1})$ then $t = \varepsilon$ and $\boldsymbol{s}$ is a single tuple of the form (y, π). This can be verified directly by invoking $\overline{\mathsf{GSC}}.\mathsf{V}$.

Otherwise $\boldsymbol{s}$ consists of at most $(d+2)$ (final or intermediate) sub-states of $(\mathsf{S}_{n-j}, \mathsf{V}_{n-j})$, and V_{n-j+1} verifies each of these sub-states by invoking V_{n-j}. To be precise, for each sub-state $\boldsymbol{s}'$ in $\boldsymbol{s}$, V_{n-j+1} first computes

1. the prefix $\boldsymbol{\beta}'$ for $\boldsymbol{s}'$, which is either $(\boldsymbol{\beta}, \gamma)$ for $\gamma \in [0, d]$, or $(\boldsymbol{\beta}, \sigma)$ for a challenge σ, and
2. the index t' for $\boldsymbol{\beta}'$, which is either ε in case $\boldsymbol{s}'$ is final for $(\mathsf{S}_{n-j}, \mathsf{V}_{n-j})$, or $(t_2, \ldots, t_\ell)$ otherwise.

Next, it checks the validity of the sub-state $\boldsymbol{s}'$ recursively by invoking V_{n-j}.

The formal description of V_{n-j+1} is given in Algorithm 4.5. Similarly to the successor S_{n-j+1}, V_{n-j+1} also receives the transcript as input to ensure that it possesses the necessary information to compute the challenges.

Algorithm 4.4 The recursive successor circuit S_{n-j+1} in RSVL_2.

procedure $\mathsf{S}_{n-j+1}(\boldsymbol{\beta}, t, \boldsymbol{s}, \boldsymbol{\tau})$
hardwired
1. an n-variate polynomial f of individual degree d over $\mathbb{F}$
2. the description of a hash function $H \in \mathcal{H}$

input
1. a prefix $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_{j-1}) \in \mathbb{F}^{j-1}$
2. an index $t = (t_1, \ldots, t_\ell) \in [0, d+1]^{\leq (n-j+1)}$
3. a state $\boldsymbol{s} \in \mathbb{F}^*$ parsed as $\{(y_0, \pi_0), (y_1, \pi_1), \ldots, \}$
4. a transcript $\boldsymbol{\tau}$ containing the statement and partial proofs

output the next state
base case $\mathsf{S}_0(\boldsymbol{\beta}, \varepsilon, \emptyset, \emptyset)$: **return** $(f(\boldsymbol{\beta}), \emptyset)$
if $t = \varepsilon$ and $\boldsymbol{s} \neq \emptyset$ **then return** $\boldsymbol{s}$ ▷ already in final state: self-loop
if $t = d+1$ **then return** $\{y_\gamma\}_{\gamma=0}^{d}$ appended to π_{d+1} ▷ Case A: merge
Compute sub-state $\boldsymbol{s}'$ by truncating $\{(y_\gamma, \pi_\gamma)\}_{\gamma=0}^{t_1-1}$ from $\boldsymbol{s}$
Set $t' := (t_2, \ldots, t_\ell)$ as the index of the sub-state $\boldsymbol{s}'$
if $t = d$ or $t_1 = d+1$ **then** ▷ increment/initialise $d+2$-th sub-state
  if $\boldsymbol{\tau} = \emptyset$ **then** initialise with statement $\boldsymbol{\tau} := (\mathbb{F}, y_0 + y_1, f, \boldsymbol{\beta})$ ▷ Case B
  Compute updated transcript $\boldsymbol{\tau}'$ by appending $\{y_\gamma\}_{\gamma=0}^{d}$ to $\boldsymbol{\tau}$ ▷ Case C
  Compute the challenge $\sigma := H(\boldsymbol{\tau})$ and append σ to $\boldsymbol{\tau}$
  Increment/initialise $d+2$-th sub-state: $\boldsymbol{s}' := \mathsf{S}_{n-j}((\boldsymbol{\beta}, \sigma), t', \boldsymbol{s}', \boldsymbol{\tau})$
  Append $\{(y_\gamma, \pi_\gamma)\}_{\gamma=0}^{d}$ back to $\boldsymbol{s}'$ to update $\boldsymbol{s}$
  return it
else ▷ $t \neq d$ and $t_1 \neq d+1$
  if $t \in [0, d-1]$ **then** ▷ Case B: initialise t_1+1-th sub-state
    return $\{(y_\gamma, \pi_\gamma)\}_{\gamma=0}^{t_1}$ appended to $\mathsf{S}_{n-j}((\boldsymbol{\beta}, t_1+1), \varepsilon, \emptyset, \emptyset)$
  else ▷ Case C: increment t_1-th sub-state
    return $\{(y_\gamma, \pi_\gamma)\}_{\gamma=0}^{t_1-1}$ appended to $\mathsf{S}_{n-j}((\boldsymbol{\beta}, t_1), t', \boldsymbol{s}', \emptyset)$
  end if
end if
end procedure

Algorithm 4.5 The recursive verifier circuit V_{n-j+1} in RSVL_2.

procedure $\mathsf{V}_{n-j+1}(\boldsymbol{\beta}, t, \boldsymbol{s}, \boldsymbol{\tau})$
hardwired

1. an n-variate polynomial f of individual degree d over $\mathbb{F}$
2. the description of a hash function $H \in \mathcal{H}$

input

1. a prefix $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_{j-1}) \in \mathbb{F}^{j-1}$
2. an index $t = (t_1, \ldots, t_\ell) \in [0, d+1]^{\leq (n-j+1)}$
3. a state $\boldsymbol{s} \in \mathbb{F}^*$ parsed as a set of pairs of prefix sums and proofs $\{(y_0, \pi_0), (y_1, \pi_1), \ldots, \}$
4. a transcript $\boldsymbol{\tau}$ containing the statement and partial proofs

output a bit indicating ACCEPT or REJECT
base case $\mathsf{V}_0(\boldsymbol{\beta}, \varepsilon, (y, \emptyset), \emptyset)$: Accept if $y = f(\boldsymbol{\beta})$ and reject otherwise

if $t = \varepsilon$ **then return** the bit $b \leftarrow \overline{\mathsf{GSC}}.\mathsf{V}((y_\varepsilon, f, \boldsymbol{\beta}), \pi_\varepsilon)$ ▷ final state
for $\gamma \in [0, t_1 - 1]$ **do** ▷ verify all final sub-states
  if $\mathsf{V}_{n-j}((\boldsymbol{\beta}, \gamma), \varepsilon, (y_\gamma, \pi_\gamma), \emptyset) = 0$ **then return** 0
end for
Compute t_1-th sub-state $\boldsymbol{s}'$ by truncating $\{(y_\gamma, \pi_\gamma)\}_{\gamma=0}^{t_1-1}$ from $\boldsymbol{s}$
Set $t' := (t_2, \ldots, t_\ell)$ as the index of the sub-state $\boldsymbol{s}'$
if $t_1 = d + 1$ **then**
  if $\boldsymbol{\tau} = \emptyset$ **then** initialise with statement $\boldsymbol{\tau} := (\mathbb{F}, y_0 + y_1, f, \boldsymbol{\beta})$
  Compute updated transcript $\boldsymbol{\tau}'$ by appending $\{y_\gamma\}_{\gamma=0}^{d}$ to $\boldsymbol{\tau}$
  Compute the challenge $\sigma := H(\boldsymbol{\tau})$ and append σ to $\boldsymbol{\tau}$
  if $\mathsf{V}_{n-j}((\boldsymbol{\beta}, \sigma), t', \boldsymbol{s}', \boldsymbol{\tau}') = 0$ **then return** 0
else ▷ $t_1 < d + 1$
  if $\mathsf{V}_{n-j}((\boldsymbol{\beta}, t_1), t', \boldsymbol{s}', \emptyset) = 0$ **then return** 0
end if
return 1 ▷ Accept
end procedure

4.3.2 The RSVL Instance

The label of the i-th vertex v_i in the proposed RSVL instance is a tuple $(t, \boldsymbol{s}_i)$, where $\boldsymbol{s}_i \in \mathbb{F}^*$ is a state and $t \in [0, d+1]^{\leq n}$ its index determined by the counter i. To be precise, t is the (address of) i-th vertex in the perfect $(d+2)$-dary tree[7] that the depth-first search *leaves* (i.e. visits for the final time). To map a counter $i \in [1, L]$ to an index $t \in [0, d+1]^{\leq n}$, we use a bijective map $DFS(\cdot)$. Note that this makes sense only if

$$T = \left|[0, d+1]^{\leq n}\right|,$$

which we show below. Its inverse is denoted by $DFS^{-1}(\cdot)$. Thus, the standard RSVL path consists of the sequence of labels

$$(0^n, \boldsymbol{s}_1), (0^{n-1}1, \boldsymbol{s}_2), (0^{n-1}, \boldsymbol{s}_3), \ldots, (11, \boldsymbol{s}_{L-2}), (1, \boldsymbol{s}_{L-1}), (\varepsilon, \boldsymbol{s}_L).$$

The successor and verifier circuits (S, V) for the RSVL instance can now be implemented using $(\mathsf{S}_n, \mathsf{V}_n)$ as shown in Algorithm 4.6. In short, on input a counter i and a label $(t, \boldsymbol{s})$, the verifier circuit V simply checks if t matches $DFS(i)$ and then invokes the recursive verifier circuit V_n on the state $\boldsymbol{s}$ and the index t. On the other hand, on input a label $(t, \boldsymbol{s})$, the successor S first ensures that $\boldsymbol{s}$ is indeed the i-th intermediate state by checking its correctness using V. Then it increments the state by calling the recursive successor function S_n. It returns this new state along with an incremented index as the next label.

Efficiency. We argue that S and V are both $\mathbf{poly}(n)$-sized circuits. Observe that for S_{n-j+1} at most one recursion S_{n-j} is active at any given point. The rest of the operations within S_{n-j+1} (viz., append, truncate, compute the hash etc.) are all efficient. Therefore $|\mathsf{S}_{n-j+1}| < |\mathsf{S}_{n-j}| + \mathbf{poly}(n)$ with $|\mathsf{S}_0| = \mathbf{poly}(n)$, and consequently $|\mathsf{S}| = \mathbf{poly}(n)$. A similar argument holds for the verifier circuit V, taking into account the fact that even though there are multiple active recursive calls within V_{n-j+1}, *all but one* are of depth 1. This is true as the verification of the final sub-states in V_{n-j} is carried out by a single call to $\overline{\mathsf{GSC}}.\mathsf{V}$.

Parameters. Recall that $L(n-j+1)$, $m(n-j+1)$ and $P(n-j+1)$ denote the number of steps, amount of memory and the final proof size for $(\mathsf{S}_{n-j+1}, \mathsf{V}_{n-j+1})$, respectively. Since S_{n-j+1} runs S_{n-j} $(d+2)$ times and then takes one step for merging, we have $L(n-j+1) = (d+2)L(n-j)+1$. By unwinding the recursion with $T(0) = 1$, we get

$$L = L(n, d) = \sum_{j \in [0,n]} (d+2)^j = \left|[0, d+1]^{\leq n}\right|.$$

For simplicity, assume that $t \in \mathbb{F}^{\leq n}$. From the description of $(\mathsf{S}_0, \mathsf{V}_0)$, it is clear that $m(0) = P(0) = \log(p)$ (where p, if you recall, denotes the size of the finite field $\mathbb{F}$). From the description of $(\mathsf{S}_{n-j+1}, \mathsf{V}_{n-j+1})$, we have $m(n-j+1) \leq m(n-j) + (d+1)\log(p)$ and $P(n-j+1) = P(n-j) + (d+1)\log(p)$. On solving the recursion, we get $m = m(n) \leq (d+1)n\log(p)$ and $P = P(n) \leq (d+1)n\log(p)$.

[7] A $(d+2)$-ary tree is called *perfect* if all its interior nodes have $(d+2)$ children and all leaves have the same depth.

Algorithm 4.6 The RSVL instance RSVL_2.

procedure $\mathsf{V}_{f,H}(v, i)$
hardwired

1. an n-variate polynomial f of individual degree d over $\mathbb{F}$
2. the description of a hash function $H \in \mathcal{H}$

input

1. a label v parsed as $(t, \boldsymbol{s}) \in [0, d+1]^{\leq n} \times \mathbb{F}^*$ where t is the index and $\boldsymbol{s}$ the state
2. a counter $i \in [1, L]$

output a bit indicating ACCEPT or REJECT
  if $t \neq DFS(i)$ **then return** 0
  return the bit $b \leftarrow \mathsf{V}_n(\varepsilon, t, \boldsymbol{s}, \emptyset)$
end procedure

procedure $\mathsf{S}_{f,H}(v)$
hardwired same as V
input a label v parsed as $(t, \boldsymbol{s}) \in [0, d+1]^{\leq n} \times \mathbb{F}^*$
output the next label
  Set the index $i := DFS^{-1}(t)$
  if $\mathsf{V}(v, i) = 0$ **then return** v ▷ self-loop
  else return $(DFS(i+1), \mathsf{S}_n(\varepsilon, t, \boldsymbol{s}, \emptyset))$
  end if
end procedure

4.3.3 Analysis

In this section we restate and prove the main theorem of the chapter.

Theorem 12 (Main Theorem)**.** *For a parameter $n \in \mathbb{N}$, fix a finite field $\mathbb{F}$ of sufficiently large size p (say $O(2^n)$). Let f be an n-variate polynomial over $\mathbb{F}$ of individual degree at most d. Pick a hash function H uniformly at random from a family $\mathcal{H}$. Let*

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

be constructed as in Algorithm 4.6 with

$$m = m(n, d, p) = (d+1)n\log(p) \quad \text{and} \quad L = L(n, d) = \sum_{j \in [0,n]} (d+2)^j.$$

Given an adversary A that solves instances of $\mathsf{RSVL}_2 = (\mathsf{S}, \mathsf{V}, L, (0^n, f(0^n), \emptyset))$ in polynomial time $T_{\mathsf{A}} = T_{\mathsf{A}}(n)$ and a non-negligible probability $\epsilon = \epsilon(n)$, it is possible to either

- *find the sum $\sum_{\boldsymbol{z} \in \{0,1\}^n} f(\boldsymbol{z})$ in time $O(T_{\mathsf{A}})$ with probability ϵ, or*
- *break the unambiguous soundness of the non-interactive Sumcheck Protocol (Assumption 3) with probability $\epsilon/(d+1) \cdot n$.*

Corollary 1. *If* #SAT *is hard (in the worst case) then, relative to a random oracle, there exists a hard distribution of* End-of-Line *instances.*

Remark 12. In the original paper [34], we had claimed that the corollary holds just relative to a random oracle, without the additional assumption on #SAT. The argument there was that since $\mathbf{P} \neq \mathbf{NP}$ relative to a random oracle [9], there exist hard instances of #SAT (since $\mathbf{P} \neq \mathbf{\#P}$). However, as pointed out in [12], we overlooked the fact that we need an explicit representation of the #SAT instance in the Sumcheck Protocol. The #SAT instance that results from the above argument may not have an explicit (succinct) representation since it is defined relative to a random oracle.

Proof. Given a SAT formula ϕ over n variables, a claim about the number of satisfying assignments can be expressed as a sumcheck claim over $\mathbb{F}$. The polynomial f is derived from ϕ, and the individual degree can be as low as 4. For this, we first transform ϕ into a 3SAT-4 formula, a 3CNF where each variable appears in at most 4 clauses. A standard arithmetization yields an appropriate polynomial f_ϕ over the field. A reduction from #SAT to RSVL (relative to a random oracle) follows Theorem 12 with $f = f_\phi$, and Theorem 11 with, for example, $p = |\mathbb{F}| = O(2^n)$ and $Q \in \mathbf{poly}(n)$. The reduction from RSVL to EOML given in Lemma 4 completes the corollary. □

Proof (of Theorem 12). The high level structure of the proof is similar to that of Theorem 6. Recall that by Definition 16 the adversary A can solve an RSVL instance in two ways: find either (i) the standard sink *or* (ii) a false positive i.e., a pair (v, i) s.t. $\mathsf{V}(v, i) = 1$ while $\mathsf{S}^i((0^n, f(0^n), \emptyset)) \neq v$.

Finding a type (i) solution is tantamount to solving the #SAT instance f since the sink of the RSVL instance RSVL_2 is $(\varepsilon, \boldsymbol{s}_L = (y_L, \pi_L))$ and contains the number of solutions to f in the form of y_L. In the discussion below we rule out solutions of type (ii) under Assumption 3. Taken together, the theorem follows.

Let v be of the form $(t, \{(\widetilde{y}_1, \widetilde{\pi}_1), \ldots, (\widetilde{y}_\ell, \widetilde{\pi}_\ell)\}$ and let

$$v_i = \mathsf{S}^i(0^n, f(0^n), \emptyset) = (t, \{(y_1, \pi_1), \ldots, (y_\ell, \pi_\ell)\})$$

be the correctly computed vertex. Also, let $\left\{\widetilde{\boldsymbol{\beta}}_1, \ldots, \widetilde{\boldsymbol{\beta}}_\ell\right\}$ and $\{\boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_\ell\}$ denote the associated prefixes. We first establish that there exists at least one index $k^* \in [1, \ell]$ such that the proof $\widetilde{\pi}_{k^*}$ breaks the unambiguous soundness of the non-interactive Sumcheck Protocol.

Assume for contradiction that A violated neither soundness nor unambiguity in the process of finding the type (ii) solution (v, i) s.t. $\mathsf{V}(v, i) = 1$ but $\mathsf{S}^i((0^n, f(0^n), \emptyset)) \neq v$. We show that (v, i) *could not* have been a type (ii) solution. To this end, we establish iteratively from $k = 1$ to $k = \ell$ that $(\widetilde{y}_k, \widetilde{\pi}_k) = (y_k, \pi_k)$.

When V_n is invoked by V on $(\varepsilon, t, \boldsymbol{s}, \emptyset)$ it recurses until $(\widetilde{y}_1, \widetilde{\pi}_1)$ is a final sub-state for some $(\mathsf{S}_{n-j}, \mathsf{V}_{n-j})$. At this point the validity of $(\widetilde{y}_1, \widetilde{\pi}_1)$ is checked using a single call to $\overline{\mathsf{GSC}}.\mathsf{V}$. Recall that it is assumed that neither soundness nor unambiguity was broken. Since $(\widetilde{y}_1, \widetilde{\pi}_1)$ passes verification and $\widetilde{\boldsymbol{\beta}}_1 = \boldsymbol{\beta}_1$, it is guaranteed that the proofs $(\widetilde{y}_1, \widetilde{\pi}_1)$ (y_1, π_1) are for the *same* statement. Therefore $(\widetilde{y}_1, \widetilde{\pi}_1) = (y_1, \pi_1)$ is the correct sub-state.

Assuming that the first $k - 1$ sub-states in v are correct, we will infer that the k-th sub-state is also correct. The first step is to show that $\boldsymbol{\beta}_k$ is correctly computed, for which there are two possibilities. If $\boldsymbol{\beta}_k$ corresponds to a challenge then, since $y_1, \ldots, y_{k-1}$ have been validated, $\boldsymbol{\beta}_k$ is also guaranteed to be computed by hashing the same arguments as in the protocol specification; otherwise, $\boldsymbol{\beta}_k$ is computed by a simple increment, which the

verifier again checks for. Therefore, by the same argument as for $k = 1$, we get $\widetilde{\pi}_k = \pi_k$ and $\widetilde{y}_k = y_k$.

Consequently, all the labels in v are as prescribed by the successor circuit S, contradicting the premise of the lemma that $v \neq \mathsf{S}^i((0^n, f(0^n), \emptyset))$. One therefore concludes that there exists at least one index $k^* \in [1, \ell]$ such that either

- $\overline{\mathsf{GSC}}.\mathsf{V}((\widetilde{y}_{k^*}, f, \widetilde{\boldsymbol{\beta}}_{k^*}), \widetilde{\pi}_{k^*}) = 1$, and $\sum_{\boldsymbol{z} \in \{0,1\}^{n-j_{k^*}}} f(\widetilde{\boldsymbol{\beta}}_{k^*}, \boldsymbol{z}) \neq \widetilde{y}_{k^*}$; *or*
- $\overline{\mathsf{GSC}}.\mathsf{V}((\widetilde{y}_{k^*}, f, \widetilde{\boldsymbol{\beta}}_{k^*}), \widetilde{\pi}_{k^*}) = 1$, and $\widetilde{\pi}_{k^*} \neq \overline{\mathsf{GSC}}.\mathsf{P}(\widetilde{y}_{k^*}, f, \widetilde{\boldsymbol{\beta}}_{k^*})$ where $\overline{\mathsf{GSC}}.\mathsf{P}$ is the prescribed prover.

Here, the first case corresponds to the setting that there is an accepting proof for an incorrect statement, while the second case corresponds to an accepting proof different from that output by the prescribed prover.

Given an adversary A that finds such a vertex i with probability ϵ we can build an adversary A' that, depending on the case we are in, breaks soundness or unambiguity. The strategy for A' in either case is identical and is described below:

1. Run A on the RSVL instance $(\mathsf{S}, \mathsf{V}, L, (0^n, f(0^n), \emptyset))$.
2. Let the output returned by A be of the form $(t, \{(\widetilde{y}_1, \widetilde{\pi}_1), \ldots, (\widetilde{y}_\ell, \widetilde{\pi}_\ell)\})$.
3. Sample a random index $k^* \leftarrow [1, \ell]$.
4. Return $\left(\left(f, \widetilde{\boldsymbol{\beta}}_{k^*}, \widetilde{y}_{k^*}\right), \widetilde{\pi}_{k^*}\right)$, where $\widetilde{\boldsymbol{\beta}}_{k^*}$ is the prefix associated to $(\widetilde{y}_{k^*}, \widetilde{\pi}_{k^*})$.

A' runs for a time that is roughly the same as that of A (i.e., T_{A}). The analysis for A''s success probability is simple and we describe it only for the first case (as the other case is identical). Informally, since there exists an index k^* that breaks soundness, A' succeeds as long as it is able to guess this index correctly. Formally,

$$\begin{aligned}\Pr[\mathsf{A}' \text{ succeeds}] &\geq \Pr[\mathsf{A} \text{ succeeds}] \cdot \Pr[k^* \text{ is a correct guess} | \mathsf{A}' \text{ succeeds}] \\ &\geq \epsilon \cdot \frac{1}{(d+1) \cdot n}\end{aligned}$$

The first inequality follows from the fact that if A succeeds, there is at least one index k^* such that $\overline{\mathsf{GSC}}.\mathsf{V}((\widetilde{y}_{k^*}, f, \widetilde{\boldsymbol{\beta}}_{k^*}), \widetilde{\pi}_{k^*}) = 1$, and $\sum_{\boldsymbol{z} \in \{0,1\}^{n-j_{k^*}}} f(\widetilde{\boldsymbol{\beta}}_{k^*}, \boldsymbol{z}) \neq \widetilde{y}_{k^*}$. The second inequality follows from the fact that a label contains at most $M(n) \leq (d+1) \cdot n$ tuples of the form (y, π). □

4.4 Instantiating Fiat-Shamir

There has been exciting recent progress [28, 68, 29, 27, 29, 31, 87] on instantiating the Fiat-Shamir Transform using *correlation-intractable hash functions* (CIHFs) [30]. In this section, we discuss the applicability of these results to our setting. We observe that the results in [27] can be extended to our setting, yielding a hash family for which the Fiat-Shamir Transform is sound when applied to the Sumcheck Protocol over **polylog**

variables, albeit under *quasi-polynomial* variants of the (strong) assumptions made in that work.

Our starting point is the application of the Fiat-Shamir Transform in [27] to construct publicly verifiable succinct arguments (pv-SNARGs). We note that while the assumptions required when the Fiat-Shamir Transform is used to construct Non-interactive Zero Knowledge (NIZK) proofs are significantly more standard (e.g., plain LWE in [87]), these results hold limited relevance to our setting. This is because the time required to evaluate those hash functions is as large as the time needed to compute the (honest) prover's messages in the interactive proof protocol. In our context, this means that evaluating the hash function would take super-polynomial time since the prover in the Sumcheck Protocol runs in time exponential in the number of parameters.

The subsequent text is (to a large extent) adapted from [27], with several (important) changes that are needed to obtain results in our setting, with the notation adapted. We highlight changes in the assumptions and theorem statements. For a comprehensive discussion, we refer the reader to [27].

4.4.1 Collision-Intractable Hash Functions

We begin with definitions, starting with the notion of *correlation-intractability* [30] that we require for our application. A function is *quasi-polynomial* in a parameter λ if it is of the form $\lambda^{\mathbf{polylog}(\lambda)}$. We denote the set of all quasi-polynomial functions in λ by $\mathbf{quasipoly}(\lambda)$.

Definition 22. For a relation ensemble $\mathcal{R} = \{\mathcal{R}_\lambda \subseteq \mathcal{X}_\lambda \times \mathcal{Y}_\lambda\}_{\lambda\in\mathbb{N}}$, a hash family $\mathcal{H} = \{H_\lambda : \mathcal{I}_\lambda \times \mathcal{X}_\lambda \to \mathcal{Y}_\lambda\}_{\lambda\in\mathbb{N}}$ is said to be $\mathcal{R}$-correlation-intractable against *quasi-polynomial* adversaries if for every quasi-polynomial-size $\mathsf{A} = \{\mathsf{A}_\lambda\}_{\lambda\in\mathbb{N}}$, there exists an ϵ such that

$$\Pr_{\substack{I\leftarrow\mathcal{I}_\lambda \\ x\leftarrow\mathsf{A}_\lambda(I)}} [(x, H_\lambda(I, x)) \in \mathcal{R}_\lambda] \leq \epsilon$$

For the context of our work, we will need ϵ to be a negligible function. We want to guarantee such a property in the standard model. But even in the random oracle model, this only makes sense for relations $\mathcal{R}$ that are *sparse*, which we formalize below.

Definition 23 (Sparsity)**.** For any relation ensemble $\mathcal{R} = \{\mathcal{R}_\lambda \subseteq \mathcal{X}_\lambda \times \mathcal{Y}_\lambda\}$, we say that $\mathcal{R}$ is ρ-*sparse* if for all $\lambda \in \mathbb{N}$ and any $x \in \mathcal{X}_\lambda$,

$$\Pr_{y\leftarrow\mathcal{Y}_\lambda} [(x, y) \in \mathcal{R}_\lambda] \leq \rho(\lambda).$$

Remark 13. When we talk about correlation-intractability with respect to quasi-polynomial time adversaries, it is not sufficient for ρ to be negligible. We will in fact require ρ to be smaller than any inverse quasi-polynomial function.

We will need the ability to *sample* from the relation $\mathcal{R}$. In fact, it is sufficient to be able to *approximately sample* from the relation. We begin by defining what it means for a distribution to be approximated.

Definition 24. A distribution P *multiplicatively* ϵ-*approximates* a distribution Q if for all outcomes w, it holds that

$$\Pr_{x\leftarrow P} [x = w] \geq \epsilon \cdot \Pr_{x\leftarrow Q} [x = w] .$$

We proceed to formalize the notion of *approximately sampling from a relation* $\mathcal{R}$.

Definition 25 (Approximate Sampleability of Relations)**.** A relation ensemble $\mathcal{R} = \{\mathcal{R}_\lambda \subseteq \mathcal{X}_\lambda \times \mathcal{Y}_\lambda\}_{\lambda\in\mathbb{N}}$ is non-uniformly ϵ-*approximately sampleable* if there is a circuit ensemble $\{\mathsf{R}_\lambda\}_{\lambda\in\mathbb{N}}$ such that for every $x \in \mathcal{X}_\lambda$, the distribution $\mathsf{R}_\lambda(x)$ multiplicatively ϵ-approximates the uniform distribution on the (by assumption, non-empty) set

$$\{y' \in \mathcal{Y}_\lambda : (x, y') \in \mathcal{R}_\lambda\}.$$

We say that $\mathcal{R}$ is (non-uniformly) *efficiently* approximately sampleable if it is non-uniformly ϵ-approximately sampleable for some $\epsilon \geq 1/\mathbf{poly}(\lambda)$.

For our application, we need relations where the sampling function R runs in quasi-polynomial time.

4.4.1.1 CIHFs from Key-Dependent Message Security

Next we present the construction of CIHFs from secret-key encryption schemes that are key-dependant message (KDM) secure due to Canetti et al. [29]. Intuitively, the hashing key is a ciphertext and messages (to be hashed) are interpreted as decryption keys. The hashing is then performed by decrypting the hash key/ciphertexts under the message/decryption key.

Definition 26. Let $\mathsf{SKE} = \{(\mathcal{K}_\lambda, \mathsf{E}_\lambda, \mathsf{D}_\lambda)\}_{\lambda\in\mathbb{N}}$ be a secret-key encryption scheme with message space $\{0,1\}^\ell$ for $\ell = \ell(\lambda)$. The CIHF associated to this encryption scheme, denoted $\mathcal{H}_{\mathsf{SKE}}$ is

$$\mathcal{H}_{\mathsf{SKE}} = \left\{H_\lambda : \mathcal{I}_\lambda \times \mathcal{K}_\lambda \to \{0,1\}^\ell\right\}_{\lambda\in\mathbb{N}} \text{ where } \; H_\lambda(C, x) := \mathsf{D}_\lambda(x, C)$$

where a key is sampled (from $\mathcal{I}$) as the random ciphertext $C := \mathsf{E}_\lambda(K, M)$ obtained by sampling $K \leftarrow \mathcal{K}_\lambda$ along with $M \leftarrow \{0,1\}^\ell$.

Turning our attention to encryption schemes, we first define what it means for an encryption scheme to have universal ciphertexts.

Definition 27. A secret-key encryption scheme $\mathsf{SKE} = \{(\mathcal{K}_\lambda, \mathsf{E}_\lambda, \mathsf{D}_\lambda)\}_{\lambda\in\mathbb{N}}$ with message space $\mathcal{M}_\lambda$ has *universal ciphertexts* if for any secret key $K \in \mathcal{K}_\lambda$, the distribution $\mathsf{E}_\lambda(K, U_{\mathcal{M}_\lambda})$ multiplicatively $1/\mathbf{poly}(\lambda)$-approximates the distribution $\mathsf{E}_\lambda(\mathcal{K}_\lambda, U_{\mathcal{M}_\lambda})$, where $U_{\mathcal{M}_\lambda}$ denotes the uniform distribution over $\mathcal{M}_\lambda$.

Moving to security, we will define the security of certain primitives with respect to quasi-polynomial adversaries, parameterised by a class of functions $\boldsymbol{\delta}$. For every quasi-polynomial adversary, there exists a function $\delta \in \boldsymbol{\delta}$ such that the success probability of the adversary is bounded by δ. In the context of our work, we will consider $\boldsymbol{\delta}$ to contain functions of the form $\mathbf{quasipoly}(\kappa)/2^\kappa$, where κ will denote the key length. We now define *key-dependent message* (KDM) security for a homomorphic encryption scheme. The security definition for the regular encryption scheme follows in a similar manner, with the evaluation key being empty.

Definition 28 (KDM-security)**.** Let $\mathsf{FHE} = \{(\mathcal{K}_\lambda, \mathsf{E}_\lambda, \mathsf{D}_\lambda, \mathsf{F}_\lambda)\}_{\lambda\in\mathbb{N}}$ be a secret-key fully-homomorphic encryption scheme with message space $\mathcal{M}_\lambda$, let $f = \{f_\lambda : \mathcal{K}_\lambda \to \mathcal{M}_\lambda\}_{\lambda\in\mathbb{N}}$ be a (potentially probabilistic) function. FHE is said to be $\boldsymbol{\delta}$*-immune to key recovery by an* f*-KDM query against quasi-polynomial adversaries* if for each quasi-polynomial-sized $\mathsf{A} = \{\mathsf{A}_\lambda\}_{\lambda\in\mathbb{N}}$, there exists a $\delta \in \boldsymbol{\delta}$ such that:

$$\Pr_{\substack{(K,E)\leftarrow\mathcal{K}_\lambda \\ (M_1,\ldots,M_\ell)\leftarrow f_\lambda(K) \\ \{C_i\leftarrow\mathsf{E}_\lambda(K,M_i)\}_{i\in[1,\ell]}}} [\mathsf{A}_\lambda(E, C_1, \ldots, C_\ell) = K] \leq \delta(\lambda)$$

Throughout this section we will abbreviate the above by saying that FHE is f*-KDM* $\boldsymbol{\delta}$*-secure* against quasi-polynomial adversaries. If $\mathcal{F}$ is a set of functions then we say that FHE is $\mathcal{F}$*-KDM* $\boldsymbol{\delta}$*-secure against quasi-polynomial adversaries* if FHE is f*-KDM* $\boldsymbol{\delta}$*-secure against quasi-polynomial adversaries* for all $f \in \mathcal{F}$.

This is a modification of the f-KDM security used in [27]. Here, an adversary is allowed to run in quasi-polynomial time. Looking ahead, $\mathcal{F}$ will be the collection of all functions, with ℓ bits of output, that can be computed in quasi-polynomial time.

KDM-security and Regev Encryption. Our construction of CIHF and its security hinge on the Regev encryption scheme [92], which we define below.

Definition 29 (Secret-Key Regev Encryption)**.** For any positive integer $q = q(\lambda) \leq 2^{\mathbf{poly}(\lambda)}$, $n' = n'(\lambda) \leq \mathbf{poly}(\lambda)$, and any $\mathbf{poly}(\lambda)$-time sampleable distribution ensembles

$$\chi_{\mathrm{sk}} = \{\chi_{\mathrm{sk}}(\lambda)\}_{\lambda\in\mathbb{N}} \quad \text{and} \quad \chi_{\mathrm{e}} = \{\chi_{\mathrm{e}}(\lambda)\}_{\lambda\in\mathbb{N}} \tag{4.5}$$

over $\mathbb{Z}_q$, we define $\mathsf{Regev}_{n',q,\chi_{\mathrm{sk}},\chi_{\mathrm{e}}}$ to be the secret-key encryption scheme $\{(\mathcal{K}_\lambda, \mathsf{E}_\lambda, \mathsf{D}_\lambda)\}$ where:

- $\mathcal{K}_\lambda$ is the distribution $\chi_{\mathrm{sk}}^{n'}$.
- $\mathsf{E}_\lambda : \mathbb{Z}_q^{n'} \times \{0,1\} \to \mathbb{Z}_q^{n'} \times \mathbb{Z}_q$ is defined so that $\mathsf{E}_\lambda(\boldsymbol{s}, m)$ is obtained by sampling a uniformly random vector $\boldsymbol{a} \leftarrow \mathbb{Z}_q^{n'}$, sampling $e \leftarrow \chi_{\mathrm{e}}(\lambda)$, and outputting
$$\left(\boldsymbol{a}, \boldsymbol{s}^t \cdot \boldsymbol{a} + m \cdot \left\lceil \frac{q}{2} \right\rfloor + e\right).$$
- $\mathsf{D}_\lambda : \mathbb{Z}_q^{n'} \times \left(\mathbb{Z}_q^{n'} \times \mathbb{Z}_q\right) \to \{0,1\}$ is defined so that $\mathsf{D}_\lambda(\boldsymbol{s}, (\boldsymbol{a}, b))$ is the bit m for which $b - \boldsymbol{s}^t \cdot \boldsymbol{a}$ is closer to $m \cdot \left\lceil \frac{q}{2} \right\rfloor$ than to $(1-m) \cdot \left\lceil \frac{q}{2} \right\rfloor$.

A pair $(\boldsymbol{a}, b) \in \mathbb{Z}_q^{n'} \times \mathbb{Z}_q$ is a *Regev encryption of* $m \in \{0,1\}$ *under* $\boldsymbol{s} \in \mathbb{Z}_q^{n'}$ *with* B*-bounded noise* if $b - \boldsymbol{s}^t \cdot \boldsymbol{a} - m \cdot \left\lceil \frac{q}{2} \right\rfloor$ is in the interval $[-B, B)$

We now define a homomorphic encryption scheme that is sufficient for our application. In some sense, these are FHE schemes that have implicit in them a (low-noise) secret-key Regev ciphertext.

Definition 30 (Regev-Extractable Secret-Key Homomorphic Encryption)**.** A secret-key fully-homomorphic bit-encryption scheme $\{(\mathcal{K}_\lambda, \mathsf{E}_\lambda, \mathsf{D}_\lambda, \mathsf{F}_\lambda)\}_{\lambda\in\mathbb{N}}$ is considered $(n', q, \chi_{\mathrm{sk}})$-*Regev-extractable* with $B(\lambda)$-bounded noise if it satisfies the following structural properties.

1. The distribution of $\boldsymbol{s}$ when sampling $(\boldsymbol{s}, E) \leftarrow \mathcal{K}_\lambda$ is $\chi_{\mathrm{sk}}^{n'}$ with χ_{sk} as in eq.(4.5).
2. There is a $\mathbf{poly}(\lambda)$-time evaluable extract function $\mathsf{X} = \{\mathsf{X}_\lambda\}_{\lambda\in\mathbb{N}}$ such that
   (a) For any λ, any $\boldsymbol{s} \in \chi_{\mathrm{sk}}^{n'}$, and any $m \in \{0,1\}$, it holds that $\mathsf{X}_\lambda(\mathsf{E}_\lambda(\boldsymbol{s}, m))$ is a Regev encryption $(\boldsymbol{a}, b)$ of m under $\boldsymbol{s}$ with B-bounded noise, and where $\boldsymbol{a}$ is uniformly random in $\mathbb{Z}_q^{n'}$.
   (b) For any $m_1, \ldots, m_{n'} \in \{0,1\}$, any circuit $\mathsf{C} : \{0,1\}^{n'} \to \{0,1\}$, and any $(\boldsymbol{s}, E) \in \mathcal{K}_\lambda$, it holds with probability 1 that
   $$\mathsf{X}_\lambda(\mathsf{F}_\lambda(E, \mathsf{C}, \mathsf{E}_\lambda(\boldsymbol{s}, m_1), \ldots, \mathsf{E}_\lambda(\boldsymbol{s}, m_{n'})))$$
   is a Regev encryption $(\boldsymbol{a}, b)$ of $\mathsf{C}(m_1, \ldots, m_{n'})$ under $\boldsymbol{s}$ with B-bounded noise.

For our applications, we require the Regev-extractable schemes to have the following security property.

Definition 31 (CPA and circular optimal (CCO) security)**.** Let FHE be an FHE scheme with key distributions $\{\mathcal{K}_\lambda\}_{\lambda\in\mathbb{N}}$. For $(K, E) \in \mathcal{K}_\lambda$, let $[[K]]$ denote the binary representation of K, and let $\kappa = \kappa(\lambda)$ denote the length of such a representation. For any $\ell = \ell(\lambda)$, FHE is said to be *quasi-polynomially* $(\kappa, \ell, \boldsymbol{\delta})$*-CCO-secure* if for every ensemble of ℓ-bit messages $\{m_\lambda\}_{\lambda\in\mathbb{N}}$, FHE is f-KDM $\boldsymbol{\delta}$-secure against quasi-polynomial adversaries for the "augmented bit-by-bit circular security function"

$$f = \left\{ f_\lambda : \mathcal{K}_\lambda \to \{0,1\}^{\ell+\kappa} \right\} \text{ where } f_\lambda(k) = m_\lambda \| [[k]].$$

4.4.2 Fiat-Shamir for the Sumcheck Protocol

Now that we have the relevant definitions, we show these help us achieve the desired result. As indicated while defining the various primitives, we will need to work with adversaries that have quasi-polynomial running time. A central component of this construction is the following theorem from [27, 29] which reduces collision-intractability of the CIHFs to KDM-security. We restate the theorem below, with some important differences for our setting. We have defined our underlying primitives to require both correlation-intractability, and KDM security, against adversaries running in quasi-polynomial time.

Theorem 13 ([29])**.** *Let* $\mathsf{SKE} = \{(\mathcal{K}_\lambda, \mathsf{E}_\lambda, \mathsf{D}_\lambda)\}_{\lambda\in\mathbb{N}}$ *be a secret-key encryption scheme with universal ciphertexts, message space* $\{0,1\}^\ell$*, and key distribution* $\mathcal{K}_\lambda$ *equal to the uniform distribution on* $\{0,1\}^\kappa$ *for some* $\kappa = \kappa(\lambda)$*. If* SKE *is* $\mathcal{F}$*-KDM* $\boldsymbol{\delta}$*-secure against quasi-polynomial adversaries and* $\mathcal{R}$ *is a* ρ*-sparse relation that is* $\lambda^{-O(1)}$*-approximately* $\mathcal{F}$*-sampleable, then for every quasi-polynomial time adversary, there is a* $\delta \in \boldsymbol{\delta}$ *such that* $\mathcal{H}_{\mathsf{SKE}}$ *is* $\mathcal{R}$*-correlation-intractable with*

$$\epsilon := \frac{\delta(\lambda) \cdot \rho(\lambda)}{2^{-\kappa}} \cdot \lambda^{O(1)}.$$

We omit the proof here, as it follows in an identical manner to the proof of the corresponding theorem in [27]. To instantiate the above theorem, we will require CCO-security against *quasi-polynomial* time adversaries for the FHE scheme, which is different from the assumption stated in [27].

Assumption 4 (Existence of quasi-polynomially CCO-secure FHE)**.** For some $n', q, \chi_{\mathrm{sk}}$, there exists a quasi-polynomially $(\kappa, \ell, \boldsymbol{\delta})$-CCO secure secret key FHE scheme that is $(n', q, \chi_{\mathrm{sk}})$-*Regev-extractable* with B-bounded noise for $\kappa = \lambda^{\Theta(1)}, \ell = \lambda^{\Omega(1)}, B \leq q/\tilde{\Omega}(\lambda)$ and $\chi_{\mathrm{sk}}^{n'}$ that is sampleable in $\tilde{O}(n')$ time using $\kappa + O(\log \lambda)$ random bits.

The following claim states that if the assumption is true, then the Regev encryption scheme has universal ciphertexts and satisfies KDM security. As stated earlier, another difference in our assumption is that the class of function $\mathcal{F}^{\ell}$ contains all functions, with output size ℓ, computable in quasi-polynomial time.

Claim 13.1. *If Assumption 4 is true, then there exist parameters $n' = n'(\lambda), q = q(\lambda)$, and $\chi_{sk} = \chi_{sk}(\lambda)$ such that for some $\ell = \lambda^{\Omega(1)}$, $\mathsf{Regev}_{n',q,\chi_{sk},\chi_e}$ is $\mathcal{F}^{\ell}$-KDM $\boldsymbol{\delta}$-secure, where $\mathcal{F}^{\ell}$ is the class of functions with ℓ-bit output computable in* $\mathbf{quasipoly}(\lambda)$ *time, where χ_e is the uniform error distribution on $[-q/4, q/4)$, and where κ is the length of the binary representation of an element of $\chi_{sk}^{n'}$.*

Proof Sketch. Here we briefly sketch the main difference in the proof that necessitates the different assumption from the underlying FHE scheme. For the proof details, we refer the reader to the original proof in [27].

Let $\mathsf{A}_{\mathrm{KDM}}$ be the adversary that breaks the KDM security of the Regev encryption scheme. We will use $\mathsf{A}_{\mathrm{KDM}}$ to build an adversary $\mathsf{A}_{\mathrm{CCO}}$ that breaks Assumption 4. $\mathsf{A}_{\mathrm{CCO}}$ is given as input the challenge $(E, c_1, \ldots, c_{\ell+\kappa})$, where $c_{\ell+1}, \ldots, c_{\ell+\kappa}$ is the encryption of the secret key K.

On initialization, $\mathsf{A}_{\mathrm{KDM}}$ queries the challenger with a function $f^* \in \mathcal{F}^{\ell}$ of its choice, and expects an encryption of $f^*(K)$. In order to facilitate this, $\mathsf{A}_{\mathrm{CCO}}$ uses the homomorphic evaluation function F to homomorphically compute $f^*(K)$ as $\mathsf{F}(E, f^*, c_{\ell+1}, \ldots, c_{\ell+\kappa})$. But given that $\mathcal{F}^{\ell}$ consists of all functions computable in quasi-polynomial time, the above F computation may take quasi-polynomial time. The rest of the reduction remains unchanged.

Therefore, the reduction requires $\mathsf{A}_{\mathrm{CCO}}$ to perform a quasi-polynomial computation, which in turn necessitates that Assumption 4 be secure against quasi-polynomial adversaries. □

We can now state the main theorem: under circular-security assumptions against quasi-polynomial time adversaries for the FHE scheme, the Fiat-Shamir Transform when applied to the Sumcheck Protocol is an adaptively-sound argument.

Theorem 14. *If Assumption 4 is true, then the non-interactive Sumcheck Protocol $\overline{\mathsf{GSC}}$ from Algorithm 4.3, instantiated with the hash family $\mathcal{H}_{\mathsf{SKE}}$, is adaptively unambiguously sound for the language L_{SC} (see §4.1.2), with formulas on $n = \mathbf{polylog}(\lambda)$ variables.*

Remark 14. We note that the size of the field $\mathbb{F}$, defined to be $2^{\omega(n)}$, is larger than any quasi-polynomial in the parameter λ. This follows from the fact that for correlation-intractability to make sense against quasi-polynomial adversaries, we require the relation to be sufficiently sparse. Setting the field-size to be quasi-polynomial is not sufficient since a quasi-polynomial time adversary can break correlation-intractability even when the hash function is modeled as a random oracle by trying quasi-polynomially many different values of x.

Proof. We proceed in two stages, initially establishing that a CIHF for the relevant relation implies that the Fiat-Shamir Transform is adaptively-sound. Next, we show that under suitable choices of parameters for the underlying primitives, we can instantiate such a CIHF.

Stage I: From correlation-intractability to soundness. The use of CIHFs to instantiate the Fiat-Shamir Transform was suggested in [30]. The first stage of our reduction can be seen as a straightforward adaptation of their argument (as provided in [27]) to our setting. For the first part, we restate the following claim from [27], removing the efficiency requirements needed for their result, and adapting it to the specific case of the Sumcheck Protocol.

The following relation specifies whether partial transcript in Sumcheck Protocol is *bad* or *good*. For any $i \in [j+1, n]$, an i-th *round partial transcript* consists of the statement $(\mathbb{F}, y, f, \boldsymbol{\beta})$, and $\boldsymbol{\tau}_i := \boldsymbol{\alpha}_{j+1}, \beta_{j+1}, \ldots, \boldsymbol{\alpha}_i, \beta_i$. Recall that $\boldsymbol{\alpha}_i = \{\alpha_{i,\gamma}\}_{\gamma=0}^{d}$ defines a unique degree d polynomial g_i. Formally the relation $\mathcal{R}_{SC}$ is defined as,

$$\mathcal{R}_{SC} := \{((\mathbb{F}, y, f, \boldsymbol{\beta}, \boldsymbol{\tau}_i, \boldsymbol{\alpha}), \beta) : (\mathbb{F}, y, f, \boldsymbol{\beta}, \boldsymbol{\tau}_i) \in \mathcal{B} \wedge (\mathbb{F}, y, f, \boldsymbol{\beta}, \boldsymbol{\tau}_i, \boldsymbol{\alpha}, \beta) \in \mathcal{G}\}$$

for some i-th *round partial transcript*. We say that a partial transcript $(\mathbb{F}, y, f, \boldsymbol{\beta}, \boldsymbol{\tau}_i) \in \mathcal{B}$ (bad) if

$$(\mathbb{F}, g_i(\beta_i), f, \boldsymbol{\beta}, \beta_{j+1}, \ldots, \beta_i) \notin \mathrm{L}_{SC}$$

where $\beta_{j+1}, \ldots, \beta_i$ and g_i are obtained from $\boldsymbol{\tau}_i$. Roughly, a partial transcript is *bad* if it leads the verifier in the interactive protocol to reject with high probability. Correspondingly, we say that a partial transcript $(\mathbb{F}, y, f, \boldsymbol{\beta}, \boldsymbol{\tau}_i) \in \mathcal{G}$ (good) if

$$(\mathbb{F}, g_i(\beta_i), f, \boldsymbol{\beta}, \beta_{j+1}, \ldots, \beta_i) \in \mathrm{L}_{SC},$$

again roughly translating to a partial transcript that leads the verifier to in the interactive protocol to accept. We now state the claim.

Claim 14.1. *Let $\Pi = (\mathsf{P}_{SC}, \mathsf{V}_{SC})$ be the $O(\mathbf{polylog}(\lambda))$-round public-coin interactive protocol for the language L_{SC} with perfect completeness and adaptive soundness. If a hash family $\mathcal{H}$ is $\mathcal{R}_{SC}$ correlation-intractable, and evaluable in time $\mathbf{poly}(\lambda)$, then Fiat-Shamir Transform gives an adaptively-sound argument for L_{SC}.*

Proof. This follows in a straightforward manner as in the proof in [27], and is included here for completeness. The completeness of the protocol follows from the completeness of the underlying protocol (P, V). For the adaptive soundness, we prove this via contradiction. Suppose there exists a cheating prover P^* that on input $(1^\lambda, H)$, where H is sampled from $\mathcal{H}_\lambda$, that produces a string $(\mathbb{F}, y^*, f^*, \beta_1^*, \ldots, \beta_j^*) \notin \mathrm{L}_{SC}$. i.e

$$\sum_{\boldsymbol{z} \in \{0,1\}^{n-j}} f^*(\beta_1^*, \ldots, \beta_j^*, \boldsymbol{z}) \neq y^*$$

and $(\boldsymbol{\alpha}_{j+1}^*, \ldots, \boldsymbol{\alpha}_n^*)$ such that V accepts the transcript derived using H. We shall use this cheating prover P^* to create an adversary $\mathsf{A} = \{\mathsf{A}_\lambda\}_{\lambda \in \mathbb{N}}$ that breaks the $\mathcal{R}_{SC}$-correlation-intractability of $\mathcal{H}$. On receiving $H \in \mathcal{H}_\lambda$, A_λ does the following:

1. Run P^* on input $(1^\lambda, H)$ to obtain $(\mathbb{F}, y^*, f^*, \beta_1^*, \ldots, \beta_j^*)$ and $(\boldsymbol{\alpha}_{j+1}^*, \ldots, \boldsymbol{\alpha}_n^*)$.

2. Sample a random index $i^* \leftarrow [j+1, n-1]$.
3. Return $\left(\mathbb{F}, y^*, f^*, \beta_1^*, \ldots, \beta_j^*, \boldsymbol{\tau}_{i^*}, \boldsymbol{\alpha}_{i^*+1}^*\right)$, where $\forall k \in [1, i]$:

$$\beta_k = H(\mathbb{F}, y^*, f^*, \beta_1^*, \ldots, \beta_j^*, \boldsymbol{\tau}_{k-1}, \boldsymbol{\alpha}_k^*).$$

From Sumcheck Protocol, for every accepting transcript for $(\mathbb{F}, y^*, f^*, \beta_1^*, \ldots, \beta_j^*) \notin \mathrm{L}_{SC}$, there must exist at least one round k such that:

$$(\mathbb{F}, y^*, f^*, \beta_1^*, \ldots, \beta_j^*, \boldsymbol{\tau}_k) \in \mathcal{B} \text{ and } \left(\mathbb{F}, y^*, f^*, \beta_1^*, \ldots, \beta_j^*, \boldsymbol{\tau}_k, \boldsymbol{\alpha}_{k+1}^*, \beta_{k+1}\right) \in \mathcal{G}.$$

Note that this follows from the fact that $(\mathbb{F}, f^*, y^*, \beta_1^*, \ldots, \beta_j^*) \notin \mathrm{L}_{SC}$, and for V to accept, the complete transcript must be $\mathcal{G}$. Thus with probability $\epsilon/(n-j-1)$, A_λ selects the appropriate index k, and outputs the correct partial transcript. This contradicts our assumption of correlation-intractability. □

Stage II: Building the appropriate CIHF. It remains to show that we can build a CIHF for the relation $\mathcal{R}_{SC}$. Now, we need to establish certain properties from the relation $\mathcal{R}_{SC}$ in order to invoke Theorem 13. We start with two simple claims regarding the relation $\mathcal{R}_{SC}$. For simplicity of exposition, let us denote by g_{i+1} the prescribed polynomial (given prefix $\boldsymbol{\beta}$) to be sent in round $i+1$, i.e.

$$g_{i+1}(x) := \sum_{\boldsymbol{z} \in \{0,1\}^{n-i-1}} f(\boldsymbol{\beta}, \beta_{j+1}, \ldots, \beta_i, x, \boldsymbol{z}).$$

Claim 14.2. *$\mathcal{R}_{SC}$ is a ρ-sparse relation for $\rho = d/|\mathbb{F}|$.*

Proof. Given $(\mathbb{F}, y, f, \boldsymbol{\beta}, \boldsymbol{\tau}_i, \boldsymbol{\alpha})$, we compute the fraction of β such that

$$((\mathbb{F}, y, f, \boldsymbol{\beta}, \boldsymbol{\tau}_i, \boldsymbol{\alpha}), \beta) \in \mathcal{R}_{SC}.$$

For $(\mathbb{F}, y, f, \boldsymbol{\beta}, \boldsymbol{\tau}_i, \boldsymbol{\alpha}, \beta) \in \mathcal{G}$, we require β to be such that $\widetilde{g}(\beta) = g_{i+1}(\beta)$, where $\widetilde{g}(x)$ denotes the polynomial described by $\boldsymbol{\alpha}$, and $g_{i+1}(x)$ is as defined above. This follows from the definition of L_{SC}. The polynomial $g_{i+1}(x) - \widetilde{g}(x)$ has degree at most d (since $g(x)$ has degree at most d), and is non-zero (since $\widetilde{g}$ is not the prescribed polynomial). Thus, from Schwartz-Zippel lemma, there are at most d roots to the above polynomial, and thus d values β such that $\widetilde{g}(\beta) = g_{i+1}(\beta)$. Thus the fractions of such values are $d/|\mathbb{F}|$. Since, we have set $|\mathbb{F}|$ to be of size $\omega(\mathbf{quasipoly}(\lambda))$, ρ is negligible. □

Claim 14.3. *$\mathcal{R}_{SC}$ is sampleable in $\mathbf{quasipoly}(\lambda)$-time.*

Proof. The algorithm for sampling a β given $((\mathbb{F}, y, f, \boldsymbol{\beta}, \boldsymbol{\tau}_i, \boldsymbol{\alpha})$ works in the following manner. Using the Cantor-Zassenhaus algorithm [32], we can enumerate all roots with probability 2/3, and therefore with any probability arbitrarily exponentially close to 1. If the factorization succeeds, we can sample an element from this set of all roots with arbitrarily small sampling error. As described above, it is sufficient to output a random root of $g_{i+1}(x) - \widetilde{g}(x)$. The running time of the above sampling strategy derives from the fact that to compute the polynomial g_{i+1}, we need to compute an exponential sum over $\mathbf{polylog}(\lambda)$ variables. We note that from Remark 14, the size of field is larger than any quasi-polynomial, and thus we do not know if we can do this determinisitcally in quasi-polynomial time.

□

Theorem 13 requires $\mathcal{R}_{SC}$ to be sampled by a function in $\mathcal{F}$. Thus, in our setting, $\mathcal{F}$ represents the set of all functions computable in $\mathbf{quasipoly}(\lambda)$ time. We additionally make the following simple observations regarding the Sumcheck Protocol:

- The total number of rounds ρ in the protocol is $\mathbf{polylog}(\lambda)$. This follows from the structure of the protocol wherein each round corresponds to reducing the claim by a single variable, and we have set the number of variables to be $\mathbf{polylog}(\lambda)$
- The length of each verifier message is $|\beta_i| = \omega(\mathbf{polylog}(\lambda))$ by our choice of parameters. This follows from the fact that each $\beta_i \in \mathbb{F}$.
- The size of the input to the hash function, $(\mathbb{F}, y, f, \boldsymbol{\beta}, \boldsymbol{\tau}_\rho)$, is $\mathbf{poly}(\lambda)$. This follows from the fact that in addition to the description of the function f (of size $\mathbf{poly}(\lambda)$), the input consists of ρ rounds of prover, and verifier, messages. A prover message consists of only $O(d)$ elements from $\mathbb{F}$. Given that the number of rounds are $\mathbf{polylog}(\lambda)$, this gives an additive overhead of $\omega(\mathbf{polylog}(\lambda))$ to the description of f.

Finally,to instantiate Theorem 13, we need an appropriate encryption scheme with universal ciphertexts, and KDM security for all quasi-polynomial computable functions. Specifically, we require an encryption scheme $\mathsf{SKE} = (\mathsf{SKE.G}, \mathsf{SKE.E}, \mathsf{SKE.D})$ with keys of length $\kappa = \kappa(\lambda) \geq \lambda^{\Omega(1)}$ and universal ciphertexts that are $\boldsymbol{\delta}$-KDM secure for arbitrary quasi-polynomial computable functions, of output length ℓ.

Assumption 4 implies that secret key Regev encryption satisfies these properties, with secret distribution χ_{sk} that is uniform on $[-B, B)$ for some B, and error distribution χ_{err} that is uniform on $\left[-\frac{q}{4}, \frac{q}{4}\right)$. For the corresponding scheme n' is set to be such that $(2B+1)^{n'} \in \{0,1\}^{|\boldsymbol{\tau}|}$, where $|\boldsymbol{\tau}|$ is the size of the largest input to the hash function, and $\ell = \omega(\mathbf{polylog}(\lambda))$ is the size of a single verifier message.

We now have all the requisite conditions for Theorem 13, and thus invoking the result, the Fiat-Shamir Transform gives us an adaptively-sound argument system. From Proposition 2, an adaptively-sound argument system is also an adaptively unambigiously-sound argument system, thus completing the proof. □

Chapter 5
Future Directions

We have seen how the average-case hardness of lower classes of **TFNP** can be established using assumptions that are of a different flavour from indistinguishability obfuscation. Moreover, it seems plausible that hardness in **PPAD** and **CLS** could be established from standard cryptographic assumptions like FACTORING or well-studied primitives such as fully-homomorphic encryption: we saw some evidence for this in the previous chapter. Our results and techniques motivate several natural research directions.

Hardness relative to random oracles. In [34], the paper that corresponds to Chapter 4, we had claimed that the second construction yields hardness in $\mathbf{CLS} \subseteq \mathbf{PPAD}$ relative to a random oracle: i.e., $\mathbf{P}^H \neq \mathbf{CLS}^H$ for a random oracle H. However, as we saw in Remark 12 this claim is wrong. Using further ideas from incrementally-verifiable computing, a recent result [12] has shown that such a statement is indeed true for the class **PLS**: $\mathbf{P}^H \neq \mathbf{PLS}^H$ for a random oracle H. However, for classes **PPAD** and **CLS** this still remains an interesting open problem.

It is worth pointing out that constructing RSVL instances is closely tied to the question of constructing verifiable delay functions (VDFs) [16, 17]. Loosely speaking, a VDF is a deterministic function $f : \mathcal{X} \to \mathcal{Y}$ that takes long *sequential time* T to compute, but can be verified efficiently in time $t \ll T$ (say $t = \log(T)$). Note that this makes a VDF a weaker object compared to RSVL.[1] It was recently shown that there are barriers to constructing VDFs having certain properties in the random oracle model [76, 41]. To be more precise, it was shown that perfectly-unique VDFs (i.e., every pair $(x, f(x))$ has a single valid proof) and tight VDFs (i.e., proofs can also be generated in time almost T) cannot be constructed in the random-oracle model. Therefore, these impossibility results carry over to constructing RSVL in the random oracle model, and any approach at showing **PPAD**-hardness via RSVL should bypass these impossibility results. Note that this is true for the RSVL constructions in the thesis as they are not tight (there is a blow-up by log(3) factor in the exponent) and the proofs are only computationally-unique.

Replacing random oracles. A natural question that is worth following up is whether the Fiat-Shamir Transform for the second construction can be instantiated under assumptions weaker than Assumption 4. This will require a finer understanding of the notion of collision-intractability for Sumcheck Protocol.

Note that this question is equally interesting (and arguably easier to address) in the setting of our first construction, where the random oracle is used 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

[1] However, if one considers a stricter "continuous" form of VDF, where the proof needs to be available also for intermediate states of computation, then it corresponds exactly to RSVL [42].

be possible to instantiate Fiat-Shamir using standard assumption or alternatively design a non-interactive proof system for certifying $y = f(N, x, T)$ directly, i.e., without relying on Fiat-Shamir.

One such approach would be to adapt the recent results from [67] to our settings, in particular to the second construction. In [67] a delegation scheme is constructed for arbitrary computations under a new, but falsifiable, assumption on bilinear maps. They start off with a long CRS (common reference string) and then use bootstrapping techniques from [103, 11] to obtain delegations schemes with short CRS. Since we don't have to deal with general computation, it might turn out that their construction can be adapted to suit our needs under weaker assumptions.

Reducing from Factoring. FACTORING is often regarded as one of the "rogue" problems in **TFNP** [52] as it has withstood attempts to be placed in one of the subclasses (with respect to Karp reductions). In the thesis we worked primarily with ITERATED-SQUARING which, as we saw, is Karp-reducible to FACTORING. Thus, hardness of FACTORING is necessary for our hardness assumption on ITERATED-SQUARING (Assumption 2) to hold. However, similar to the RSW assumption or the RSA assumption, it is not clear if hardness of FACTORING is sufficient. Therefore, orthogonal to the question raised above (about removing random oracles), it is natural to ask if our techniques can be improved to reduce from FACTORING in the random-oracle model. For instance, is it possible to implement one of the many factorisation algorithms in an incrementally verifiable manner? We describe below some of our attempts and explain why they failed.

1. *Exploiting the* SVL *oracle.* Our reduction basically shows that the access to an RSVL oracle enables one to compute exponential powers modulo a composite efficiently (and incrementally-verifiably). One could ask whether this oracle can be exploited to factor integers. Unfortunately, this problem seems closely related to a long-unresolved problem from [14] in the context of pseudo-random generators (PRG). The PRG in [14] (known as the Blum-Blum-Shub) works in the same modulus N as in ITERATED-SQUARING. Starting from a random seed $x \in \mathbb{Z}_N^*$, it generates random stream of bits by squaring the current state and extracting its least significant bit (LSB): i.e.

$$LSB(x^2) \to LSB(x^{2^2}) \to LSB(x^{2^3}) \cdots \to LSB(x^{2^T}) \ (\mathrm{mod} N). \tag{5.1}$$

   One of the open questions posed there (in §9) is whether random access to this stream allows factoring N.

2. *Computing square-root.* The structure of the squaring function on the modulus in ITERATED-SQUARING (IS) is fairly well-understood [14]. It is known that for elements $x \in QR_N$ it has a cyclic structure

$$\left\{x, x^2, x^{2^2}, x^{2^3}, \ldots, x^{2^{\pi(x)}} = x \bmod N\right\}, \tag{5.2}$$

   with the length of the cycle $\pi(x)$ related closely to the factors of N. Moreover, the penultimate entry in this sequence $x^{2^{\pi(x)-1}}$ is the square-root of x modulo N, and computing it is equivalent to factoring N [91]. Therefore, in order to reduce from FACTORING, one could define an SVL instance which computes the sequence in eq.(5.2) incrementally-verifiably as we did for IS.

However, there is one issue: it is not clear how to set the length parameter L beforehand (since it is hard to compute without the factors). To this end, let's define a slight variant of the SVL problem, *without* the length parameter, and with the sink vertex that is obtained by simply following the successor function till it self-loops set as the solution. The successor function in the instance can now be defined just like in our first construction but with one difference: since there is no length parameter, it continues to incrementally and verifiably compute IS *until* the penultimate entry in the sequence i.e., $x^{2^{\pi(x)-1}}$, (which can be tested efficiently by squaring the current value) at which point it self-loops. In order to now mimic the reduction from SVL to EOL (Lemma 2) for the above variant of SVL, we need to maintain a counter that tracks the distance from the source in the label. The successor would have to increment this counter at every step.

Although the approach seems to work at first, a closer inspection reveals an inherent problem. This problem arises from the introduction of the counter in the label, and has to do with the fact that

$$x^{2^i} = x^{2^{i+\pi(x)}} \bmod N.$$

A successor function that is input the label with a counter $i + \pi(x)$ has no way of knowing that this point is *beyond* the sequence and therefore simply applies the normal succession rule. As a result, it will soon run out of space for labels and therefore will be forced self-loop at the lexicographically-last label, making it an easy-to-find sink (called "Pavel" sinks in some circles).

We remark that we run into similar issues if we try to incrementally-verifiably implement the factoring algorithms that rely on cycle-finding (e.g., Pollard's [90] or Brent's [23] algorithm). The bottom-line is that we will have to tackle cycles of *unknown* length if we are to make any progress towards reducing from Factoring.

Reducing from Factoring-like assumptions. It should also be of interest if assumptions related to Factoring, such as RSA and composite residuosity (CR) assumption [83], can be reduced to EOL (even in the random-oracle model). These problems both reduce to Factoring (see Figure 5.1) and it is widely believed that RSA is not equivalent to Factoring [18]. Moreover, they have a richer structure that could be exploited.

Towards this goal, it would be interesting to look at the family of classes **PPA**-k (for $k \in \mathbb{N}$) defined in [84], where **PPA**-2 corresponds to **PPA** from §1.1.1.1. The class has recently generated a lot of interest as it was shown that **PPAD** $\subseteq$ **PPA**-k for every $k \geq 2$ [60, 57]. Therefore, a good starting point to showing **PPAD**-hardness from Factoring and its related assumptions would be to study their relationship with **PPA**-k (for $k > 2$), which still remains open.

Incrementally-verifiable computation of Lucas sequences. The crucial observation we make in this thesis is that the possibility to merge proofs somewhat-efficiently is sufficient for performing certain computations in an incrementally-verifiable manner. We expect this technique to find applications in different contexts.

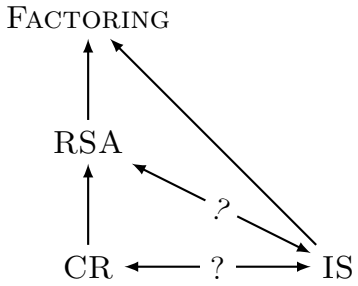


Figure 5.1: Factoring and its related assumptions. $A \to B$ denotes that problem A reduces to problem B. The arrows with a question mark denote unknown relationships.

One such possibility could be to the computation of Lucas sequences [71], which are certain constant-recursive recurrence relations of the form

$$x_n = P \cdot x_{n-1} + Q \cdot x_{n-2},$$

where $P, Q \in \mathbb{N}$ are two parameters. An alternative method to compute n-th term in this sequence is using iterated squaring in an extension field using the closed form $x_n = \alpha^{2^n} + \beta^{2^n}$, where α, β are elements of the extension field that can be derived from P and Q. Thus its computation is similar to that of ITERATED-SQUARING, and similar techniques potentially apply.

One of the applications of Lucas sequences is in Lucas-Lehmer test, which is used to test the primarily of Mersenne primes, i.e. are primes of the form $M_p = 2^p - 1$. Given a prime p, to test whether M_p is also a prime we first compute the sequence $s_0 = 4, s_1, \ldots, s_{p-2} \bmod M_p$, where $s_{i+1} = s_i^2 \bmod M_p$. By a theorem of Lucas and Lehmer it follows that M_p is a prime iff $s_{p-2} = 0 \bmod M_p$. We would like to point out that the largest known prime known was discovered using this method and has p of size around 80 million digits. Therefore, these computations are extremely time-consuming (taking order of months) and the only way that another party can verify is by repeating the computation. By using incrementally-verifiable procedure, one could generate a certificate of primarily along with the computation, which would enable other parties to validate such primes more efficiently.

Bibliography

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

[2] Ajtai, M. Generating hard instances of lattice problems (extended abstract). In *28th Annual ACM Symposium on Theory of Computing* (Philadephia, PA, USA, May 22–24, 1996), ACM Press, pp. 99–108.

[3] Babichenko, Y. Query complexity of approximate Nash equilibria. *J. ACM 63*, 4 (2016), 36:1–36:24.

[4] Ban, F., Jain, K., Papadimitriou, C. H., Psomas, C.-A., and Rubinstein, A. Reductions in **PPP**. *Information Processing Letters 145* (2019), 48 – 52.

[5] 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.

[6] Beame, P., Cook, S., Edmonds, J., Impagliazzo, R., and Pitassi, T. The relative complexity of **NP** search problems. *Journal of Computer and System Sciences 57*, 1 (1998), 3 – 19.

[7] Bellare, M., and Rogaway, P. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM CCS 93: 1st Conference on Computer and Communications Security* (Fairfax, Virginia, USA, Nov. 3–5, 1993), D. E. Denning, R. Pyle, R. Ganesan, R. S. Sandhu, and V. Ashby, Eds., ACM Press, pp. 62–73.

[8] Bennett, C. H. Time/space trade-offs for reversible computation. *SIAM J. Comput. 18*, 4 (1989), 766–776.

[9] Bennett, C. H., and Gill, J. Relative to a random oracle A, $\mathbf{P}^A \neq \mathbf{NP}^A \neq \mathbf{co\text{-}NP}^A$ with probability 1. *SIAM Journal on Computing 10*, 1 (1981), 96–113.

[10] Bitansky, N., Canetti, R., Chiesa, A., and Tromer, E. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *ITCS 2012: 3rd Innovations in Theoretical Computer Science* (Cambridge, MA, USA, Jan. 8–10, 2012), S. Goldwasser, Ed., Association for Computing Machinery, pp. 326–349.

[11] Bitansky, N., Canetti, R., Chiesa, A., and Tromer, E. Recursive composition and bootstrapping for SNARKS and proof-carrying data. In *45th Annual ACM Symposium on Theory of Computing* (Palo Alto, CA, USA, June 1–4, 2013), D. Boneh, T. Roughgarden, and J. Feigenbaum, Eds., ACM Press, pp. 111–120.

[12] Bitansky, N., and Gerichter, I. On the cryptographic hardness of local search. In *ITCS 2020: 11th Innovations in Theoretical Computer Science Conference* (Seattle, WA, USA, Jan. 12–14, 2020), T. Vidick, Ed., vol. 151, LIPIcs, pp. 6:1–6:29.

[13] 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.

[14] Blum, L., Blum, M., and Shub, M. A simple unpredictable pseudo-random number generator. *SIAM Journal on Computing 15*, 2 (1986), 364–383.

[15] 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.

[16] Boneh, D., Bonneau, J., Bünz, B., and Fisch, B. Verifiable delay functions. In *Advances in Cryptology – CRYPTO 2018, Part I* (Santa Barbara, CA, USA, Aug. 19–23, 2018), H. Shacham and A. Boldyreva, Eds., vol. 10991 of *Lecture Notes in Computer Science*, Springer, Heidelberg, Germany, pp. 757–788.

[17] 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`.

[18] Boneh, D., and Venkatesan, R. Breaking RSA may not be equivalent to factoring. In *Advances in Cryptology – EUROCRYPT'98* (Espoo, Finland, May 31 – June 4, 1998), K. Nyberg, Ed., vol. 1403 of *Lecture Notes in Computer Science*, Springer, Heidelberg, Germany, pp. 59–71.

[19] Brakerski, Z. Fully homomorphic encryption without modulus switching from classical GapSVP. In *Advances in Cryptology – CRYPTO 2012* (Santa Barbara, CA, USA, Aug. 19–23, 2012), R. Safavi-Naini and R. Canetti, Eds., vol. 7417 of *Lecture Notes in Computer Science*, Springer, Heidelberg, Germany, pp. 868–886.

[20] Brakerski, Z., Gentry, C., and Vaikuntanathan, V. (Leveled) fully homomorphic encryption without bootstrapping. In *ITCS 2012: 3rd Innovations in Theoretical Computer Science* (Cambridge, MA, USA, Jan. 8–10, 2012), S. Goldwasser, Ed., Association for Computing Machinery, pp. 309–325.

[21] Brakerski, Z., and Vaikuntanathan, V. Efficient fully homomorphic encryption from (standard) LWE. In *52nd Annual Symposium on Foundations of Computer Science* (Palm Springs, CA, USA, Oct. 22–25, 2011), R. Ostrovsky, Ed., IEEE Computer Society Press, pp. 97–106.

[22] Brakerski, Z., and Vaikuntanathan, V. Lattice-based FHE as secure as PKE. In *ITCS 2014: 5th Conference on Innovations in Theoretical Computer Science* (Princeton, NJ, USA, Jan. 12–14, 2014), M. Naor, Ed., Association for Computing Machinery, pp. 1–12.

[23] Brent, R. P. An improved monte carlo factorization algorithm. *BIT Numerical Mathematics 20*, 2 (1980), 176–184.

[24] 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.

[25] Buresh-Oppenheim, J. On the **TFNP** complexity of factoring. Unpublished, `http://www.cs.toronto.edu/~bureshop/factor.pdf`, 2006.

[26] Cai, J. ., Lipton, R. J., Sedgewick, R., and Yao, A. C. . Towards uncheatable benchmarks. In *[1993] Proceedings of the Eigth Annual Structure in Complexity Theory Conference* (May 1993), pp. 2–11.

[27] Canetti, R., Chen, Y., Holmgren, J., Lombardi, A., Rothblum, G. N., Rothblum, R. D., and Wichs, D. Fiat-Shamir: from practice to theory. In *51st Annual ACM Symposium on Theory of Computing* (Phoenix, AZ, USA, June 23–26, 2019), M. Charikar and E. Cohen, Eds., ACM Press, pp. 1082–1090.

[28] Canetti, R., Chen, Y., and Reyzin, L. On the correlation intractability of obfuscated pseudorandom functions. In *TCC 2016-A: 13th Theory of Cryptography Conference, Part I* (Tel Aviv, Israel, Jan. 10–13, 2016), E. Kushilevitz and T. Malkin, Eds., vol. 9562 of *Lecture Notes in Computer Science*, Springer, Heidelberg, Germany, pp. 389–415.

[29] 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.

[30] Canetti, R., Goldreich, O., and Halevi, S. The random oracle methodology, revisited. *J. ACM 51*, 4 (July 2004), 557–594.

[31] Canetti, R., Lombardi, A., and Wichs, D. Fiat-Shamir: From practice to theory, part II (NIZK and correlation intractability from circular-secure FHE). Cryptology ePrint Archive, Report 2018/1248, 2018. `https://eprint.iacr.org/2018/1248`.

[32] Cantor, D. G., and Zassenhaus, H. A new algorithm for factoring polynomials over finite fields. *Mathematics of Computation 36* (1981), 587–592.

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

[34] Choudhuri, A. R., Hubácek, P., Kamath, C., Pietrzak, K., Rosen, A., and Rothblum, G. N. Finding a nash equilibrium is no easier than breaking Fiat-Shamir. In *51st Annual ACM Symposium on Theory of Computing* (Phoenix, AZ, USA, June 23–26, 2019), M. Charikar and E. Cohen, Eds., ACM Press, pp. 1103–1114.

[35] 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`.

[36] Choudhuri, A. R., Hubacek, P., Kamath, C., Pietrzak, K., Rosen, A., and Rothblum, G. N. PPAD-hardness via iterated squaring modulo a composite. Cryptology ePrint Archive, Report 2019/667, 2019. https://eprint.iacr.org/2019/667.

[37] Chung, F., Diaconis, P., and Graham, R. Combinatorics for the east model. *Advances in Applied Mathematics 27*, 1 (2001), 192–206.

[38] Cohen, H. *A course in computational algebraic number theory*, vol. 138 of *Graduate texts in mathematics*. Springer, 1993.

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

[40] 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.

[41] Döttling, N., Garg, S., Malavolta, G., and Vasudevan, P. N. Tight verifiable delay functions. Cryptology ePrint Archive, Report 2019/659, 2019. https://eprint.iacr.org/2019/659.

[42] 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.

[43] Fearnley, J., Gordon, S., Mehta, R., and Savani, R. Unique end of potential line. In *ICALP 2019: 46th International Colloquium on Automata, Languages and Programming* (Patras, Greece, July 9–12, 2019), C. Baier, I. Chatzigiannakis, P. Flocchini, and S. Leonardi, Eds., vol. 132 of *LIPIcs*, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, pp. 56:1–56:15.

[44] 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.

[45] Filos-Ratsikas, A., and Goldberg, P. W. The complexity of splitting necklaces and bisecting ham sandwiches. In *51st Annual ACM Symposium on Theory of Computing* (Phoenix, AZ, USA, June 23–26, 2019), M. Charikar and E. Cohen, Eds., ACM Press, pp. 638–649.

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

[47] Garg, S., Gentry, C., Halevi, S., Raykova, M., Sahai, A., and Waters, B. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th Annual Symposium on Foundations of Computer Science* (Berkeley, CA, USA, Oct. 26–29, 2013), IEEE Computer Society Press, pp. 40–49.

[48] Garg, S., Pandey, O., and Srinivasan, A. Revisiting the cryptographic hardness of finding a nash equilibrium. In *Advances in Cryptology – CRYPTO 2016,*

Part II (Santa Barbara, CA, USA, Aug. 14–18, 2016), M. Robshaw and J. Katz, Eds., vol. 9815 of *Lecture Notes in Computer Science*, Springer, Heidelberg, Germany, pp. 579–604.

[49] Gentry, C., Sahai, A., and Waters, B. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *Advances in Cryptology – CRYPTO 2013, Part I* (Santa Barbara, CA, USA, Aug. 18–22, 2013), R. Canetti and J. A. Garay, Eds., vol. 8042 of *Lecture Notes in Computer Science*, Springer, Heidelberg, Germany, pp. 75–92.

[50] Gentry, C., and Wichs, D. Separating succinct non-interactive arguments from all falsifiable assumptions. In *43rd Annual ACM Symposium on Theory of Computing* (San Jose, CA, USA, June 6–8, 2011), L. Fortnow and S. P. Vadhan, Eds., ACM Press, pp. 99–108.

[51] Goldberg, P. W., and Hollender, A. The hairy ball problem is PPAD-complete. In *ICALP 2019: 46th International Colloquium on Automata, Languages and Programming* (Patras, Greece, July 9–12, 2019), C. Baier, I. Chatzigiannakis, P. Flocchini, and S. Leonardi, Eds., vol. 132 of *LIPIcs*, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, pp. 65:1–65:14.

[52] Goldberg, P. W., and Papadimitriou, C. H. Towards a unified complexity theory of total functions. In *ITCS 2018: 9th Innovations in Theoretical Computer Science Conference* (Cambridge, MA, USA, Jan. 11–14, 2018), A. R. Karlin, Ed., vol. 94, LIPIcs, pp. 37:1–37:20.

[53] Goldreich, O. Candidate one-way functions based on expander graphs. In *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation - In Collaboration with Lidor Avigad, Mihir Bellare, Zvika Brakerski, Shafi Goldwasser, Shai Halevi, Tali Kaufman, Leonid Levin, Noam Nisan, Dana Ron, Madhu Sudan, Luca Trevisan, Salil Vadhan, Avi Wigderson, David Zuckerman.* 2011, pp. 76–87.

[54] Goldreich, O., and Rothblum, G. N. Worst-case to average-case reductions for subclasses of P. *Electronic Colloquium on Computational Complexity (ECCC) 24* (2017), 130.

[55] Goldwasser, S., and Kalai, Y. T. On the (in)security of the Fiat-Shamir paradigm. In *44th Annual Symposium on Foundations of Computer Science* (Cambridge, MA, USA, Oct. 11–14, 2003), IEEE Computer Society Press, pp. 102–115.

[56] Goldwasser, S., Micali, S., and Rackoff, C. The knowledge complexity of interactive proof systems. *SIAM J. Comput. 18*, 1 (1989), 186–208.

[57] Göös, M., Kamath, P., Sotiraki, K., and Zampetakis, M. On the complexity of modulo-q arguments and the chevalley-warning theorem. *CoRR abs/1912.04467* (2019).

[58] Hirsch, M. D., Papadimitriou, C. H., and Vavasis, S. A. Exponential lower bounds for finding Brouwer fix points. *J. Complexity 5*, 4 (1989), 379–416.

[59] 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.

[60] Hollender, A. The classes ppa-k: Existence from arguments modulo k. In *WINE* (2019), vol. 11920 of *Lecture Notes in Computer Science*, Springer, pp. 214–227.

[61] Hubácek, P., Naor, M., and Yogev, E. The journey from NP to TFNP hardness. In *ITCS 2017: 8th Innovations in Theoretical Computer Science Conference* (Berkeley, CA, USA, Jan. 9–11, 2017), C. H. Papadimitriou, Ed., vol. 4266, LIPIcs, pp. 60:1–60:21.

[62] Hubácek, P., and Yogev, E. Hardness of continuous local search: Query complexity and cryptographic lower bounds. In *28th Annual ACM-SIAM Symposium on Discrete Algorithms* (Barcelona, Spain, Jan. 16–19, 2017), P. N. Klein, Ed., ACM-SIAM, pp. 1352–1371.

[63] Impagliazzo, R. A personal view of average-case complexity. In *Computational Complexity Conference* (1995), IEEE Computer Society, pp. 134–147.

[64] Jeřábek, E. Integer factoring and modular square roots. *J. Comput. Syst. Sci. 82*, 2 (2016), 380–394.

[65] 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.

[66] 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.

[67] Kalai, Y. T., Paneth, O., and Yang, L. How to delegate computations publicly. In *51st Annual ACM Symposium on Theory of Computing* (Phoenix, AZ, USA, June 23–26, 2019), M. Charikar and E. Cohen, Eds., ACM Press, pp. 1115–1124.

[68] 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.

[69] 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* (Berkeley, CA, USA, Oct. 15–17, 2017), C. Umans, Ed., IEEE Computer Society Press, pp. 622–632.

[70] 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.

[71] Lehmer, D. H. An extended theory of Lucas' functions. *Annals of Mathematics 31*, 3 (1930), 419–448.

[72] Lemke, C. E., and Howson, J. T. Equilibrium points of bimatrix games. *SIAM Journal on Applied Mathematics 12*, 2 (1964), 413–423.

[73] Lenstra, A. K., Lenstra Jr., H. W., Manasse, M. S., and Pollard, J. M. The number field sieve. In *22nd Annual ACM Symposium on Theory of Computing* (Baltimore, MD, USA, May 14–16, 1990), ACM Press, pp. 564–572.

[74] Lipton, R. J. New directions in testing. In *Distributed Computing And Cryptography, Proceedings of a DIMACS Workshop, Princeton, New Jersey, USA, October 4-6, 1989* (1989), pp. 191–202.

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

[76] Mahmoody, M., Smith, C., and Wu, D. J. A note on the (im)possibility of verifiable delay functions in the random oracle model. Cryptology ePrint Archive, Report 2019/663, 2019. `https://eprint.iacr.org/2019/663`.

[77] 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.

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

[79] Micali, S. Computationally sound proofs. *SIAM Journal on Computing 30*, 4 (2000), 1253–1298.

[80] Naor, M., Paneth, O., and Rothblum, G. N. Incrementally verifiable computation via incremental PCPs. In *TCC 2019: 17th Theory of Cryptography Conference, Part II* (Nuremberg, Germany, Dec. 1–5, 2019), D. Hofheinz and A. Rosen, Eds., vol. 11892 of *Lecture Notes in Computer Science*, Springer, Heidelberg, Germany, pp. 552–576.

[81] Nash, J. Non-cooperative Games. *The Annals of Mathematics 54*, 2 (1951), 286–295.

[82] Oren, Y. On the cunning power of cheating verifiers: Some observations about zero knowledge proofs (extended abstract). In *28th Annual Symposium on Foundations of Computer Science* (Los Angeles, CA, USA, Oct. 12–14, 1987), IEEE Computer Society Press, pp. 462–471.

[83] Paillier, P. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology – EUROCRYPT'99* (Prague, Czech Republic, May 2–6, 1999), J. Stern, Ed., vol. 1592 of *Lecture Notes in Computer Science*, Springer, Heidelberg, Germany, pp. 223–238.

[84] 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.

[85] Papadimitriou, C. H. *Computational complexity*. Academic Internet Publ., 2007.

[86] Pass, R., and Venkitasubramaniam, M. A round-collapse theorem for computationally-sound protocols; or, TFNP is hard (on average) in pessiland. Cryptology ePrint Archive, Report 2019/754, 2019. `https://eprint.iacr.org/2019/754`.

[87] Peikert, C., and Shiehian, S. Noninteractive zero knowledge for NP from (plain) learning with errors. In *Advances in Cryptology – CRYPTO 2019, Part I* (Santa Barbara, CA, USA, Aug. 18–22, 2019), A. Boldyreva and D. Micciancio, Eds., vol. 11692 of *Lecture Notes in Computer Science*, Springer, Heidelberg, Germany, pp. 89–114.

[88] Pietrzak, K. Simple verifiable delay functions. In *ITCS 2019: 10th Innovations in Theoretical Computer Science Conference* (San Diego, CA, USA, Jan. 10–12, 2019), A. Blum, Ed., vol. 124, LIPIcs, pp. 60:1–60:15.

[89] Pointcheval, D., and Stern, J. Security arguments for digital signatures and blind signatures. *Journal of Cryptology 13*, 3 (June 2000), 361–396.

[90] Pollard, J. M. A monte carlo method for factorization. *BIT Numerical Mathematics 15*, 3 (Sep 1975), 331–334.

[91] Rabin, M. O. Digitalized signatures and public-key functions as intractable as factorization. Tech. rep., USA, 1979.

[92] Regev, O. On lattices, learning with errors, random linear codes, and cryptography. In *37th Annual ACM Symposium on Theory of Computing* (Baltimore, MA, USA, May 22–24, 2005), H. N. Gabow and R. Fagin, Eds., ACM Press, pp. 84–93.

[93] Reingold, O., Rothblum, G. N., and Rothblum, R. D. Constant-round interactive proofs for delegating computation. In *48th Annual ACM Symposium on Theory of Computing* (Cambridge, MA, USA, June 18–21, 2016), D. Wichs and Y. Mansour, Eds., ACM Press, pp. 49–62.

[94] Rivest, R. L., Shamir, A., and Wagner, D. A. Time-lock puzzles and timed-release crypto. Tech. rep., Cambridge, MA, USA, 1996.

[95] 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.

[96] Sahai, A., and Waters, B. How to use indistinguishability obfuscation: deniable encryption, and more. In *46th Annual ACM Symposium on Theory of Computing* (New York, NY, USA, May 31 – June 3, 2014), D. B. Shmoys, Ed., ACM Press, pp. 475–484.

[97] Savani, R., and von Stengel, B. Exponentially many steps for finding a Nash equilibrium in a bimatrix game. In *45th Annual Symposium on Foundations of Computer Science* (Rome, Italy, Oct. 17–19, 2004), IEEE Computer Society Press, pp. 258–267.

[98] Schnorr, C. Optimal algorithms for self-reducible problems. In *ICALP* (1976), Edinburgh University Press, pp. 322–337.

[99] Schwartz, J. T. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM 27*, 4 (Oct. 1980), 701–717.

[100] Sotiraki, K., Zampetakis, M., and Zirdelis, G. PPP-completeness with connections to cryptography. In *59th Annual Symposium on Foundations of Computer Science* (Paris, France, Oct. 7–9, 2018), M. Thorup, Ed., IEEE Computer Society Press, pp. 148–158.

[101] Sperner, E. Neuer beweis für die invarianz der dimensionszahl und des gebietes. *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg 6*, 1 (Dec 1928), 265–272.

[102] Tovey, C. A. A simplified NP-complete satisfiability problem. *Discrete Applied Mathematics 8*, 1 (1984), 85–89.

[103] 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.

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

[105] Wesolowski, B. Efficient verifiable delay functions. In *Advances in Cryptology – EUROCRYPT 2019, Part III* (Darmstadt, Germany, May 19–23, 2019), Y. Ishai and V. Rijmen, Eds., vol. 11478 of *Lecture Notes in Computer Science*, Springer, Heidelberg, Germany, pp. 379–407.

[106] Zippel, R. Probabilistic algorithms for sparse polynomials. In *EUROSAM* (1979), vol. 72 of *Lecture Notes in Computer Science*, Springer, pp. 216–226.

Chapter A
Missing Proofs

A.1 Proof of Lemma 5

Lemma 5 (Bad queries are hard to find)**.** *For any $N = p \cdot q$ where $p = 2p' + 1, q = 2q' + 1$ are $(\lambda_{RSA}/2)$-bit safe primes, the following holds: any adversary that makes at most Q queries to the random oracle $H : \{0,1\}^* \to \{0,1\}^\lambda$ will make a bad query with probability at most $3 \cdot Q/2^\lambda$*

Proof (of Lemma 5). Recall that a query is a tuple (μ, x, y, T) where $\mu, x, y \in QR_N^+$ and $T \in \mathbb{N}$, 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 $y' := \mu^r \circ y$. We show that

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

This suffices as H is modelled as a random oracle Using $\Pr[a \vee b] = \Pr[a \wedge \overline{b}] + \Pr[b]$, this can be rewritten as

$$\Pr_r[(y' = x'^{2^{T/2}}) \wedge (x' \in QR_N^\star)] + \Pr_r[x' \notin QR_N^\star] \leq 3/2^\lambda. \tag{A.1}$$

We bound the two probabilities separately in Claims 14.4 and 14.5, and the lemma follows by a union bound over all the queries. □

Claim 14.4. $\Pr_r[x' \notin QR_N^\star] \leq 2/2^\lambda$.

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 $\left|QR_N^+\right| = p'q'$). 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 if

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

Note that this is uses the fact that $2^\lambda < \min(p', q')$, which is a consequence of the choice of $p = 2p' + 1$ and $q = 2q' + 1$ as $\lambda_{\text{RSA}}/2$-bit primes: as noted in Remark 4, λ_{RSA} is usually (much) larger than the corresponding security parameter λ. Clearly $|\mathcal{B}| \leq 2$ and the claim follows. □

Claim 14.5. $\Pr_r[(y' = x'^{2^{T/2}}) \wedge (x' \in QR_N^\star)] \leq 1/2^\lambda$.

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

$$\Pr_r[y' = {x'}^{2^{T/2}} \wedge x' \in QR_N^\star] = \Pr_r[y' = {x'}^{2^{T/2}} \mid x' \in QR_N^\star] \cdot \Pr_r[x' \in QR_N^\star] \tag{A.3}$$

For the second factor in eq.(A.3) we have with $\mathcal{B}$ as in eq.(A.2)

$$\Pr_r[x' \in QR_N^\star] = \frac{2^\lambda - |\mathcal{B}|}{2^\lambda} \ . \tag{A.4}$$

Conditioned on $x' \in QR_N^\star$ the r is uniform in $\mathbb{Z}_{2^\lambda} \setminus \mathcal{B}$, so the first factor in eq.(A.3) is

$$\Pr_r[y' = {x'}^{2^{T/2}} \mid x' \in QR_N^\star] = \Pr_{r \leftarrow \mathbb{Z}_{2^\lambda} \setminus \mathcal{B}}[y' = {x'}^{2^{T/2}}] \ . \tag{A.5}$$

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

$$r(e_\mu - 2^{T/2}) + e_y - e_\mu 2^{T/2} = 0 \bmod p'q' \ . \tag{A.6}$$

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.(A.6) becomes

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

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

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

and the claim follows from the above equation and eq.(A.3)-eq.(A.5) as

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

□