

On the Average-Case Hardness of Total Search Problems

by

Chethan Kamath

January, 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 Guët, IST Austria, Klosterneuburg, Austria

Signature: _____

© by Chethan Kamath, January, 2020
All Rights Reserved

The copyright of this thesis rests with the author. Unless otherwise indicated, its contents are licensed under a Creative Commons Attribution 4.0 International License.

Under this license, you may copy and redistribute the material in any medium or format. You may also create and distribute modified versions of the work. This is on the condition that: you credit the author and do not use it, or any derivative works, for a commercial purpose.

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
January, 2020

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 back traced 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 our group over the years (Benedikt, Guille, Hamza, Joël, Karen, Maciej, Michal, Michael, Michelle, Peter and the numerous interns), 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.

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

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.

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

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*. 51st Annual ACM SIGACT Symposium on Theory of Computing – STOC 2019:1103–1114.
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, 2019.

Table of Contents

Abstract	v
Acknowledgments	vi
About the Author	vii
List of Publications	viii
List of Figures	x
List of Abbreviations	xi
1 Introduction	xii
1.1 Total Search Problems	1
1.2 Cryptography and Total Search Problems	11
1.3 Overview of the Thesis	15
2 Background	26
2.1 Definitions	27
2.2 Hardness in PPAD	29
2.3 Hardness in CLS	33
2.4 Relaxing the SVL Problem	34
3 PPAD-Hardness from Iterated-Squaring	37
3.1 Hardness Assumption	37
3.2 Pietrzak’s Proof System	41
3.3 The Reduction	48
4 PPAD-Hardness and the Fiat-Shamir Methodology	60
4.1 Overview	61
4.2 The Sumcheck Protocol	70
4.3 The Reduction	79
4.4 Instantiating Fiat-Shamir	90
5 Conclusions and Future Directions	99
Bibliography	103
A Missing Proofs	113
A.1 Proof of Lemma 5	113

List of Figures

1.1	Sperner's Lemma	2
1.2	The TFNP landscape	5
1.3	Sample LONELY instance	6
1.4	Problems in TFNP	8
1.5	Sample END-OF-LINE instance	10
1.6	SPERNER is Karp-reducible to END-OF-LINE	11
1.7	The Fiat-Shamir Transform	17
1.8	Sample SINK-OF-VERIFIABLE-LINE instance	19
1.9	The halving sub-protocol	21
2.1	Simulating S and P using reversible pebbling.	32
2.2	Sample RELAXED-SINK-OF-VERIFIABLE-LINE instance	34
3.1	Recursive structure of the RSVL instance.	53
3.2	Labels of RSVL instance.	54
4.1	The Sumcheck Protocol	72
4.2	Sumcheck Protocol with prefixes.	73
4.3	Non-interactive Sumcheck Protocol with prefixes.	77
5.1	Factoring and its related assumptions.	101

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

Cryptographic primitives:

- OWF** One-Way Function
- OWP** One-Way Permutation
- PRG** 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
- PPT** Probabilistic Polynomial Time

1 Introduction

1.1 Total Search Problems

Consider the following¹ 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 function²

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

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 some circuit

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

that is of size **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 an algorithm to be deemed efficient, it would now have to run in time **poly**(m) and therefore the brute force approach from before is no longer efficient. Let's call this problem SPERNER.

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

²Throughout, we use $[a, b]$ to denote $\{a, a + 1, \dots, b\}$.

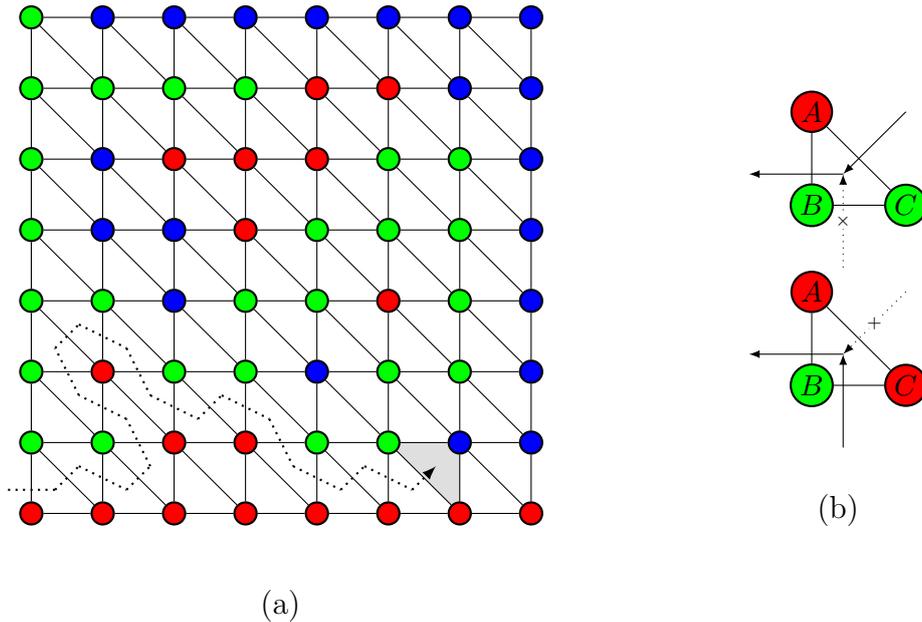


Figure 1.1: (a) Sperner's Lemma on an 8×8 grid. The walk is continued by always taking a green-red edge with the green vertex on the left (invariant) (b) Why the invariant rules out the walk looping.

Totality of Sperner. Although SPERNER may at first seem like any other search problem, it has one distinguishing feature: a theorem of Sperner [89] 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 triangle always taking a *green-red edge with the green vertex on the left* — this is an invariant. Let's see what can be said about the walk.

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 $\triangle ABC$ 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 $\triangle ABC$, 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 $\triangle ABC$ 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. 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's formalise these notions a bit more.

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 $\mathcal{S}_I \subseteq \{0, 1\}^*$. It is an **NP** search problem if the set of solutions is *also* efficiently-decidable. 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 [69] defined the class *Total Functional Non-deterministic Polynomial-time* (**TFNP**). They went on to show that this class is in fact the same as **FNP** \cap **co-FNP**,³ where **co-FNP** is the search counterpart of **co-NP** (i.e., a counterexample can be efficiently verified).

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 [72], 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 motivations 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.

³To show that **TFNP** = **FNP** \cap **co-FNP**, it is easier if we use the alternative definition of **NP** search problems using relations [69]. Let $\mathcal{R} \subseteq \{0, 1\}^* \times \{0, 1\}^*$ denote a polynomially-balanced search relation (i.e., $(I, S) \in \mathcal{R}$ implies that $|S| = \text{poly}(|I|)$). A polynomially-balanced relation \mathcal{R} defines a **NP** search problem if \mathcal{R} is polynomial-time computable. Moreover \mathcal{R} defines a total search problem if for every instance I , there exists a solution S such that $(I, S) \in \mathcal{R}$. It is easy to show one direction of the containment, i.e. **TFNP** \subseteq **FNP** \cap **co-FNP**: given a relation \mathcal{R} that defines a problem in **TFNP** define the **FNP** relation as $\mathcal{R}_1 = \mathcal{R}$ and set $\mathcal{R}_2 = \emptyset$. In the other direction, i.e. to show that **FNP** \cap **co-FNP** \subseteq **TFNP**, we proceed as follows. Let \mathcal{R}_1 and \mathcal{R}_2 denote **FNP** and **co-FNP** relations corresponding to a search problem. Since we know that for each I there exists either S such that either $(I, S) \in \mathcal{R}_1$ or $(I, S) \in \mathcal{R}_2$, one can define the corresponding **TFNP** relation as $\mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2$.

1.1.1.1 The Subclasses of TFNP

It is known that **TFNP** is unlikely to contain **NP**-complete problems unless $\mathbf{NP} = \mathbf{co-NP}$ [69].⁴ Moreover, **TFNP** is a *semantic* class and therefore unlikely to have a complete problem.⁵ 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 [75]. For instance, one of the subclasses is defined based on the 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 7) to this canonical problem. Therefore in order to study the class in its entirety, it suffices to focus on the canonical complete problem.

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 [5]. More formally, it was shown in [5] that these classes are separate with respect to (even) Turing reductions (Definition 8): i.e., there exist oracles relative to which the complete problem of one is not 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 balls are placed in $m < n$ bins then at least one bin must contain more than one ball.* The hardness of this class is captured by its canonical complete problem PIGEON, which is described below.

Definition 1. PIGEON

Instance. A circuit $C : \{0, 1\}^m \rightarrow \{0, 1\}^m$

Solution. One of the following:

1. Preimage of 0^m : $x \in \{0, 1\}^m$ such that $C(x) = 0^m$

⁴Suppose for contradiction that **TFNP** contains the **NP**-complete SAT. On input a SAT instance Φ , the algorithm return an assignment x . But if x does not satisfy Φ the we know that Φ is not satisfiable and therefore solve the **co-NP**-complete problem UNSAT.

⁵A 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 [76]. 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.

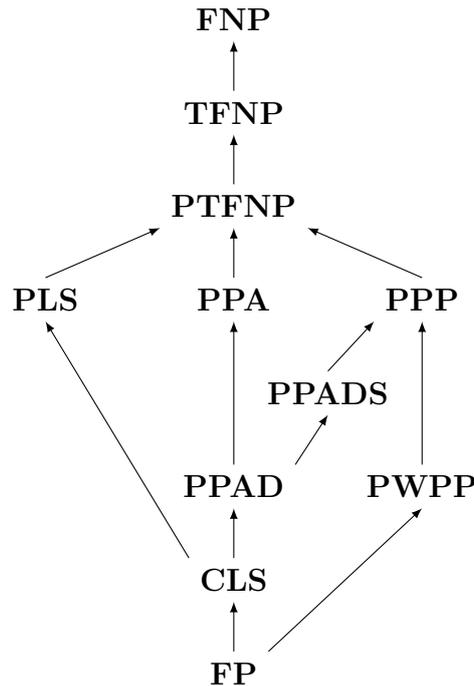


Figure 1.2: The **TFNP** landscape. The arrows indicate containment, and this relationship is transitive. An absence of arrow is to be interpreted as a separation between the classes. The dotted lines indicate that neither containment nor separation is known between the classes.

2. Collision: $x, y \in \{0, 1\}^m$ such that $C(x) = C(y)$

An interesting problem that lies inside the class (but is not known to be complete) is EQUALSUMS [3]: given $a_1, \dots, a_n \in \mathbb{N}$ such that $\sum_{i \in [n]} a_i < 2^n - 1$, find two subsets $S \neq T \subseteq [n]$ such that $\sum_{i \in S} a_i = \sum_{i \in T} a_i$. Recently, it was shown that a problem related to Blichfeldt’s theorem on lattices (BLICHFELDT) is complete for **PPP** [88]. This is the first *natural* problem that was shown to be complete for the class.⁶

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.⁷

Definition 2. LONELY

Instance. A circuit $C : \{0, 1\}^m \rightarrow \{0, 1\}^m$

⁶A 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.

⁷We 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** [5]. Although they are also canonical in nature, their definition relies on a circuit that maps n bits to $O(n)$ or **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.

Guarantee. 0^m is unpaired: $C(0^m) = 0^m$

Solution. One of the following:

1. Unpaired vertex: $x \in \{0, 1\}^m \setminus \{0^m\}$ such that $C(x) = x$
2. Violation: $x, y \in \{0, 1\}^m$ such that $C(x) = y$ but $C(y) \neq x$

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 $C(x) = C(y)$ 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 Tucker's lemma [37] and Necklace Splitting [41] from combinatorics. The aforementioned problems are also known to be complete for **PPA**.

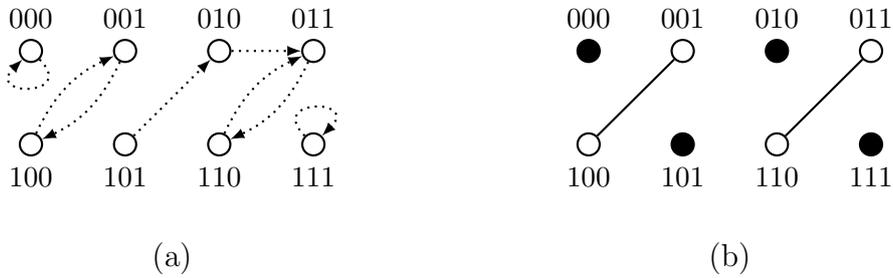


Figure 1.3: Sample LONELY instance with eight vertices $\{0, 1\}^3$. The output of the circuit is indicated in (a) by the dotted, directed edges: i.e., an edge (u, v) implies that $C(x) = y$. The solid, undirected edges are the edges in the implicit (involution) graph shown in (b): i.e. an edge (u, v)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

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 acyclic graph (DAG) has a sink*. The canonical complete problem for the class is called LOCAL-SEARCH (LS) and is described below [58].

Definition 3. LOCAL-SEARCH (LS)

Instance.

1. A successor circuit $S : \{0, 1\}^m \rightarrow \{0, 1\}^m$
2. A potential circuit $F : \{0, 1\}^m \rightarrow [2^m]$

Solution. Local optimum: $x \in \{0, 1\}^m$ such that $F(x) \geq F(S(x))$

Intuitively S and F , together, implicitly define a DAG where (x, y) is an edge if $y = S(x)$ and $F(y) \geq F(x)$, 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 $\mathbf{PPA} \cap \mathbf{PPP}$ 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** belong 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 by Daskalakis, Goldberg and Papadimitriou [35], and Chen, Deng and Teng [30]. **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 $\mathbf{CLS} \subseteq \mathbf{PLS} \cap \mathbf{PPAD}$ contains “continuous” variants of the problems in **PLS** [36]. 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 [39]. Another interesting problem that lies in this class (but not known to be complete) is **END-OF-METERED-LINE (EOML)** [56]. 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)* [57] 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 [48]. It contains all of the subclasses that we have discussed above, but the membership of some of the “rogue” **TFNP** problems (viz. **FACTORING**, **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.

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* [75]. This problem can be solved in linear time when the graph is given explicitly, but there is no known

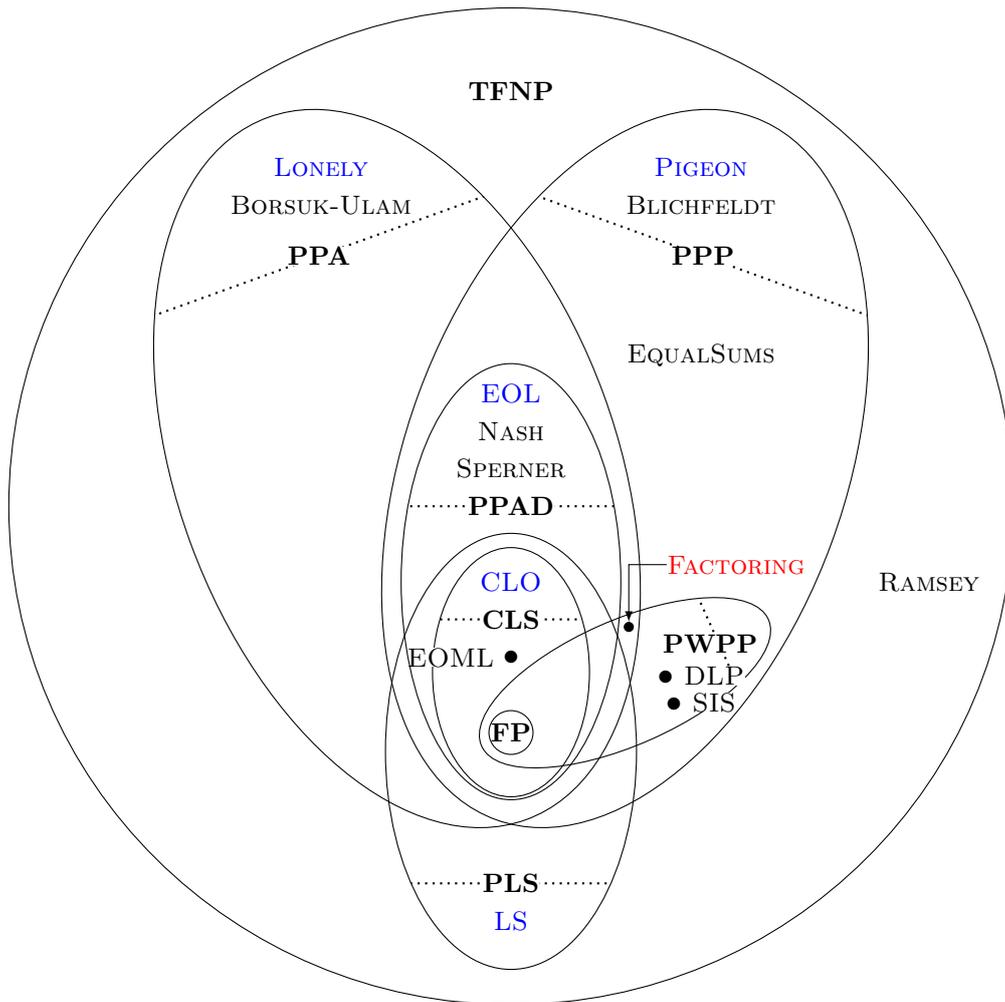


Figure 1.4: Problems in **TFNP**. A problem that lies above dotted line is complete for that class. The canonical complete problems for their classes are in blue. Inclusions that are not established using a Karp reduction (e.g., randomised or Turing) are in red.

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. END-OF-LINE (EOL)

Instance.

1. A successor circuit $S : \{0, 1\}^m \rightarrow \{0, 1\}^m$
2. A predecessor circuit $P : \{0, 1\}^m \rightarrow \{0, 1\}^m$

Guarantee. 0^m is unbalanced: $P(0^m) = 0^m$ and $S(0^m) \neq 0^m$

Solution. An unbalanced vertex $v \in \{0, 1\}^m$: $P(S(v)) \neq v$ or $S(P(v)) \neq v \neq 0^m$

To see why totality is not semantic but syntactic, we have to first explain a few technicalities [47]. Note that any badly defined edge, i.e. $S(u) = v$ and $P(v) \neq u$ or $P(v) = u$ and $S(u) \neq v$, qualifies as a solution of EOL as defined above (because $P(S(u)) \neq u$ or $S(P(u)) \neq u$ respectively: see Figure 1.5.(a). Note that 0^m is a source of the graph, unless $P(S(0^m)) \neq 0^m$, in which case 0^m is a valid solution to the problem as stated above. One could now think of “sanitised” EOL instance (S', 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 $S'(v) = u$ and $P'(u) = v$ (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 $C(x) := S(x)$. 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 [5].

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.

Theorem. SPERNER \in PPAD [75].

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's 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

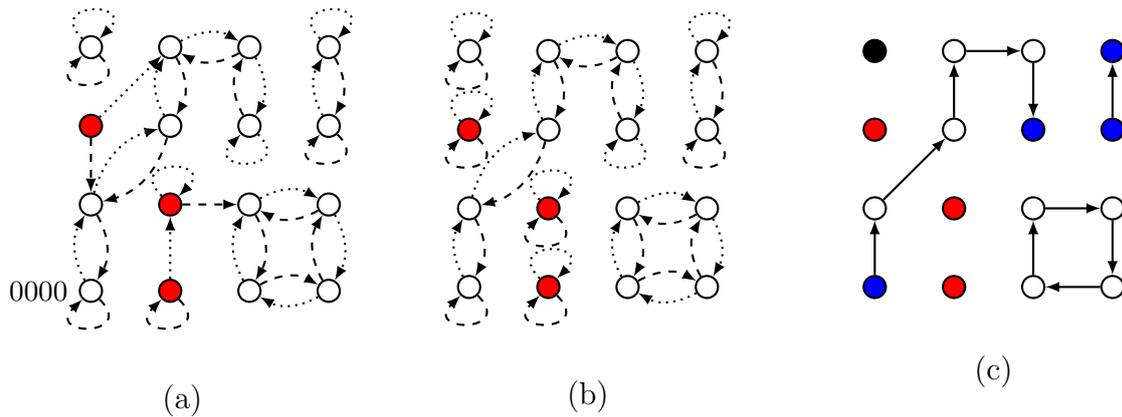


Figure 1.5: Sample END-OF-LINE instance with 16 vertices $\{0, 1\}^4$. The output of the successor (resp., predecessor) circuit is indicated in (a) by the dotted (resp., dashed), directed edges: i.e., a dotted (resp., dashed) edge (u, v) implies that $S(u) = v$ (resp., $P(u) = v$). The vertices having badly defined edges are shown in red. The graph in (b) corresponds the “sanitised” EOL instance (S', 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 $S(x) = y$ and $P(y) = x$. The source and sink vertices are shown in blue; self-loops are in black.

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’s 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 $C : \{0, 1\}^m \times \{0, 1\}^m \rightarrow \{0, 1, 2\}$. 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

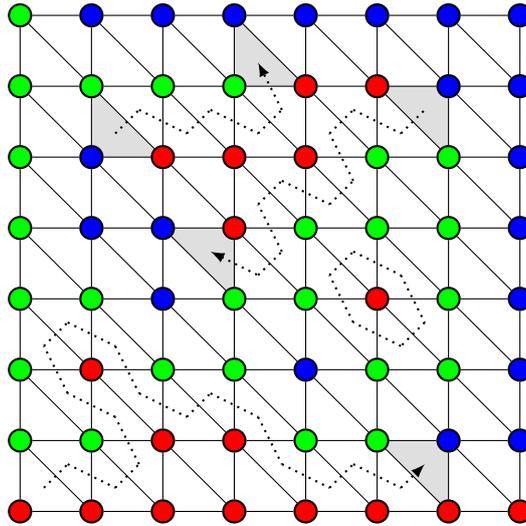


Figure 1.6: SPERNER is Karp-reducible to END-OF-LINE. The END-OF-LINE instance that is constructed out of the SPERNER instance is shown using dotted lines. The source and sink of the EOL instance corresponds to the trichromatic triangles (which are shaded). The triangles which are not the part of any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

from any unbalanced vertex in the EOL instance, we can recover the coordinates of a trichromatic triangle, completing the reduction. \square

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.⁸ 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 Lemke-Howson algorithm, which efficiently solves most instances of NASH that occur in practice [65]. In light of this, it is natural to seek “extrinsic” evidence supporting **TFNP** hardness. That is, as raised in Papadimitriou’s original paper [75], 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?

⁸Currently, 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 $\mathbf{TFNP} \subset \mathbf{FP}$) We do however know of worst-case hard instances for a *specific algorithm* for finding a Nash equilibrium [87] or of *oracle-based* worst-case hard instances of **PPAD**-complete problems [53; 2].

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 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** [75]. Let (π, y^*) denote an instance of OWP, where $\pi : \{0, 1\}^m \rightarrow \{0, 1\}^m$ 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

$$C_{y^*}(x) := \pi(x) \oplus y^*$$

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

$H : \{0, 1\}^m \rightarrow \{0, 1\}^{m-1}$ that shrinks the input by one bit, the Karp reduction maps it to the PIGEON circuit

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

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) imply CRHFs, it follows that they also give rise to **PPP**-hardness.

3. FACTORING implies hardness of **PPA** albeit via *randomised* reductions [23; 57]. Given an instance $N \in \mathbb{N}$ of FACTORING the reduction, *roughly speaking*, defines the LONELY instance as

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

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 $\text{GCD}(x - 1, N)$.

In addition, Hubáček, Naor and Yogev [55] recently 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. Though, it is not known whether their distribution gives rise to average-case hardness in any of the syntactic subclasses of **TFNP**. Komargodski, Naor and Yogev [62] 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 [58; 69] was later strengthened to show that it is unlikely to base average-case **TFNP** hardness even on problems in the polynomial hierarchy [22], 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 [68]. A recent result [85] 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 [4]. That is, given a program P represented in an appropriate model of computation (say, Turing machines or Boolean circuits), the obfuscator returns its scrambled version \widehat{P} such that $P(x) = \widehat{P}(x)$ 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{P} should not reveal any information on P other than its input-output behaviour. It was shown in [4] 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) [86]. Moreover, many candidate constructions were proposed, albeit from new, non-standard hardness assumptions [43].

PPAD-hardness via obfuscation. The connection between obfuscation 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 [7].

Bitansky, Paneth and Rosen [12], 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 from [43].

The result in [12] was extended by Hubáček and Yagev [56], 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. First, by Garg, Pandey and Srinivasan [44], who reduced from breaking IO with polynomial (instead of sub-exponential) hardness (or alternatively

compact public-key functional encryption) and one-way permutations. Second, by Komargodski and Segev [63], who 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 problem that we call **ITERATED-SQUARING**, and requires a random oracle, i.e. a random function that is accessible to all parties [6]. 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**), described next. For some time parameter $T \in \mathbb{N}$, **IS** involves computing the function

$$f(N, x, T) = x^{2^T} \bmod N, \quad (1.2)$$

where N is the product of two random $\lambda/2$ -bit safe primes (where p is a safe prime if $(p - 1)/2$ is also a prime) and $x \in \mathbb{Z}_N^*$. Computing f was suggested as a hard problem by Rivest, Shamir and Wagner[84], who conjectured that for any T , computing $f(N, x, T)$ either requires $\Omega(T)$ *sequential* time or total computation sufficient to factor N . Our hardness assumption on **IS** is milder: it is sufficient for us that eq.(1.2) cannot be computed in time $\text{poly}(\lambda)$ 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 [78] 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 generated as described above. If there exists a time parameter $T = T(\lambda) \leq 2^\lambda$ such that no $\mathbf{poly}(\lambda)$ -time algorithm, on input (N, x, T) , can compute $x^{2^T} \bmod N$ except with negligible probability then, relative to a random oracle, there exists a hard distribution of END-OF-LINE instances. That is, ITERATED-SQUARING is Turing-reducible to END-OF-LINE relative to a random oracle.*

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 [40], 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 to 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 $i \in [\ell]$, 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 [67] to show that #SAT $\in \mathbf{IP}$. 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

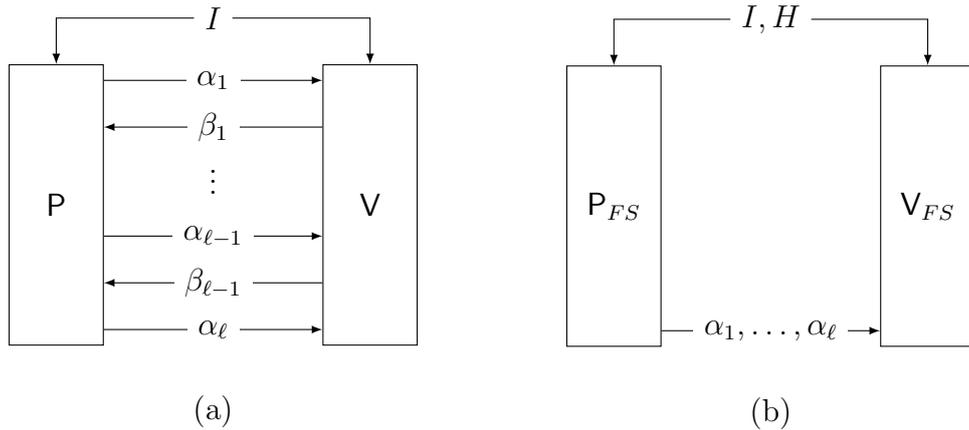


Figure 1.7: The Fiat-Shamir Transform. (a) An ℓ -round interactive protocol between a prover P and a verifier V . I denotes the statement that the prover is trying to convince the verifier about. The i -th message from the prover to the verifier is denoted by α_i ; the i -th message in the opposite direction 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. $H(I, \alpha_1, \beta_1, \dots, \alpha_{i-1})$. Therefore only the prover messages $\alpha_1, \dots, \alpha_\ell$ are now required to be sent.

reducible to $\#\text{SAT}$ (in fact to SAT) it follows that efficiently solving the above distribution is no easier than breaking Fiat-Shamir.

Theorem (informal). *Solving the END-OF-LINE problem is no easier than breaking the (adaptive) soundness of the Fiat-Shamir Transformation, when applied to the Sumcheck Protocol.*

Moreover, we show that the soundness indeed holds when H is instantiated with a random oracle.⁹ Therefore, we get the following theorem and its corollary.

Corollary (informal). *If $\#\text{SAT}$ is hard (in the worst case) then, relative to a random oracle, there exists a hard distribution of END-OF-LINE instances.¹⁰*

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 [24]. Therefore, it is plausible that the hardness of our construction is based on 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 [56]. We discuss more on this in §2.3 in the next chapter.

⁹In 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 [78].

¹⁰In the original paper [31], we had claimed that the corollary holds just relative to a random oracle, without the additional assumption on $\#\text{SAT}$. However, this is not correct: see Remark 8.

1.3.2 Techniques

In both constructions, we follow the blueprint from [1; 12] that we discussed in §1.2.2. That is, we proceed via the following two steps:

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 [91].

1.3.2.1 Sink-of-Verifiable-Line

Definition 5. SINK-OF-VERIFIABLE-LINE (SVL)

Instance.

1. Length $L \in \{1, \dots, 2^m\}$
2. A successor circuit $S : \{0, 1\}^m \rightarrow \{0, 1\}^m$
3. A verifier circuit $V : \{0, 1\}^m \times \{1, \dots, L\} \rightarrow \{0, 1\}$

Promise. For every $v \in \{0, 1\}^m$ and $i \in \{1, \dots, L\}$: $V(v, i) = 1$ iff $v = S^i(0^m)$.

Solution. Sink vertex: $v \in \{0, 1\}^m$ such that $V(v, L) = 1$

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 (S, V, L) is a *valid* SVL instance since V might not satisfy the promise about its behaviour: see Footnote 5. Moreover, there may not be an efficient algorithm for verifying whether a given tuple (S, 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 ps being presently tested).

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 5: see Lemma 1 and Remark 1.

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 \mathbf{S} is the squaring modulo N function, yielding the standard path:

$$x \rightarrow x^2 \rightarrow x^{2^2} \rightarrow x^{2^3} \cdots \rightarrow x^{2^T} \pmod{N}. \quad (1.4)$$

Notice that, assuming $x^{2^T} \pmod{N}$ 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 \mathbf{V} that certifies that a vertex $v = y$ is obtained by invoking \mathbf{S} for i successive times on x . This is where Pietrzak's proof system for certifying $y = f(N, x, T)$ comes into play [78].

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 $y = x^{2^T} \pmod{N}$ using $t = \log T$ rounds of interaction. It does not require either prover or verifier to know the factorization of N . The protocol is recursive in the time parameter T . In the first step, the prover sends the midpoint $\mu = x^{2^{T/2}} \pmod{N}$ as a commitment to the verifier. If

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

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

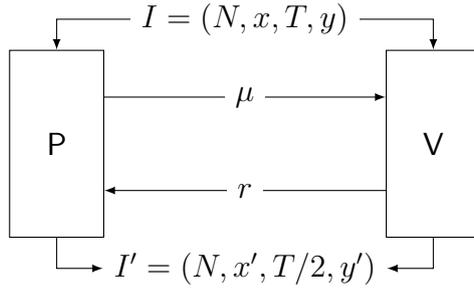


Figure 1.9: The halving sub-protocol between a prover P and a verifier V for $T > 2$ results in a statement I turned into a new statement I' .

by applying a hash function H to the prover’s messages. The non-interactive proof on challenge (N, x, T) is of the form $(N, x, T, y, \mu_1, \dots, \mu_t)$, and we denote it by $\pi(x \xrightarrow{T} y)$. We point out the following four 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’ll 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 $\mathbf{poly}(n, \log \ell)$ 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 (1.4). A first attempt would be to augment the vertex labels $x_i = x^{2^i} \bmod N$ in (1.4) with a corresponding proof, i.e., consider the standard path

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

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 $\pi(x_0 \xrightarrow{i+1} x_{i+1})$. 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:

$$\begin{aligned} &\pi(x_0 \xrightarrow{0} x_0) \rightarrow \pi(x_0 \xrightarrow{1} x_1) \rightarrow \pi(x_0 \xrightarrow{2} x_2) \rightarrow \pi(x_0 \xrightarrow{2} x_2) \parallel \pi(x_2 \xrightarrow{1} x_3) \rightarrow \\ &\pi(x_0 \xrightarrow{4} x_4) \rightarrow \pi(x_0 \xrightarrow{4} x_4) \parallel \pi(x_4 \xrightarrow{1} x_5) \rightarrow \dots \end{aligned}$$

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 \mathbf{S} and verification circuit \mathbf{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 \mathbf{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 \mathbf{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 \mathbf{S} . The above (recursive) construction can be thought of as an *incrementally-verifiable* procedure [91] 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. Note that Pietrzak’s protocol has only *logarithmic* number of rounds compared to polynomial number of rounds in the Sumcheck Protocol. Hence, it is potentially easier to instantiate (with a concrete hash function) the Fiat-Shamir Transform for the former than the latter. Taken together, one could argue that the two results are in some sense incomparable.

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 5). 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 [91], 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

$$C = C_0 \rightarrow \dots \rightarrow C_T$$

denote the computation that we would like to carry out in an incrementally-verifiable manner. Let’s 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):

$$\begin{aligned} \pi(C_0 \xrightarrow{0} C_0) \rightarrow \pi(C_0 \xrightarrow{1} C_1) \rightarrow \pi(C_0 \xrightarrow{2} C_2) \rightarrow \pi(C_0 \xrightarrow{2} C_2) \parallel \pi(C_2 \xrightarrow{1} C_3) \rightarrow \\ \pi(C_0 \xrightarrow{4} C_4) \rightarrow \dots \rightarrow \pi(C_0 \xrightarrow{T/2} C_{T/2}) \parallel \dots \parallel \pi(C_{T-2} \xrightarrow{1} C_{T-1}) \rightarrow \pi(C_0 \xrightarrow{T} C_T). \end{aligned}$$

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 [91] was constructing a proof system that allows efficient merging using a technique called *recursive proof-composition*. The construction in [91] 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 [9; 46]. 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.¹²

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**.¹³

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.

¹²The issue is that Valiant's proof system cannot be composed to prove statements about a non-explicit oracle.

¹³A key ingredient in Valiant's construction is a CS proof [70] 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 [71] has managed to construct incrementally-verifiable PCPs, but their construction is not unambiguous either.

In particular, we focus on the results in [12] and [56] 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 [78], 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 [32].

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 [67], 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 [24], 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 [31].

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.

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 [12] and [56], 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 recall some prerequisite formal definitions in §2.1.

2.1 Definitions

The definitions of total search problems are originally from [69] (see Footnote 3). In this thesis we prefer to use an alternative, equivalent formulation from [56].

Definition 6 (Search problem). A search problem P consists of an efficiently-decidable set of instances¹ $\mathcal{I} \subseteq \{0,1\}^*$ such that for every instance $I \in \mathcal{I}$ there exists a set of solutions $\mathcal{S}_I \subseteq \{0,1\}^*$. On input an instance $I \in \mathcal{I}$, an algorithm A is said to *solve* P if it:

1. outputs a solution $S \in \mathcal{S}_I$; or
2. outputs \perp if $\mathcal{S}_I = \emptyset$ (i.e., if no solution exists).

Next, we 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 [69]. Note 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.

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

Definition 7 (Karp-reducibility). A search problem P is said to (polynomial-time) Karp-reduce to another search problem P' if there exists two polynomial-time computable functions f and g such that

1. f maps every instance $I \in \mathcal{I}$ of P to a valid instance $I' \in \mathcal{I}'$ of P' ; and
2. g maps every solution $S' \in \mathcal{S}'_{I'}$ to the instance I' to a solution $S := g(S') \in \mathcal{S}_I$ of I .

Definition 8 (Turing-reducibility). A search problem P is said to (polynomial-time) Turing-reduce to another search problem P' if there exists a polynomial-time Turing machine R that solves P (as defined in Definition 6) when given oracle-access to a solver A for P' .

Finally, we formally define a search problems and the notion of average-case hardness that we use to study them. It is followed by the formal definition of a total search problem and its average-case hardness, where we add the additional constraint that every instance has a solution.

Definition 9 (**FNP**). A search problem 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 $\mathcal{S}_I \subseteq \{0, 1\}^{\text{poly}(|I|)}$, given a candidate solution $S \in \mathcal{S}_I$ it is decidable in polynomial time (in $|I|$) whether $S \in \mathcal{S}_I$. The class of all **NP** search problems is called *Functional Non-deterministic Polynomial-time* (**FNP**). Alternatively, **FNP** can be defined as the set of all **NP** search problems that are Karp-reducible to computational SAT.

Definition 10 (Hard-on-average search problem). A search problem P is hard-on-average if there exists an efficient instance-sampling algorithm l such that: for every efficient adversary A , there exists a negligible function² μ such that for every $\lambda \in \mathbb{N}$

$$\Pr_{l, A} [S \in \mathcal{S}_I | I \leftarrow l(1^\lambda); S \leftarrow A(I)] \leq \mu(\lambda).$$

Definition 11 (**TFNP**). A problem in **FNP** is *total* if for every instance the set of its solutions is non-empty, i.e., for every $I \in \mathcal{I}$: $\mathcal{S}_I \neq \emptyset$. The class *Total Functional Non-deterministic Polynomial-time* (**TFNP**) consists of the set of all total problems in **FNP**.

Definition 12 (Hard-on-average total search problem). A search problem P is hard-on-average if there exists an efficient instance-sampling algorithm l such that:

1. Every instance has a solution: for every $\lambda \in \mathbb{N}$

$$\Pr_l [\mathcal{S}_I \neq \emptyset | I \leftarrow l(1^\lambda)] = 1.$$

2. For every efficient adversary A , there exists a negligible function μ such that for every $\lambda \in \mathbb{N}$

$$\Pr_{l, A} [S \in \mathcal{S}_I | I \leftarrow l(1^\lambda); S \leftarrow A(I)] \leq \mu(\lambda)$$

In both cases, we say that l defines a *hard distribution* on the instances of problem P .

²Recall that a function $g : \mathbb{N} \rightarrow [0, 1]$ is negligible if for every polynomial $p(n) \in \text{poly}(n)$ there exists an $n_0 \in \mathbb{N}$ s.t. $g(n) \leq p(n)$ for all $n \geq n_0$. g is overwhelming if $1 - g$ is negligible.

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 [75; 12] with 0^m as the standard unpaired vertex as in Definition 4.

Definition 13. END-OF-LINE (EOL)

Instance.

1. A successor circuit $S : \{0, 1\}^m \rightarrow \{0, 1\}^m$
2. A predecessor circuit $P : \{0, 1\}^m \rightarrow \{0, 1\}^m$
3. A vertex $v_0 \in \{0, 1\}^m$

Guarantee. v_0 is unbalanced: $P(v_0) = v_0$ and $S(v_0) \neq v_0$

Solution. An unbalanced vertex $v \in \{0, 1\}^m$: $P(S(v)) \neq v$ or $S(P(v)) \neq v \neq v_0$

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 $(S, P, v_0 = 0^m)$. Second, we can reduce in the opposite direction by shifting the main line by v_0 as follows. Given an EOL instance (S, P, v_0) , define the new EOL instance as (S', P') with source 0^m , where $S'(v) := S(v \oplus v_0)$ and $P'(v) := P(v \oplus v_0)$, where \oplus denotes the bitwise XOR operation. \square

Remark 1. 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. The construction of hard distribution on EOL instances in [12] follows the blueprint that we described in §1.3.2. That is, they proceed via the following two steps:

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 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. $v_i = F_K(i)$. The SVL instance consists of a standard path of length $L = \omega(\mathbf{poly}(\lambda))$ (for a security parameter λ) and the i -th vertex on this path is set as (i, v_i) as shown below.

$$(0, F_K(0)) \rightarrow (1, F_K(1)) \rightarrow (2, F_K(2)) \rightarrow \cdots \rightarrow (L-1, F_K(L-1)) \rightarrow (L, F_K(L))$$

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 (for circuits) IO takes a circuit C as input and returns a functionally-equivalent circuit \widehat{C} . Now, let us reconsider the construction described above but instead with the successor and verifier circuits obfuscated: $\widehat{S} := IO(S)$ and $\widehat{V} = IO(V)$. 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 [86] 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 [12] for further details.

2.2.2 Simulating the Predecessor Circuit

Given an SVL instance $SVL = (S, V, L)$, the END-OF-LINE instance $EOL = (S, P, v_0)$ is essentially simulated using *reversible black pebbling*, a game on directed acyclic graphs that was introduced to model reversible computation [7]. 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. A *reversible black pebbling* of a directed acyclic graph $G = (\mathcal{V}, \mathcal{E})$ is a sequence $\mathcal{P} := (\mathcal{P}_0, \dots, \mathcal{P}_L)$ of pebbling configurations $\mathcal{P}_\ell \subseteq \mathcal{V}$. Two subsequent configurations 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. Starting with an empty graph (i.e., $\mathcal{P}_0 = \emptyset$), the goal of the game is to place a pebble on

the sink nodes. The *space-complexity* of a valid sequence $\mathcal{P} = (\mathcal{P}_0, \dots, \mathcal{P}_L)$ for a DAG G is defined as

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

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

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

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

Lemma 2. SINK-OF-VERIFIABLE-LINE is *Karp-reducible* to END-OF-LINE [1; 12].

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 becomes eventually exponentially large which would render the resulting circuits \mathbf{S} and \mathbf{P} inefficient relative to the size of the SVL instance. To resolve this, we rely on reversible black pebbling.

For an SVL instance $\text{SVL} = (\mathbf{S}, \mathbf{V}, L)$, let us consider the implicitly-defined standard path $\{0^m = v_0, \dots, v_L\}$, where the i -th vertex is $v_i := \text{SVL.S}^i(v_0)$ (see Figure 2.1). Since the EOL instance $\text{EOL} = (\mathbf{S}, \mathbf{P}, \mathbf{v}_0)$ we define, with a standard path $\{\mathbf{v}_0, \dots, \mathbf{v}_L\}$, is determined by the space-optimal pebbling sequence for this standard path, let us look at this particular pebbling sequence. For simplicity, let us consider the case when the length of the standard path is $L = 2^\ell$ for some $\ell \in \mathbb{N}$ (as 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 $2^\ell - 1$, the lone sink. As shown by Chung, Diaconis and Graham [33], the optimal efficient 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 $2^{\ell-1} - 1$ using only $\ell - 1$ pebbles. Then, place an additional pebble at $2^{\ell-1}$. Next, free the first $\ell - 1$ pebbles by reversing the original sequence of moves performed in the first part. Finally, perform the same sequence starting from $2^{\ell-1}$. As illustrated in Figure 2.1, this strategy will end with a pebble at position $2^\ell - 1$ while using only t pebbles.

The standard path implicit in EOL corresponds to the above pebbling sequence, and its successor and predecessor circuits are implemented by simulating the optimal strategy (see Figure 2.1). To be more precise, each vertex \mathbf{v}_i in EOL corresponds to a configuration in the sequence 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

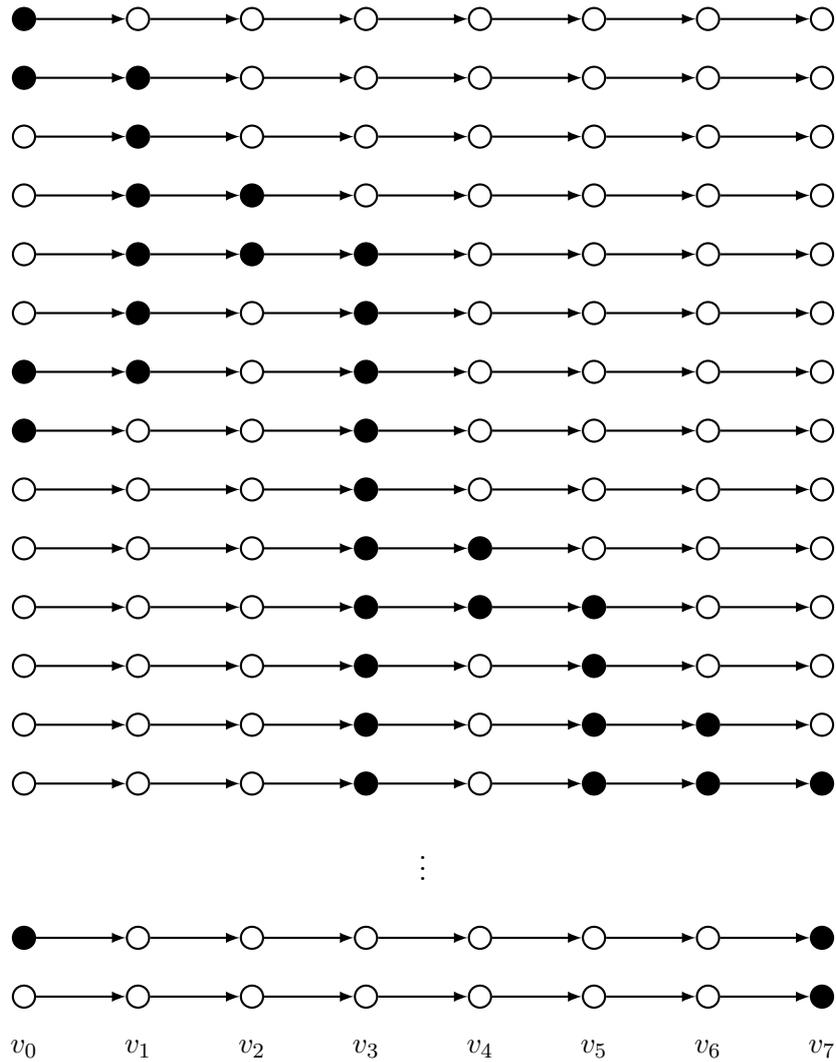


Figure 2.1: Simulating S and P using reversible pebbling. Each path graph above corresponds to an SVL instance (v_0, \dots, v_7) of length 8 (with the other vertices in self-loops). The EOL instance is determined by the space-optimal pebbling sequence for this path graph given above.

$\ell = \log_2(L)$ pebbles, i.e., a tuple of pairs $\mathbf{v}_i := ((v_{i_1}, i_1), \dots, (v_{i_\ell}, i_\ell))$ where each pebble corresponds to a pair (v_i, i) where, if you recall, $v_i = \text{SVL.S}^i(v_0)$. (We presume that configurations with strictly less than ℓ pebbles are padded accordingly.) For example, the labels on the standard path $\mathbf{v}_0, \dots, \mathbf{v}_{27}$ of the EOL instance given in Figure 2.1 are:

$$(v_0, 0) \rightarrow (v_0, 0) \parallel (v_1, 1) \rightarrow (v_1, 1) \rightarrow (v_1, 1) \parallel (v_2, 2) \rightarrow \dots \rightarrow (v_0, 0) \parallel (v_7, 7) \rightarrow (v_7, 7).$$

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 optimal pebbling strategy. If the next move in the sequence 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 efficient pebbling 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 $v_L = \mathbf{S}^L(0^m)$. Note that the length of the standard path in this EOL instance is $L^{\log_2 3}$ and the size of its labels is $2m \cdot \log L$. In particular, every vertex corresponding to an intermediate state of the pebbling strategy is followed by the subsequent state, and the final step of the pebbling strategy is a self-loop under EOL.S. For a more formal description of the reduction see [56]. \square

2.3 Hardness in CLS

Recall that **CLS** consists of all problems that are Karp-reducible to CONTINUOUS-LOCAL-OPTIMUM (see [36] for the formal definition). To establish average-case hardness in **CLS**, however, we rely on END-OF-METERED-LINE (EOML) [56], a problem that is known to lie in **CLS** but not known to be complete.

Definition 15. END-OF-METERED-LINE (EOML)

Instance.

1. A successor circuit $\mathbf{S} : \{0, 1\}^m \rightarrow \{0, 1\}^m$
2. A predecessor circuit $\mathbf{P} : \{0, 1\}^m \rightarrow \{0, 1\}^m$
3. A meter circuit $\mathbf{M} : \{0, 1\}^m \rightarrow \{0, \dots, 2^m\}$

Guarantee.

1. 0^m is unbalanced: $\mathbf{P}(0^m) = 0^m$ and $\mathbf{S}(0^m) \neq 0^m$
2. 0^m is the first vertex: $\mathbf{M}(0^m) = 1$

Solution. A vertex $v \in \{0, 1\}^m$ satisfying one of the following:

- (i) **End of line:** either $\mathbf{P}(\mathbf{S}(v)) \neq v$ or $\mathbf{S}(\mathbf{P}(v)) \neq v \neq 0^m$,
- (ii) **False source:** $v \neq 0^m$ and $\mathbf{M}(v) = 1$,
- (iii) **Miscount:** either $\mathbf{M}(v) > 0$ and $\mathbf{M}(\mathbf{S}(v)) - \mathbf{M}(v) \neq 1$ or $\mathbf{M}(v) > 1$ and $\mathbf{M}(v) - \mathbf{M}(\mathbf{P}(v)) \neq 1$.

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 \mathbf{M} . On input a vertex v , this circuit \mathbf{M} outputs the number of steps required to reach v from the source. Since the behaviour of \mathbf{M} is not guaranteed syntactically, any vertex that attests to deviation in the correct behaviour of \mathbf{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 Yegorov [56] 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 . Therefore, the same reduction basically yields an EOML instance.

Lemma 3. *SINK-OF-VERIFIABLE-LINE is Karp-reducible to END-OF-METERED-LINE [56].*

Proof. The construction of the successor and predecessor circuits for the EOML instance $\text{EOML} = (S, P, M)$ 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 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 $\text{END-OF-METERED-LINE}$ instance (S, P, 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 optimal pebbling strategy. \square

2.4 Relaxing the SVL Problem

The definition of the SVL problem that we saw in Definition 5 (§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.

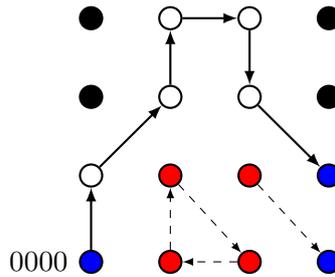


Figure 2.2: Sample $\text{RELAXED-SINK-OF-VERIFIABLE-LINE}$ instance with 16 vertices (i.e., $m = 4$) with a standard path of length $L = 6$. The output of the successor circuit is indicated by the directed edges: i.e., an edge (u, v) implies that $S(v) = u$. The source (resp., sink) vertices are in 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. Length $L \in \{1, \dots, 2^m\}$
2. A successor circuit $\mathbf{S} : \{0, 1\}^m \rightarrow \{0, 1\}^m$
3. A verifier circuit $\mathbf{V} : \{0, 1\}^m \times \{1, \dots, L\} \rightarrow \{0, 1\}$

Promise. For every $v \in \{0, 1\}^m$ and $i \in \{1, \dots, L\}$: $\mathbf{V}(v, i) = 1$ if $v = \mathbf{S}^i(0^m)$.

Solution. One of the following:

- (i) **The sink:** a vertex $v \in \{0, 1\}^m$ such that $\mathbf{V}(v, L) = 1$; or
- (ii) **False positive:** a pair $(v, i) \in \{0, 1\}^m \times \{0, \dots, 2^m\}$ such that $v \neq \mathbf{S}^i(0^m)$ and $\mathbf{V}(v, i) = 1$.

The main difference from Definition 5 is that the promise about the behaviour of the verifier circuit \mathbf{V} is relaxed so that it can also accept vertices off the standard path: the promise is relaxed from an *iff* to an *if*. However, any vertex off the main line accepted by \mathbf{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 $\text{RSVL} = (\mathbf{S}, \mathbf{V}, L)$ be an RSVL instance. Let's consider the END-OF-METERED-LINE instance $\text{EOML} = (\mathbf{S}, \mathbf{P}, \mathbf{M})$ 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 $L^{\log_2 3}$ 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 $v_L = \text{RSVL}.\mathbf{S}^L(0^m)$. 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 \mathbf{v} is labeled by a tuple of $\ell = \log_2(L)$ pairs of the form $(v, i) \in \{0, 1\}^m \times \{1, \dots, L\}$.

2. Every vertex \mathbf{v} that is not a self-loop (i.e., with $\text{EOML.S}(\mathbf{v}) \neq \mathbf{v}$ or $\text{EOML.P}(\mathbf{v}) \neq \mathbf{v}$) contains in its label only pairs $(v, i) \in \{0, 1\}^m \times \{1, \dots, L\}$ such that $\text{RSVL.V}(v, i) = 1$ and is given a non-zero value by EOML.M .
3. A vertex \mathbf{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 $(v, i) \in \{0, 1\}^m \times \{1, \dots, L\}$ such that $v = \text{RSVL.S}^i(0^m)$.

Items 1 and 2 are immediate from the description of the reduction. Item 3 follows since any vertex \mathbf{v} which is not a self-loop corresponds to a valid pebbling configuration. Thus, if the label of \mathbf{v} contains only pairs $(v, i) \in \{0, 1\}^m \times \{1, \dots, L\}$ such that $v = \text{RSVL.S}^i(0^m)$ then the successor (resp. predecessor) of \mathbf{v} on the main directed EOML line is the successive (resp. preceding) pebbling configuration. Note that the converse implication is straightforward.

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

Otherwise, we show that the solution \mathbf{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, \mathbf{v} lies off the main standard path and, by Items 2 and 3 above, its label must contain a pair (v, i) such that $\text{RSVL.V}(v, i) = 1$ but $v \neq \text{RSVL.S}^i(0^m)$, i.e., a false positive. Since there are $\log_2(L)$ 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). \square

3 PPAD-Hardness from 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. 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 the interactive protocol for the *decision* version of IS given in [78], 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 [84] introduced the notion of time-lock puzzles. Such a puzzle is specified by a sampling algorithm `sample` 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 **poly**(λ) parallelism cannot compute the solution much faster than the honest (sequential) algorithm. They also propose a simple and elegant construction, described below.

On input (λ, T) , a puzzle is sampled by choosing two random $\lambda/2$ -bit primes p, q , which define a λ -bit modulus $N = pq$, together with any $x \in \mathbb{Z}_N^*$.¹ The puzzle and

¹Strictly speaking, 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 solve the construction using the best known attacks. Therefore, we should really be describing our constructions in terms of two parameters:

solution are then defined as

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

Let's call this problem ITERATED-SQUARING (IS). The solution to IS be efficiently computed by the puzzle sampling algorithm in two steps using the knowledge of the group order $\phi(N) = (p-1)(q-1)$ as

$$e = 2^T \bmod \phi(N), \quad S = x^e \bmod N. \quad (3.1)$$

It is conjectured in [84] that the fastest way to solve IS (without the knowledge of the group order) is through repeated squaring:

$$x \rightarrow x^2 \rightarrow x^{2^2} \rightarrow x^{2^3} \rightarrow \dots \rightarrow x^{2^T} \pmod{N}. \quad (3.2)$$

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 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. For a security parameter $\lambda \in \mathbb{N}$, let N be the product of two random $\lambda/2$ -bit primes and $x \in \mathbb{Z}_N^*$ be sampled uniformly at random. For any $T \in \mathbb{N}$ and any algorithm A (whose running time is significantly smaller than what is required to factor N) that, on input (N, x, T) , outputs $y = x^{2^T} \bmod N$ with overwhelming probability, requires *sequential* time not much less than what is required to compute T sequential squarings in \mathbb{Z}_N^* [84].

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. First, we require the primes p, q that define the modulus N to be *safe* primes (where, recall that, p is safe if $(p-1)/2$ is also prime). Second, the group that we assume our hardness is the group of signed quadratic residues (defined below), which carries slightly more structure than \mathbb{Z}_N^* – this extra structure helps guarantee *unambiguous soundness* in Pietrzak’s protocol (cf. §3.2.2 for the details). However, we justify in Remarks 2 and 4 below why these changes do not really affect the strength of the assumption.

a (proper) security parameter λ and the bit length $\lambda_{\text{RSA}} \in \mathbf{poly}(\lambda)$ of the modulus N that corresponds to λ -bits of security. For instance, for 128-bit security the size of the modulus should be around 2048 bits. However, in the chapter we will use λ_{RSA} as the security parameter λ to avoid cluttering. Moreover, since λ and λ_{RSA} are polynomially-related it suffices for our purposes as we work with asymptotic security (rather than concrete security).

Signed quadratic residues. For two safe primes p and q , and $N := p \cdot q$ the signed quadratic residues [42; 54] 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

$$\left\{ -\frac{N-1}{2}, \dots, \frac{N-1}{2} \right\}. \quad (3.3)$$

Since $-1 \in \mathbb{Z}_N^*$ is a quadratic non-residue with Jacobi symbol $+1$, the function $|\cdot|$ acts as an (efficiently-computable) isomorphism² 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 $+1$.

3.1.2.1 The Assumption

The hardness assumption that underlies the RSVL instance proposed in this chapter is stated below. It pertains to the ITERATED-SQUARING problem in the setting of signed quadratic residues, which we denote by ITERATED-SQUARING⁺ (IS⁺) to avoid conflation.

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

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

It is conjectured that for some constant c , there are $c \cdot 2^\lambda / \lambda^2$ safe λ -bit primes (cf. [92]), so a random λ -bit prime is safe with probability $\approx c/\lambda$. Under the weaker requirement that there are at least $2^\lambda / \mathbf{poly}(\lambda)$ λ -bit primes for some polynomial in **poly**(λ), Assumption 1 is at least as strong as an assumption where we additionally require p and q to be safe, since if a $\Theta(1/\mathbf{poly}(\lambda))$ fraction of all λ -bit primes is safe, an N sampled as in Assumption 1 will be the product of two safe primes with noticeable probability $\Theta(1/\mathbf{poly}(\lambda)^2)$.

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

Remark 3 (Using other groups and operations). One can prove soundness of the protocol also when a standard RSA modulus is used (i.e., p and q are just random primes), or in fact any other group, but then one needs a computational assumption to argue soundness of the protocol, namely, that it is hard to find elements of small order [15; 93]. So the hardness of our RSVL instance would rely on hardness of computing x^{2^T} as in Assumption 2, and additionally on the hardness of finding some element z where $z^e = x$ for some e of polynomial size.

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

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

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

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

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

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

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

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

Before moving on the proof system, we point out some properties of the set of generators of the quadratic residues. This will prove useful later in establishing hardness of the RSVL instance proposed in §3.3 (Claim 1.2 in Theorem 1).

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

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

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

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

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

$$\Pr_{x \leftarrow QR_N^+} [x \in QR_N^*] = 1 - \frac{p' + q' - 1}{p' \cdot q'} \geq 1 - \frac{1}{2^{\lambda/2-2}} . \quad (3.4)$$

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

Secondly, since the squaring function is an automorphism³ of QR_N^+ (and also QR_N) $x \in QR_N^*$ implies $x^2 \in QR_N^*$. As a result, starting with any $x \in QR_N^*$, repeated squaring generates a subset of QR_N^* : i.e., for any $x \in QR_N^*$ we have

$$\left\{ x, x^2, x^{2^2}, x^{2^3}, \dots, x^{2^{(p'-1)(q'-1)-1}} = x \right\} \subseteq QR_N^* . \quad (3.5)$$

3.2 Pietrzak's Proof System

The key component of our construction is Pietrzak's interactive protocol [78] for the decision version of 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 so-called “verifiable delay function” [15]. The protocol we use in this work differs in some minor aspects from the one in [78]. 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 [83], 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, which is deterministic, 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 ℓ -round

³Note 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 originally shown by Blum [14] — it is an automorphism.

interactive protocol, in each round $i \in [\ell]$, 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 $(I, (\beta_1, \dots, \beta_\ell), (\alpha_1, \dots, \alpha_\ell))$. 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, $(\ell \cdot (b + a) \cdot \log_2(|\Sigma|))$.

Interactive proofs (IPs). The classical notion of an interactive proof for a language L is due to Goldwasser, Micali and Rackoff [52].

Definition 17. An interactive protocol (P, V) is a δ -*sound* interactive proof (IP) for L if:

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

Unambiguous IPs. Reingold, Rothblum and Rothblum [83] 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 $I \notin L$) 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 $\tilde{\mathsf{P}}$ is some arbitrary (cheating) strategy, we say that $\tilde{\mathsf{P}}$ *deviates* from the protocol at round i^* if the message sent by $\tilde{\mathsf{P}}$ 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 $\beta_1, \dots, \beta_{i^*-1}$ in rounds $1, \dots, i^* - 1$ respectively, we say that $\tilde{\mathsf{P}}$ deviates from the protocol at round i^* if

$$\tilde{\mathsf{P}}(I, i^*, (\beta_1, \dots, \beta_{i^*-1})) \neq \mathsf{P}(I, i^*, (\beta_1, \dots, \beta_{i^*-1})).$$

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

Definition 18. An interactive protocol (P, V) , in which we call P the *prescribed prover*, is a (δ, ϵ) -unambiguously sound IP for L if the following three properties hold:

- **Prescribed Completeness:** For every $I \in \{0, 1\}^*$, if V interacts with P on common input I , then V outputs $L(I)$ with probability 1.
- **Soundness:** For every $I \notin L$ and every (computationally-unbounded) cheating prover strategy $\tilde{\mathsf{P}}$, the verifier V accepts when interacting with $\tilde{\mathsf{P}}$ with probability less than $\delta(|I|)$, where $\delta = \delta(n)$ is called the *soundness error* of the proof system.

- **Unambiguity:** For every $I \in \mathbf{L}$, every (computationally-unbounded) cheating prover strategy $\tilde{\mathbf{P}}$, every round $i^* \in [\ell]$, and for every $\beta_1, \dots, \beta_{i^*-1}$, if $\tilde{\mathbf{P}}$ first deviates from the protocol in round i^* (given the messages $\beta_1, \dots, \beta_{i^*-1}$ in rounds $1, \dots, i^* - 1$ respectively), then at the end of the protocol \mathbf{V} accepts with probability at most $\epsilon(|I|)$, where the probability is over \mathbf{V} 's coin tosses in rounds i^*, \dots, ℓ .

An IP is simply δ -unambiguously 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 power⁴, we assume that both prover and verifier have access to a common reference string (CRS). When the CRS is simply a uniformly random string, 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 19. A pair of machines (\mathbf{P}, \mathbf{V}) , where \mathbf{P} is the prescribed prover, is a (δ, ϵ) -unambiguously sound *adaptive* non-interactive proof system for a language \mathbf{L} if \mathbf{V} is probabilistic polynomial-time and the following three properties hold:

- **Prescribed Completeness:** For every $I \in \{0, 1\}^*$,

$$\Pr_R [\mathbf{V}(I, \mathbf{P}(I, R), R) = \mathbf{L}(I)] = 1.$$

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

$$\Pr_R \left[\mathbf{V}(I, \tilde{\pi}, R) = 1 \mid (I, \tilde{\pi}) \leftarrow \tilde{\mathbf{P}}(R), I \notin \mathbf{L} \right] \leq \delta(|I|).$$

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

$$\Pr_R \left[\mathbf{V}(I, \tilde{\pi}, R) = 1 \mid (I, \tilde{\pi}) \leftarrow \tilde{\mathbf{P}}(R), \pi \leftarrow \mathbf{P}(I, R), \tilde{\pi} \neq \pi, I \in \mathbf{L} \right] \leq \epsilon(|I|).$$

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

Remark 6. A non-interactive proof system is called an *argument* if the soundness and unambiguity properties hold only against computationally-bounded (i.e., polynomial in the size of the instance) cheating prover strategy $\tilde{\mathbf{P}}$.

⁴It was shown in [73] that a non-interactive protocol without a CRS exists only for trivial languages, i.e. those in **BPP**.

3.2.2 The Interactive Protocol

We describe here the interactive protocol from [78] for the decision version of IS^+ , i.e., it allows a prover to convince a verifier that a tuple $I = (N, x, T = 2^t, y)$ satisfies the relation $y = x^{2^T}$ in the group QR_N^+ (i.e., here $x^2 := x \circ x$) using $\log T$ rounds of interaction. In the first step, the prover sends the midpoint $\mu = x^{2^{T/2}}$ as a commitment to the verifier. Note that if $x^{2^{T/2}} = \mu$ and $\mu^{2^{T/2}} = y$ are both true, so is the original claim $y = x^{2^T}$.

At this point we have reduced the task to prove a statement for time parameter T to proving two statements $I_1 := (N, x, T/2, \mu)$ and $I_2 := (N, \mu, T/2, y)$ for time parameter $T/2$. Next, the verifier merges these two statements into a single statement $I' := (N, x', T/2, y')$ by computing a random linear combination: using the challenge r it computes $x' := x^r \circ \mu$ and $y' := \mu^r \circ y$.

One can show that with overwhelming probability over the choice of r the following holds: if the original statement was wrong, i.e., $x^{2^T} \neq y$, so will the new one, i.e., $x'^{2^{T/2}} \neq y'$. This constitutes the halving sub-protocol: see Figure 1.9. This sub-protocol is now repeated $\log T$ times, halving the time parameter every time, until we arrive at a claim for $T = 1$ at which point the verifier can efficiently check correctness without the help of the prover by making one squaring.

3.2.2.1 Removing Interaction

The interactive proof system just outlined is public-coin and has an exponentially-small soundness error, which means we could make it non-interactive using the Fiat-Shamir methodology [40]. However, this is known to be sound in the random-oracle model for *constant-round* interactive protocols, unlike the proof system above which involves $O(\log T)$ rounds of interaction.⁵ Nonetheless, it was shown in [78] that a close analogue of the Fiat-Shamir Transform does yield a non-interactive protocol that is sound in the random-oracle model. In particular, to remove interaction, we derive verifier’s message (i.e., the r ’s) in each round by applying a hash function $H : \{0, 1\}^* \rightarrow \{0, 1\}^{3\lambda}$ to the prover’s message for that round — the range of H (thought of as $\mathbb{Z}_{2^{3\lambda}}$) is chosen to be sufficiently large so that $H(\cdot) \bmod p'q'$, when H is modelled as a random oracle, is extremely close to uniform over $\mathbb{Z}_{p'q'}$. As a result, we get a non-interactive proof system (also with exponentially-small soundness error) for the statement “ (N, x, T, y) satisfies $y = x^{2^T}$ in QR_N^+ ”.

⁵In [40] the assumption that the transform is sound was made for *constant-round* protocols. It was later shown in [79] 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 amplification theorem, the resultant protocol has 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.

A number of recent works have aimed at relaxing the assumptions under which the Fiat-Shamir Transform can be proved sound [61; 59; 26]. In particular, [26] 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 in §4.4.

Remark 7 (Comparison with [78]). 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 \mathbf{S} and \mathbf{V} in our RSVL instance to be of polynomial size.

1. The range of the hash function H is much larger in this work than in [78]. This allows us to argue that $x^{H(\cdot)} \bmod N$ is close to uniform.
2. The prover described above uses the minimum space necessary, even though additional space can significantly improve the efficiency of the computation of the proof.
3. It suffices for us to consider a time parameter of the form $T = 2^t$ — the original protocol is described for arbitrary T .
4. We iterate the protocol until the parameter $T = 2^t$, which is halved in every round, is down to 1, even though it is more efficient to stop at an earlier round. In other words, the *base proof* in our case is of time parameter 1, whereas it was greater than that in [78].

3.2.3 The Non-interactive Protocol

We describe here in full detail the non-interactive protocol $(\mathbf{P}_{\text{PP}}, \mathbf{V}_{\text{PP}})$ that results from the discussion in the previous section. (The subscript indicates that it is a part of Pietrzak’s protocol and is introduced to avoid confusion with the successor and verifier circuits of the RSVL instance we build from it.) This protocol will serve as the basis for our main construction of RELAXED-SINK-OF-VERIFIABLE-LINE in §3.3. Also described is the simple algorithm \mathbf{S}_{PP} that computes the solution (and is identical to the solver of the RSW time-lock puzzle).

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

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

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

Then $\mathsf{P}_{\text{PP}}(N, x, T, y)$ outputs the proof

$$\pi(x \xrightarrow{T} y) = (N, x, T, y, \boldsymbol{\mu}) \in \mathbb{N} \times \mathcal{QR}_N^+ \times \mathbb{N} \times \mathcal{QR}_N^+ \times (\mathcal{QR}_N^+)^t, \quad (3.6)$$

where $\boldsymbol{\mu}$ denotes μ_1, \dots, μ_t .

- *Verifying a proof*, V_{PP} . The verification algorithm V_{PP} on input a purported proof $\tilde{\pi}(x \xrightarrow{T} y) =: (N, x, T, y, \boldsymbol{\mu})$ first checks that x, y and all μ_i are in \mathcal{QR}_N^+ — if this check fails $\mathsf{V}_{\text{PP}}(\tilde{\pi}(x \xrightarrow{T} y))$ outputs 0. Otherwise let $(x_1, y_1) = (x, y)$ and then for $i = 1, \dots, t$ compute:

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

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

Notation. We end our description with a comment on notation that will be used further in this chapter: we reserve

$$\pi(x \xrightarrow{T} y) := \mathsf{P}_{\text{PP}}(N, x, T, x^{2^T})$$

to denote honestly computed proofs for true statements, and $\tilde{\pi}(x \xrightarrow{T} y)$ for any string that parses as a possible proof, i.e., starts with N, x, T, y and is in $\mathbb{N} \times \mathcal{QR}_N^+ \times \mathbb{N} \times \mathcal{QR}_N^+$.

3.2.4 Unambiguous Soundness

The unambiguous soundness of the proof system $(\mathsf{P}_{\text{PP}}, \mathsf{V}_{\text{PP}})$ can be shown in the random-oracle model assuming that an adversary never finds a “bad query” as defined in Definition 20. 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 the proof system has *statistically-unique* proofs.

Definition 20 (Bad query). A query is a tuple (μ, x, y, T) where $\mu, x, y \in QR_N^+$ and $T \in \mathbb{N}$. Let

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

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

- (i) $x' \notin QR_N^*$; 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 $\lambda/2$ -bit safe primes, the following holds: any adversary that makes at most Q queries to the random oracle H will make a bad query with probability at most $3 \cdot Q/2^{\lambda/2-1}$*

The proof of Lemma 5 is a straightforward adaptation of [78, 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. It not only states that it is hard to find proofs for wrong statements, but it is even hard to find any accepting proofs that differ from honestly generated proofs for true statements.

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

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

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

Proof. Let *break* denote the event that an adversary that makes at most Q queries to the random oracle finds a proof $\tilde{\pi}(x \xrightarrow{T} y)$ such that $\mathbf{V}_{\text{PP}}(\tilde{\pi}(x \xrightarrow{T} y)) = 1$ 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:

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

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

3.2.5 Efficiency

Finally, we point out three properties concerning the efficiency of $(\mathbf{P}_{\text{PP}}, \mathbf{V}_{\text{PP}})$. 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 that follows next in §3.3.

Property 1 (Cost of computing solutions and proofs). *The computational cost incurred to compute $x^{2^T} := \mathbf{S}_{\text{PP}}(N, x, T)$ is $T + \mathbf{poly}(\lambda)$ multiplications in (QR_N, \circ) . The computational cost incurred to compute $\pi(x \xrightarrow{T} y) := \mathbf{P}_{\text{PP}}(N, x, T, y)$ is also $T + \mathbf{poly}(\lambda)$ multiplications in (QR_N, \circ) . The space required is $\mathbf{poly}(\lambda)$ in both cases.*

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

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

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

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

$$r := H(y = \mu_1, I, z, 2 \cdot T), \quad x' := x^r \circ y, \quad y' := y^r \circ z \quad (3.7)$$

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

$$\pi(x' \xrightarrow{T} y') = (N, x', T, y', \mu'_1, \dots, \mu'_t) := \mathbf{P}_{\text{PP}}(N, x', T, y').$$

From the proof for the merged statement, we can reconstruct the proof for $\pi(x \xrightarrow{2T} z)$ as

$$(N, 2T, x, z, \mu_1, \mu'_1, \dots, \mu'_t). \quad (3.8)$$

3.3 The Reduction

To construct a hard distribution of RSVL instances, we rely on the hardness of computing x^{2^T} in the group QR_N^+ as stated in Assumption 2. In particular, we aim to construct an efficient successor circuit \mathbf{S} such that applying it iteratively to the initial state (x, \dots) we reach a (final) state (x^{2^T}, \dots) . Meanwhile, every intermediate state can be efficiently certified to lie on the line using the verifier \mathbf{V} — in order to construct such a \mathbf{V} , we intend to use Pietrzak’s proof system 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 [2] alphabets). Σ^t denotes the set of all strings of length t over Σ ; $\Sigma^{\leq t}$ denotes $\cup_{j \in [t]} \Sigma^j$ (with the empty string denoted by ε). For a string $a \in \Sigma^t$ and $j \in [t]$, $a[j]$ refers to the j -th symbol in a . For two strings a and b , ab represents their concatenation.

We address each node of a complete binary tree of depth t using the binary string that naturally encodes its position — i.e., a node at level $l \in [t]$ is encoded by an l -bit string and, e.g., the root is ε , its children 0, 1 and so on. An analogous system is used for the complete trinary tree (cf. Figure 3.2).

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 (N, x, T) is first sampled as in Assumption 2.

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

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

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

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

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

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

$$\tilde{\pi}(x_0 \xrightarrow{i} y) \neq \pi(x_0 \xrightarrow{i} x_i) : \mathbf{V}_P(\tilde{\pi}(x_0 \xrightarrow{i} y)) = 1$$

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 $\mathbf{S}^T(\pi(x_0 \xrightarrow{0} x_0)) = \pi(x_0 \xrightarrow{T} x_T)$ (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 \mathbf{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 $\pi(x_0 \xrightarrow{i} x_i)$.⁶

An attempt assuming efficient merging. Assume it is possible, say using a procedure \mathbf{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 [91]: reduce the computation of a proof for time parameter 2ℓ to the computation of two proofs for time parameter ℓ , and then use \mathbf{M} to merge. The resulting algorithm \mathbf{F} is given in Algorithm 1.

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

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

The description of the successor and verifier circuits for the corresponding RSVL instance can now be obtained by simulating $\mathbf{F}(N, x, T)$ using iterations and 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 \mathbf{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 line are:

$$\begin{aligned} \emptyset \rightarrow \pi(x_0 \xrightarrow{1} x_1) \rightarrow \pi(x_0 \xrightarrow{2} x_2) \rightarrow \pi(x_0 \xrightarrow{2} x_2) \parallel \pi(x_2 \xrightarrow{1} x_3) \rightarrow \pi(x_0 \xrightarrow{4} x_4) \rightarrow \\ \pi(x_0 \xrightarrow{4} x_4) \parallel \pi(x_4 \xrightarrow{1} x_5) \rightarrow \pi(x_0 \xrightarrow{4} x_4) \parallel \pi(x_4 \xrightarrow{2} x_6) \rightarrow \cdots \rightarrow \pi(x \xrightarrow{T} x_T). \end{aligned}$$

⁶As a way around the above problem, instead of assuming that \mathbf{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 \mathbf{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 \mathbf{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).

The verifier \mathbf{V} simply checks if the input label corresponds to a valid sequence of accepting proofs, which can be done efficiently. The successor \mathbf{S} — given that an input label is a 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 $\mathbf{S}^i(\emptyset)$ for some i requires breaking soundness of the underlying proof system.

Our solution using somewhat-efficient merging. As noted in the introduction, the last approach is similar to Valiant’s construction of general-purpose IVC [91]. In particular, it can be considered to be its instantiation for a *specific* computation, that of solving ITERATED-SQUARING. Unfortunately we do not know how to implement efficient merging for Pietrzak’s proof system (without resorting to heavy machinery like in [91]). 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., $\pi(x \xrightarrow{\ell} y), \pi(y \xrightarrow{\ell} z)$ 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 1). 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.(3.7). Thus, the computation of a proof for time parameter 2ℓ is reduced to the

computation of 3 proofs for time parameter ℓ (unlike 2 proofs for the case merging is efficient), which then can be reduced to computing $3 \cdot 3$ proofs of time parameter $\ell/2$, and so on and so forth until $\ell = 1$ at which point we can efficiently compute the *base proof*.

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

$F(N, x, \ell)$	
1: if $x \notin QR_N^+$ or ℓ is not a power of 2 then	
2: return \perp	▷ Invalid input
3: end if	
4: if $\ell = 1$ then	▷ Base case
5: return $\pi(x \xrightarrow{1} x^2) := P_P(N, x, 1, x^2)$	
6: else	
7: $\pi(x \xrightarrow{\ell/2} y) := F(N, x, \ell/2)$	▷ First recursive call
8: $\pi(y \xrightarrow{\ell/2} z) := F(N, y, \ell/2)$	▷ Second recursive call
9: $\mu := y, r := H(\mu, x, z, \ell/2)$ and $x' := x^r \circ \mu$	▷ Reduce $\pi(x \xrightarrow{\ell} z)$ to $\pi(x' \xrightarrow{\ell/2} y')$
10: $\pi(x' \xrightarrow{\ell/2} y') := F(N, x', \ell/2)$	▷ Third recursive call
11: Parse $\pi(x' \xrightarrow{\ell/2} y')$ as $(\ell/2, x', y', \mu'_1, \dots, \mu'_{\log(\ell/2)})$	
12: return $\pi(x \xrightarrow{\ell} z) := (\ell, x, z, \mu, \mu'_1, \dots, \mu'_{\log(\ell/2)})$	▷ Reconstruct $\pi(x \xrightarrow{\ell} z)$
13: end if	

Algorithm 2: Recursive description of the RSVL instance.

Unwinding the recursion. We obtain the description of successor and verifier circuits for the RSVL instance by simulating F using stack traces. To this end, we view the execution of $F(N, x, T)$ as a complete *ternary* tree τ of depth $t = \log T$, where each node represents a call to F . In particular, a node $i \in [2]^{\leq t}$ 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.(3.7), and so on until the leaves which are labelled using base proofs. For example, the tree corresponding to $F(N, x, 4)$ is depicted in Figure 3.1.(b).

The RSVL instance we propose consists of a standard path of length $3^{\log T}$ 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 $i \in [2]^t$, the i -th vertex on the standard path consists of set the of proofs in the stack of $F(N, x, T)$ when its execution begins the recursion at i — *the stack trace at i* , for short.

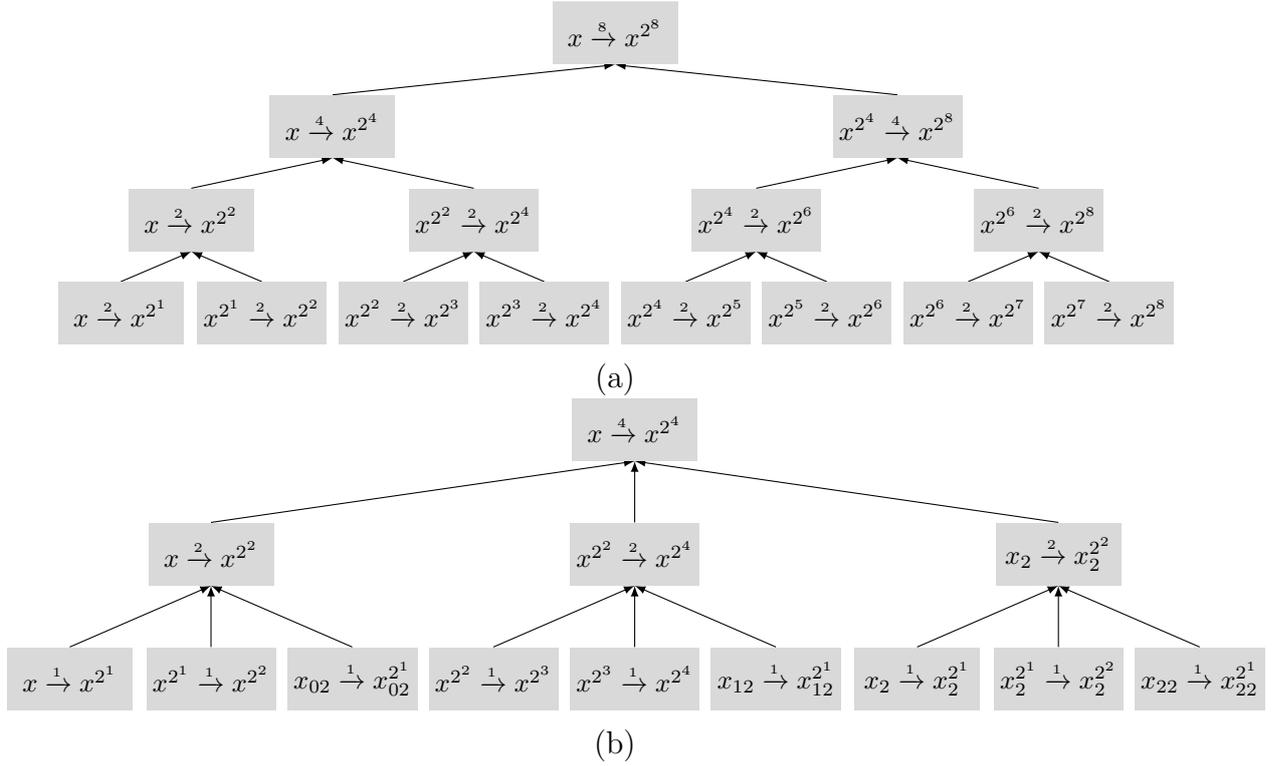


Figure 3.1: (a) The complete binary tree that corresponds to $F(N, x, 8)$ from Algorithm 1. (b) The complete ternary tree that corresponds to $F(N, x, 4)$ from Algorithm 2. The value of the x_2, x_{02}, x_{12} and x_{22} can be computed using eq.(3.7).

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

A quick inspection of Algorithm 2 (and Figure 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 $\mathbf{Trace}(i)$, the standard path in our RSVL instance is defined as

$$\emptyset = (\pi_j)_{j \in \mathbf{Trace}(0^t)} \rightarrow (\pi_j)_{j \in \mathbf{Trace}(0^{t-1})} \rightarrow (\pi_j)_{j \in \mathbf{Trace}(0^{t-2})} \rightarrow (\pi_j)_{j \in \mathbf{Trace}(10^{t-1})} \rightarrow \dots \rightarrow (\pi_j)_{j \in \mathbf{Trace}(2^{t-1})} \rightarrow (\pi_j)_{j \in \mathbf{Trace}(2^t)} \rightarrow \pi(x \xrightarrow{T} x_T).$$

Consequently, the label for a vertex consists of at most $2 \log T$ proofs of the underlying proof system (i.e., eq.(3.6)), 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 (red square), the path to the root is dashed in red, and thus its inclusive ancestors are $(122, 12, 1, \varepsilon)$. The $\mathbf{Trace}(122) = (0, 10, 11, 120, 121)$ is hatched north east in red, and

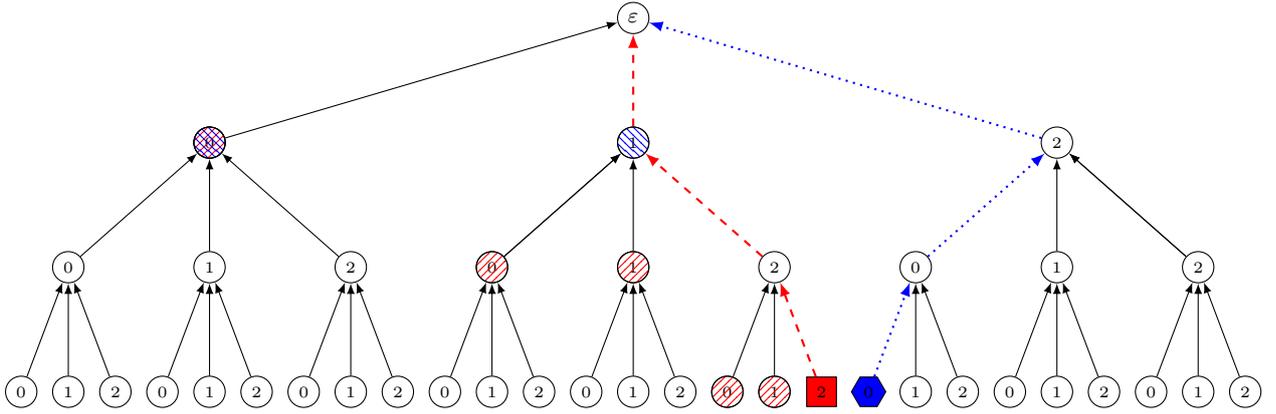


Figure 3.2: A schematic diagram of τ for $F(N, x, 8)$. The resulting RSVL instance is of length 27. The red square node denotes the vertex $i = 122$, whereas the blue hexagon node denotes $i + 1 = 200$. The path from $i = 122$ to the root is dashed in red and the vertices in $\mathbf{Trace}(i)$ are hatched north east in red. Similarly, the path from $i + 1 = 200$ to the root is dotted in blue and the vertices in $\mathbf{Trace}(i + 1)$ are hatched north west in blue. (The vertex 0, as it appears in both the traces, is double-hatched.)

the correct label for the 122-th vertex is thus $(\pi_0, \pi_{10}, \pi_{11}, \pi_{120}, \pi_{121})$ where, for example, $\pi_0 = \pi(x_0 \xrightarrow{4} x_4)$ and $\pi_{10} = \pi(x_4 \xrightarrow{2} x_6)$.

With the RSVL line defined as above, the successor and verifier functions follow quite logically. The verifier, given as input a vertex \mathbf{v} and an index i , ensures that \mathbf{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 \mathbf{V} and \mathbf{S} are provided in the next section.

3.3.2.2 The RSVL Instance

The verifier circuit. On input an index i and a vertex \mathbf{v} , parsed as a sequence of proofs $(\tilde{\pi}_j)_{j \in \mathbf{Trace}(i)}$, the verifier \mathbf{V} ensures that \mathbf{v} is a valid stack trace at i by performing a series of checks on the sequence. Recall from eq.(3.6) that each proof $\tilde{\pi}_j$ in the sequence is of the form $(N, \ell_j, x_j, y_j, \boldsymbol{\mu}_j)$, where ℓ_j denotes the time parameter of the proof, x_j and y_j are its start and end points respectively, and $\boldsymbol{\mu}_j$ denotes the midpoints.

The verifier \mathbf{V} first ensures that each $\tilde{\pi}_j$ is valid by invoking the \mathbf{V}_P algorithm of the underlying proof system (from §3.2.3).

Second, \mathbf{V} checks whether the time parameter of each proof matches its level: the correct time parameter of a proof $\tilde{\pi}_j$ is $T/2^{|j|}$, where $|j|$ is the level of the node j in the tree τ (with the root at level 0 and the leaves at level t). As a concrete example, consider the node 122 from Figure 3.2 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., $\ell_{121} = 1$).

Finally, provided that each proof satisfies the first two conditions, \mathbf{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.(3.7). The start of $\tilde{\pi}_j$ in case (ii), however, just coincides with the endpoint of the proof that precedes it in the sequence (and in case $\tilde{\pi}_j$ is the first proof in the sequence, it must start at x).

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

$$\pi_0 \leftrightarrow \pi_{10} \leftrightarrow, \pi_{11} \leftrightarrow \pi_{120} \leftrightarrow \pi_{121}, \quad (3.9)$$

where the ‘ \leftrightarrow ’ 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.(3.7).

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

Algorithm 3: The verifier circuit for our RSVL instance.

The successor circuit. Given as input a vertex \mathbf{v} , the successor circuit \mathbf{S} first uses a function⁷ $index(\cdot)$ to extract the index i that is implicitly embedded in the sequence of proofs in \mathbf{v} . Next, it confirms whether or not \mathbf{v} is the valid i -th vertex on the line, i.e. the stack trace at i , by invoking the verifier \mathbf{V} . In case \mathbf{v} is invalid, \mathbf{S} forms a self-loop at \mathbf{v} ; otherwise, \mathbf{v} is the valid stack trace at i and \mathbf{S} utilises it to compute the stack trace at $i + 1$.

⁷To be precise, the function $index(\cdot)$ on input a sequence of proof \mathbf{v} computes the index i as follows: it counts the number of proofs of length k in \mathbf{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 .

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

For example, let's consider the successor circuit applied to $(\pi_0, \pi_{10}, \pi_{11}, \pi_{120}, \pi_{121})$, the $i = 122$ -th vertex in the RSVL instance given in Figure 3.2. \mathbf{S} first computes π_{122} , and since this concludes the recursive call $\mathbf{F}(N, x^4, 4)$ at the (source) vertex 1, \mathbf{S} reconstructs the corresponding proof π_1 by merging twice: first it merges π_{120} , π_{121} and π_{122} (using eq.(3.8)) to reconstruct π_{12} , and then it merges π_{10} , π_{11} and π_{12} to obtain π_1 . Finally it assembles the next vertex ($i + 1 = 200$) as (π_0, π_1) , and since $\mathbf{Trace}(200) = (0, 1)$ the newly assembled proof indeed is the stack trace at $i + 1$.

However applying the successor again, as no merging is involved, requires simply adding the base proof π_{200} to the input label (π_0, π_1) . That is, the label for the 201-th vertex is $(\pi_0, \pi_1, \pi_{200})$.

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

Algorithm 4: The successor circuit for our RSVL instance.

Concurrent work. In a concurrent and independent work, Ephraim et al. [38] construct objects called “continuous verifiable delay functions” and show how they can be used to construct SVL instances. Their construction from ITERATED-SQUARING is similar to ours except that they use a $k + 1$ -ary tree instead of a ternary tree in our construction. Appropriately setting the parameter k allows them to relax the assumption to the soundness of (i) $\alpha(1)$ -round Fiat-Shamir Transform to construct hard SVL instances, and (ii) constant-round Fiat-Shamir Transform to separate $\mathbf{P} \cap \mathbf{PPAD}$ from \mathbf{NC} . Our construction given can be thought of as their construction with k set to 2.

3.3.3 Analysis

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

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

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

be defined as in Algorithms 3 and 4, where

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

The family of distributions $\{(\mathbf{S}, \mathbf{V}, \emptyset, T)\}_{\lambda \in \mathbb{N}}$ constitutes a family of hard RSVL instances relative to the random oracle H .

Proof. First, in Claim 1.1 we show that the RSVL instance is efficient: i.e., \mathbf{S} and \mathbf{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 $Q = Q(\lambda) \leq \mathbf{poly}(\lambda)$ 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 value x^{2^T} ; or (ii) a false positive i.e., a pair (\mathbf{v}, i) s.t. $\mathbf{V}(\mathbf{v}, i) = 1$ while $\mathbf{S}^i(\emptyset) \neq \mathbf{v}$.

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 1.2 below we will show that the probability of a type (ii) solution is at most $4 \cdot Q / 2^{\lambda/2-1}$. Therefore, the total probability of the adversary breaking the hardness of our RSVL instance is

$$\Pr[\text{type (i)} \vee \text{type (ii)}] = \Pr[\text{type (i)}] + \Pr[\text{type (ii)}] \leq \frac{4 \cdot Q}{2^{\lambda/2-1}} + p(\lambda) \in \mathbf{negl}(\lambda), \quad (3.10)$$

completing the proof. □

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

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 , from eq.(3.6) we infer 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.⁸

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

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

Proof. For the event “bad query” as defined in Definition 20, the probability that an adversary produces a solution of type (ii) is

$$\begin{aligned} \Pr[\text{type (ii)}] &= \Pr[\text{type (ii)} \wedge (\text{bad query} \vee \neg \text{bad query})] \\ &= \Pr[\text{type (ii)} | \text{bad query}] \cdot \Pr[\text{bad query}] + \Pr[\text{type (ii)} \wedge \neg \text{bad query}] \\ &\leq \Pr[\text{type (ii)} | \text{bad query}] \cdot \Pr[\text{bad query}] + 1/2^{\lambda+1} & (3.11) \\ &\leq 3 \cdot Q/2^{\lambda/2-1} + 1/2^{\lambda/2-2} & (3.12) \\ &\leq 4 \cdot Q/2^{\lambda/2-1}. \end{aligned}$$

The upper bound in eq.(3.12) above directly follows Lemma 5, and we argue below that eq.(3.11) is a consequence of Lemma 6. The properties of QR_N^* (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 (\mathbf{v}, i) of type (ii), i.e., $\mathbf{V}(\mathbf{v}, i) = 1$ but $\mathbf{S}^i(\emptyset) \neq \mathbf{v}$, *without* having made a bad query. Since \mathbf{V} accepts, \mathbf{v} is of the form $(\tilde{\pi}_j)_{j \in \mathbf{Trace}(i)}$, and this sequence is guaranteed to be a valid chain starting at x as described in §3.3.2.2. We argue that, provided $x \in QR_N^*$, the adversary *could not* have output the type (ii) solution (\mathbf{v}, i) since such a vertex \mathbf{v} would equal $\mathbf{S}^i(\emptyset)$ and hence lie on the RSVL line leading to a contradiction. Since a random $x \in QR_N^+$ also belongs to QR_N^* with a overwhelming probability of $1 - 1/2^{\lambda/2-2}$ (cf.eq.(3.4)), the upper bound in eq.(3.11) follows.

Provided $x \in QR_N^*$ and that the adversary never makes a bad query, let's see why the type (ii) solution it outputs lies on the RSVL line. Assume that

$$\mathbf{v} =: (\tilde{\pi}_j)_{j \in \mathbf{Trace}(i)} = \{\tilde{\pi}_{j_1}, \tilde{\pi}_{j_2}, \dots, \tilde{\pi}_{j_\ell}\}.$$

⁸This can also be established by analysing Algorithm 2. Let $m(\cdot)$ denote the upper bound on the size of the vertices (in bits). This parameter is governed by the recursion $m(T) \leq 2 \log T + m(T/2)$, with $m(1) \leq 4\lambda$. 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 $m(T) < 8 \log^2 T \cdot \lambda \in \mathbf{poly}(\log T, \lambda)$.

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 honestly generated proof, as a consequence of the soundness of the proof system (Lemma 6), we get $\tilde{\pi}(x \xrightarrow{\ell_1} y_1) = \pi(x \xrightarrow{\ell_1} y_1)$.

Next, there are two possibilities: either $\tilde{\pi}_{j_1} \leftrightarrow \tilde{\pi}_{j_2}$ or $\tilde{\pi}_{j_1} \leftrightarrow\leftrightarrow \tilde{\pi}_{j_2}$ (with \leftrightarrow and $\leftrightarrow\leftrightarrow$ as defined in eq.(3.9)). In the first case, $x_{j_2} = y_1$ and by the property of the generators given in eq.(3.5), we have $y_1 \in QR_N^*$. Since $y_1 \in QR_N^*$, 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 $x_2 \in QR_N^*$ and, by Lemma 6, we get $\tilde{\pi}_{j_2} = \pi_{j_2}$.

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

□

4 PPAD-Hardness and the Fiat-Shamir Methodology

In this chapter, we present the second construction of hard-on-average distribution on the RELAXED-SINK-OF-VERIFIABLE-LINE problem. 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) #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 [67]. We describe this protocol, both its original form and the variations required for our construction, in §4.2.

The construction of the RELAXED-SINK-OF-VERIFIABLE-LINE instance 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. 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 [24], 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. [67], which is an n -round interactive proof for counting 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 Claim 4.1 (§4.1.2) for the case of Sumcheck Protocol unambiguous soundness reduces to plain soundness.

Theorem 2 (informal). *Solving the END-OF-LINE problem is no easier than breaking the (adaptive) soundness of the Fiat-Shamir Transform, when applied to the Sumcheck Protocol.*

We prove this theorem by constructing an efficiently sampleable distribution 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 $\#\mathbf{P}$ complete problem (and thus any problem in $\#\mathbf{P}$). Since breaking the soundness of Fiat-Shamir is reducible to $\#\text{SAT}$ (in fact to SAT) it follows that efficiently solving the above distribution is no easier than breaking Fiat-Shamir. (We note that, like in the previous chapter, all our theorems apply to the class \mathbf{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 [51] demonstrated theoretical barriers towards the instantiation of Fiat-Shamir in the case of certain computationally sound protocols (a.k.a. “arguments”). Secure instantiations for information theoretically sound protocols (i.e. “proofs”), 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 [61; 26; 24; 28; 77]. 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 [78] from the previous chapter and in very recent work of Canetti et al. [24].

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. [24].

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 3. *Relative to a random oracle, finding a Nash equilibrium is no easier than solving #SAT (and in particular no easier than inverting any one-way function).*

FHE-based instantiation. Canetti et al. [24] 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 sum-check protocol. Adapting their results to our setting gives rise to a hard distribution in the class **CLS**.

Theorem 4 (Informal Statement, see Theorem 9). *Assuming that any one of the LWE-based fully homomorphic encryption schemes in the literature (such as [19; 18; 17; 45; 20]) 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 $\text{quasipoly}(\lambda)/2^\lambda$.

To obtain this result, we need a hash function for which the Fiat-Shamir Transform is sound when it is applied to a sum-check protocol for a hard language. Specifically, we consider Sumcheck Protocol counting the number of satisfying assignments for a formula with poly-logarithmically many variables. By the random self-reducibility of #P [66; 50], 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. [24] 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 sum-check protocol we consider. See §4.4 for an exposition on this result and a formal statement of Theorem 4.

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-

¹Specifically, 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 [50].

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

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 [49]. 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, 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 $f : \mathbb{F}^n \rightarrow \mathbb{F}$ 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:

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

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 $\beta = (\beta_1, \dots, \beta_j) \in \mathbb{F}^j$. We refer to these as *prefix sums*, or the sum with prefix β . A sumcheck proof can prove statements of the form:

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

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 [67]. 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 [90]. A standard arithmetization yields an appropriate polynomial f over the field. In what follows, we use the numbers $\{0, 1, \dots, d\}$ 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 $\beta \in \mathbb{F}^j$. The goal of the procedure is computing the value y of the sum with prefix β , 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: \mathbf{S} and \mathbf{V} . The procedure \mathbf{S} performs single steps, receiving as input the current state, and computing the next state. The *completeness* requirement is that applying \mathbf{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 β , 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 $t \in [L]$. 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 \mathbf{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 \mathbf{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 \mathbf{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 $(\mathbf{S}_{n-j}, \mathbf{V}_{n-j})$ 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 $(\mathbf{S}_{n-j+1}, \mathbf{V}_{n-j+1})$ 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 $(\mathbf{S}_0, \mathbf{V}_0)$ for computing sums of size 1 takes a single step, uses **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 [91].

To construct $(\mathbf{S}_{n-j+1}, \mathbf{V}_{n-j+1})$, given a prefix $\beta \in \mathbb{F}^{j-1}$, the naive idea is to use \mathbf{S}_{n-j} to sequentially compute two sums of size 2^{n-j} . We refer to the process of running \mathbf{S}_{n-j} for L steps to compute a prefix sum as a *full execution* of (\mathbf{S}_{n-j}) . In the naive approach, a full execution of \mathbf{S}_{n-j+1} is comprised of two sequential full executions of \mathbf{S}_{n-j} : a first execution for computing the sum for prefix $(\beta, 0) \in \mathbb{F}^j$, and a second execution computing the sum for prefix $(\beta, 1) \in \mathbb{F}^j$. 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 $(\mathbf{S}_{n-j+1}, \mathbf{V}_{n-j+1})$ 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 **poly**(n) time, and π would be of similar length to π_0 and π_1 . This was the approach used in [91], 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 **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 $\{(\beta, \gamma)\}_{\gamma=0}^d$ (sums of size 2^{n-j}), computing a proof π for the count $y = (y_0 + y_1)$ of the sum with prefix β (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 $(\mathbf{S}_{n-j+1}, \mathbf{V}_{n-j+1})$. Given a prefix $\beta \in \mathbb{F}^{j-1}$, a full execution of \mathbf{S}_{n-j+1} runs $(d+1)$ full executions of \mathbf{S}_{n-j} , computing the prefix sums (and proofs) for sums with prefixes $\{(\beta, \gamma)\}_{\gamma=0}^d$ (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 \mathbf{S}_{n-j} to compute a “merged” proof π for the sum with prefix β (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 (β, σ) for a single field element $\sigma \in \mathbb{F}$. Thus, the number of steps for a full execution of \mathbf{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:

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

The protocol proceeds in n rounds. In the first round, the prover sends a univariate polynomial \tilde{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 valuations over the first $(d+1)$ field elements, and the prover sends these valuations $\alpha_1 = \{\alpha_{1,\gamma} = \tilde{g}_1(\gamma)\}_{\gamma=0}^d$ as its message. On receiving these valuations, the verifier interpolates to recover \tilde{g}_1 , and checks that this polynomial is consistent with the prover’s past claims, i.e., that $\tilde{g}_1(0) + \tilde{g}_1(1) = y$ (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 $\tilde{g}_1, \dots, \tilde{g}_i$ and a prefix of field elements β_1, \dots, β_j . In the $(i+1)$ -th round the parties run the same two-message protocol described above to verify the i -th claim:

$$\sum_{z \in \{0,1\}^{n-i}} f(\beta_1, \dots, \beta_j, z_{i+1}, \dots, z_n) = \tilde{g}_i(\beta_i).$$

Note that this i -th claim is about the sum with the prefix $\beta = (\beta_1, \dots, \beta_j)$, 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 $(\beta_1, \dots, \beta_n)$. Soundness of the protocol follows from the fact that if the i -th claim is false, then for any \tilde{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 \tilde{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 $\beta \in \mathbb{F}^j$ for any $j \in \{0, \dots, n\}$. 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 $(\alpha_1, \dots, \alpha_n)$ in a particular execution of the Sumcheck Protocol. The particular execution is specified by computing for each i the verifier's challenge $\beta_i = H(\mathbb{F}, y, f, \alpha_1, \beta_1, \dots, \beta_{i-1}, \alpha_i)$. 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 $(\alpha_1, \beta_1, \dots, \alpha_n, \beta_n)$. 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 $(\mathbb{F}, \tilde{y}, f)$ 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 $\beta \in \mathbb{F}^j$. In fact, if we define the language appropriately, adaptive soundness of this protocol directly implies adaptive *unambiguous* soundness. We elaborate on this below, see Claim 4.1.

Merging proofs by computing a (small) sum. Recall our setting: for a fixed prefix $\beta \in \mathbb{F}^{j-1}$, we have computed the sums with prefixes $\{(\beta, \gamma) \in \mathbb{F}^j\}_{\gamma=0}^d$, 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 β . 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 (β, γ) . Thus, the first message is simply the values $\alpha_1 = \{y_\gamma\}_{\gamma=0}^d$. Once we know the prover's first message α_1 , we can compute the verifier's random challenge σ by applying the Fiat-Shamir hash function to the instance and to α_1 . To complete a transcript for the non-interactive Sumcheck Protocol (and a proof for the sum with prefix β), we now need a proof for the next claim, i.e., a proof that:

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

In particular, all we need is to compute a sum of size 2^{n-j} with prefix (β, σ) , 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 that an interactive proof system is *unambiguously sound* [83] 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 i , 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 (adaptively) unambiguously sound if there is a prescribed proof for every YES instance, and no PPTM cheating prover can come up with a pair $(x, \tilde{\pi})$ that is accepted by the verifier unless x is a YES instance and $\tilde{\pi}$ is its prescribed proof.

The Sumcheck Protocol is known to be unambiguously sound [83]. 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, see Lemma 7. 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. See Claim 4.1.

We show that adaptive soundness of the non-interactive sum-check protocol applied to a particular language L_{SC} implies adaptive unambiguous soundness for the same language. Moreover, the protocol's adaptive unambiguous soundness for L_{SC} suffices for (unambiguous) soundness of our incrementally-verifiable computation scheme. We begin by defining L_{SC} . The language is defined over tuples that include an instance to the Sumcheck Protocol, a fixed prefix $\beta_1, \dots, \beta_j \in \mathbb{F}$, and a fixed partial transcript of the Sumcheck Protocol. A tuple is of the form

$$(\mathbb{F}, y, f, \beta_1, \dots, \beta_j, \tilde{\alpha}_{j+1}, \beta_{j+1}, \dots, \tilde{\alpha}_{i+j}),$$

where $i, j \in \{0, \dots, n\}$ and their sum is at most n . The language is defined as follows:

1. When $i = j = 0$, this is simply a standard input for the Sumcheck Protocol, and the tuple is in the language if and only if indeed the sum of f over all 2^n inputs equals y .
2. For $j \geq 1$ and $i = 0$, the tuple is in the language if and only if the sum over all 2^{n-i} assignments with prefix β_1, \dots, β_j equals y .
3. For general j and $i \geq 1$, the tuple is in the language if and only if the final prover message $\tilde{\alpha}_{i+j}$ is consistent with the prescribed (honest) prover's message, given the fixed prefix β_1, \dots, β_j and the verifier's messages $\beta_{i+1}, \dots, \beta_{i+j-1}$ (there is no condition on y or on the prior prover messages, only the last one matters).³

³Here, 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.

With this language in mind, we can view each round of the sumcheck as reducing from a claim that a tuple is in L_{SC} , to a claim that a longer tuple 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.

Claim 4.1. *If the 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 $x \in L_{SC}$, whose prescribed proof is π , and an accepting proof $\tilde{\pi} \neq \pi$. Let

$$x = (\mathbb{F}, y, f, \beta_1, \dots, \beta_j, \alpha_{j+1}, \beta_{j+1}, \dots, \alpha_{i+j}),$$

and $\tilde{\pi} = (\tilde{\alpha}_{i+j+1}, \dots, \tilde{\alpha}_n)$. We can use A to break the adaptive soundness of the same argument system, by picking a random index $\ell \in \{i+j+1, n\}$, and computing the challenges:

$$\left\{ \hat{\beta}_k = H(x, \hat{\beta}_{i+j}, \tilde{\alpha}_{i+j+1}, \hat{\beta}_{i+j+1}, \dots, \tilde{\alpha}_k) \right\}_{k=i+j}^{\ell-1}.$$

The new instance is:

$$x' = \left(x, \hat{\beta}_{i+j}, \tilde{\alpha}_{i+j+1}, \hat{\beta}_{i+j+1}, \dots, \tilde{\alpha}_\ell \right).$$

The proof for this new instance is $\pi' := (\tilde{\alpha}_{\ell+1}, \dots, \tilde{\alpha}_n)$. By construction, if $\tilde{\pi}$ is an accepting proof for x , then also π' will be an accepting proof for x' . We claim that with probability at least $1/n$, however, x' is a NO instance of L_{SC} . To see this, let the prescribed proof be $\pi = (\alpha_{i+j+1}, \dots, \alpha_n)$. Finally, let ℓ^* be the *smallest* index s.t. $\tilde{\alpha}_{i+\ell^*} \neq \alpha_{i+\ell^*}$ (such a ℓ^* must exist because $\tilde{\pi} \neq \pi$). It follows that the instance

$$x' = \left(\mathbb{F}, y, f, \alpha_1, \beta_1, \dots, \alpha_i, \hat{\beta}_i, \tilde{\alpha}_{i+1}, \dots, \tilde{\alpha}_{i+\ell^*} \right)$$

is a NO instance for L_{SC} . Thus, in the above reduction, when $\ell = \ell^*$ the adversary finds an accepting proof for a NO instance, and this happens with probability at least $1/n$ over the choice of ℓ . \square

4.2 The Sumcheck Protocol

The Sumcheck Protocol was introduced by Lund et al. [67] to show that $\#\mathbf{P}$ is contained in \mathbf{IP} . In this section, we recall the original protocol (§4.2.1) and then describe the non-interactive protocol that is obtained by applying the Fiat-Shamir Transform to the interactive protocol (§4.2.2).

4.2.1 Interactive 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 $f : \mathbb{F}^n \rightarrow \mathbb{F}$ 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

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

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 $\mathbf{poly}(n, d, \log_2(|\mathbb{F}|))$. In Figure 4.1, we review the standard Sumcheck Protocol from [67], denoted by

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

P_{SC} is an interactive Turing machine, and V_{SC} is a probabilistic interactive Turing machine with oracle access to $f : \mathbb{F}^n \rightarrow \mathbb{F}$. The prover $\mathsf{P}_{\text{SC}}(y, f)$ runs in time $\mathbf{poly}(|\mathbb{F}|^n)$.⁴ The verifier $\mathsf{V}_{\text{SC}}^g(y)$ runs in time $\mathbf{poly}(n, \log_2(|\mathbb{F}|), d)$ and queries the oracle f at a single point. The communication complexity is $\mathbf{poly}(n, \log_2(|\mathbb{F}|), d)$, and the total number of bits sent from the verifier to the prover is $O(n \cdot \log_2 |\mathbb{F}|)$. 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.

Sumcheck Protocol for L_{SC} . Recall the definition of the language L_{SC} from §4.1.2. Although the variant of Sumcheck Protocol described above works for “plain” L_{SC} — i.e., L_{SC} without a prefix and partial transcript and hence $i = j = 0$, — it can be easily adapted for “full” L_{SC} where i and j can be both greater than zero, which is required for our application.⁵ As a first step, we describe in Figure 4.2 the protocol for “prefixed” L_{SC} — i.e., L_{SC} with $j > 0$ but $i = 0$. We show in Theorem 5 that this protocol is an unambiguously-sound interactive proof system (for prefixed L_{SC}). The sketch of the protocol and the corresponding theorem for full L_{SC} then follows Theorem 5. We remark that in both the protocols the verifier is given the polynomial f , unlike oracle-access as in the original Sumcheck Protocol in Figure 4.1. Thus, the verifier’s run-time can be $\mathbf{poly}(n, d, \log_2(|\mathbb{F}|), |f|)$, where $|f|$ refers to the size of the polynomial f under some appropriate representation (e.g., arithmetic circuits).

Theorem 5. *Let $f : \mathbb{F}^n \rightarrow \mathbb{F}$ be an n -variate polynomial of degree at most $d < |\mathbb{F}|$ in each variable. Sumcheck Protocol, described in Figure 4.2, is a $(d(n - j)/|\mathbb{F}|)$ -unambiguously sound interactive proof system for prefixed L_{SC} (i.e., with $j > 0$ and $i = 0$). That is, it satisfies the following three properties.*

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

⁵We remark that the language L_{SC} in its most general form (i.e., with $i, j > 0$) is used only in Claim 4.1. In the discussions that follow, the prefixed language suffices.

Interactive Sumcheck Protocol for $\sum_{z_1, \dots, z_n \in \mathbb{H}} f(z_1, \dots, z_n) = y$

Parameters:

1. \mathbb{F} (field), n (dimension), d (individual degree)
2. $\mathbb{H} \subset \mathbb{F}$

Protocol:

1. Set $y_0 = y$
2. For $i \leftarrow 1, \dots, n$: (at the beginning of round i , both P_{SC} and V_{SC} know y_{i-1} and $\beta_1, \dots, \beta_{i-1} \in \mathbb{F}$)
 - (a) P_{SC} computes the degree- d univariate polynomial

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

P_{SC} sends this polynomial to V_{SC} by specifying its values on the first $d + 1$ field elements, i.e. by sending $\alpha_i = \{\alpha_{i,\gamma} = g_i(\gamma)\}_{\gamma=0}^d$.

- (b) V_{SC} receives $d + 1$ field elements $\tilde{\alpha}_i = \{\tilde{\alpha}_{i,\gamma}\}_{\gamma=0}^d$, and interpolates the (unique) degree- d polynomial \tilde{g}_i s.t. $\forall \gamma \in \{0, \dots, d\}, \tilde{g}_i(\gamma) = \tilde{\alpha}_{i,\gamma}$
 V_{SC} then checks that:

$$\sum_{x \in \mathbb{H}} \tilde{g}_i(x) = y_{i-1}.$$

If not, then V_{SC} rejects.

- (c) V_{SC} chooses a random element $\beta_i \in_R \mathbb{F}$, $y_i = \tilde{g}_i(\beta_i)$, and sends β_i to P_{SC} .
3. V_{SC} uses a single oracle call to f to check that $y_n = f(\beta_1, \dots, \beta_n)$.

Figure 4.1: Sumcheck Protocol ($\mathsf{P}_{\text{SC}}(y, f), \mathsf{V}_{\text{SC}}^f(y)$) from [67].

- **Prescribed Completeness:** For every $y \in \mathbb{F}$ and $\beta \in \mathbb{F}^j$,

$$\Pr[(\mathsf{P}_{\text{SC}}(y, f, \beta), \mathsf{V}_{\text{SC}}(y, f, \beta)) = \mathsf{L}_{\text{SC}}(y, f, \beta, \emptyset)] = 1.$$

- **Soundness:** If $\sum_{z \in \mathbb{H}^{n-j}} f(\beta, z) \neq y$ then for every (computationally unbounded) interactive Turing machine $\tilde{\mathsf{P}}$,

$$\Pr\left[\left(\tilde{\mathsf{P}}(y, f, \beta), \mathsf{V}_{\text{SC}}(y, f, \beta)\right) = 1\right] \leq \frac{d(n-j)}{|\mathbb{F}|}.$$

- **Unambiguity:** If $\sum_{z \in \mathbb{H}^{n-j}} f(\beta, z) = y$ then for every (computationally unbounded) interactive Turing machine $\tilde{\mathsf{P}}$ that deviates from the protocol, V_{SC} accepts with probability at most $d(n-j)/|\mathbb{F}|$.

Interactive β -Prefix Sumcheck Protocol for $\sum_{z_{j+1}, \dots, z_n \in \mathbb{H}} f(\beta, z_{j+1}, \dots, z_n) = y$

Parameters:

1. \mathbb{F} (field), n (dimension), d (individual degree)
2. $\mathbb{H} \subset \mathbb{F}$

Protocol:

1. Let $\beta = (\beta_1, \dots, \beta_j)$ and set $y_j = y$ and
2. For $i \leftarrow j+1, \dots, n$: (at the beginning of round i , both P_{SC} and V_{SC} know y_{i-1} and $\beta_1, \dots, \beta_{i-1} \in \mathbb{F}$)
 - (a) P_{SC} computes the degree- d univariate polynomial

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

P_{SC} sends this polynomial to V_{SC} by specifying its values on the first $d+1$ field elements, i.e. by sending $\alpha_i = \{\alpha_{i,\gamma} = g_i(\gamma)\}_{\gamma=0}^d$.

- (b) V_{SC} receives $d+1$ field elements $\tilde{\alpha}_i = \{\tilde{\alpha}_{i,\gamma}\}_{\gamma=0}^d$, and interpolates the (unique) degree- d polynomial \tilde{g}_i s.t. $\forall \gamma \in \{0, \dots, d\}, \tilde{g}_i(\gamma) = \tilde{\alpha}_{i,\gamma}$. V_{SC} then checks that:

$$\sum_{x \in \mathbb{H}} \tilde{g}_i(x) = y_{i-1}.$$

If not, then V_{SC} rejects.

- (c) V_{SC} chooses a random element $\beta_i \in_R \mathbb{F}$, sets $y_i = \tilde{g}_i(\beta_i)$, and sends β_i to P_{SC} .
3. V_{SC} checks that $y_n = f(\beta_1, \dots, \beta_n)$.

Figure 4.2: β -prefix Sumcheck Protocol ($\mathsf{P}_{\text{SC}}(y, f, \beta), \mathsf{V}_{\text{SC}}(y, f, \beta)$)

Proof. We show below that completeness, soundness and unambiguity holds for $j = 0$ — the argument when $j > 0$ follows similarly by fixing the prefix β .

Prescribed completeness follows from the protocol description. As for the soundness, let $f : \mathbb{F}^n \rightarrow \mathbb{F}$ be a polynomial of degree at most d in each variable, such that $\sum_{z \in \mathbb{H}^n} f(z) \neq y$. Assume for the sake of contradiction that there exists a cheating prover $\tilde{\mathsf{P}}$ for which

$$s := \Pr \left[\left(\tilde{\mathsf{P}}(y, f), \mathsf{V}_{\text{SC}}(y, f) \right) = 1 \right] > \frac{dn}{|\mathbb{F}|}.$$

Recall that in the Sumcheck Protocol the prover sends n univariate polynomials $seqg_1 g_n$, and the verifier sends $n-1$ random field elements $\beta_1, \dots, \beta_{n-1} \in \mathbb{F}$. For every $i \in [1, n]$,

let A_i denote the event that

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

Let S denote the event that $(\tilde{\mathbf{P}}(y, f), \mathbf{V}_{\text{SC}}(y, f)) = 1$. Notice that $\Pr[S|A_1 \wedge \dots \wedge A_n] = 0$. We will reach a contradiction by proving that

$$\Pr[S|A_1 \wedge \dots \wedge A_n] \geq s - \frac{dn}{|\mathbb{F}|}.$$

To this end, we prove by (reverse) induction that for every $j \in [1, n]$,

$$\Pr[S|A_j \wedge \dots \wedge A_n] \geq s - \frac{(n-j+1)d}{|\mathbb{F}|}. \quad (4.1)$$

For $j = n$,

$$s = \Pr[S] \leq \Pr[S|\neg(A_n)] + \Pr[S|A_n] \leq d/|\mathbb{F}| + \Pr[S|A_n],$$

where the latter inequality follows by the Schwartz-Zippel lemma, i.e. the fact that every two distinct univariate polynomials of degree $\leq d$ over \mathbb{F} agree in at most d points. Thus,

$$\Pr[S|A_n] \geq s - d/|\mathbb{F}|.$$

Assume that Equation (4.1) holds for j , and we will show that it holds for $j-1$.

$$\begin{aligned} s - \frac{(n-j+1)d}{|\mathbb{F}|} &\leq \Pr[S|A_j \wedge \dots \wedge A_n] \\ &\leq \Pr[S|\neg(A_{j-1}) \wedge A_j \wedge \dots \wedge A_n] + \Pr[S|A_{j-1} \wedge A_j \wedge \dots \wedge A_n] \\ &\leq d/|\mathbb{F}| + \Pr[S|A_{j-1} \wedge \dots \wedge A_n], \end{aligned}$$

which implies that

$$\Pr[S|A_{j-1} \wedge \dots \wedge A_n] \geq s - \frac{(n-(j-1)+1)d}{|\mathbb{F}|},$$

as desired.

Finally, to prove unambiguity, we describe the proof in [83]. Let $f : \mathbb{F}^n \rightarrow \mathbb{F}$ be a polynomial of degree at most d in each variable, such that $\sum_{\mathbf{z} \in \mathbb{H}^n} f(\mathbf{z}) = y$, and let $\tilde{\mathbf{P}}$ be a cheating prover. Consider $i^* \in [n]$ and $\beta_1, \dots, \beta_{i^*-1} \in \mathbb{F}$ be the first round that $\tilde{\mathbf{P}}$ deviates from the prescribed strategy. i.e. $\forall i < i^*, g_i \equiv \tilde{g}_i$ and $g_{i^*} \not\equiv \tilde{g}_{i^*}$, where $g_i \equiv \tilde{g}_i$ denotes that two polynomials g_i and \tilde{g}_i are equivalent. Here g_i is the output of the prescribed prover \mathbf{P}_{SC} on input $\beta_1, \dots, \beta_{i-1}$ and \tilde{g}_i is the output of $\tilde{\mathbf{P}}$ on the same input.

Claim 5.1. *Let $i \in \{i^*, \dots, n-1\}$ and $\beta_{i^*}, \dots, \beta_{i-1} \in \mathbb{F}$ and suppose $g_i \not\equiv \tilde{g}_i$. Then with probability at least $(1 - d/|\mathbb{F}|)$ over the choice of β_i , either \mathbf{V}_{SC} rejects or $g_{i+1} \not\equiv \tilde{g}_{i+1}$.*

Proof. By the Schwartz-Zippel lemma, since g_i and \tilde{g}_i have degree at most d , $g_i(\beta_i) \neq \tilde{g}_i(\beta_i)$ with probability $(1 - d/|\mathbb{F}|)$ over the choice of β_i . Suppose this is the case, and yet $g_{i+1} \equiv \tilde{g}_{i+1}$, then

$$\sum_{x \in \mathbb{H}} \tilde{g}_{i+1}(x) = \sum_{x \in \mathbb{H}} g_{i+1}(x) = g_i(\beta_i) \neq \tilde{g}_i(\beta_i) = y_i.$$

Thus \mathbf{V}_{SC} would reject when it makes the check to see if $\sum_{x \in \mathbb{H}} \tilde{g}_{i+1}(x) = y_i$. \square

Applying union bound over all $i \in \{i^*, \dots, n-1\}$ for the above claim, we have that with probability at least $(1 - (n-1) \cdot d/|\mathbb{F}|)$ over the choice of $\beta_{i^*}, \dots, \beta_{i-1}$ either \mathbf{V}_{SC} rejects or $g_n \not\equiv \tilde{g}_n$. If $g_n \not\equiv \tilde{g}_n$, by the Schwartz-Zippel lemma, with probability $(1 - d/|\mathbb{F}|)$ over the choice of β_n ,

$$\tilde{g}_n(\beta_n) \neq g_n(\beta_n) = f(\beta_1, \dots, \beta_n).$$

Thus when \mathbf{V}_{SC} makes the oracle call to check if $y_n = f(\beta_1, \dots, \beta_n)$, it will reject. Therefore, by a further application of the union bound, with probability at least $(1 - nd/|\mathbb{F}|)$ either \mathbf{V}_{SC} rejects or $y_n = \tilde{g}_n(\beta_n) \neq g_n(\beta_n) = f(\beta_1, \dots, \beta_n)$. \square

The full Sumcheck Protocol. When $i = 0$, the Sumcheck Protocol for full \mathbf{L}_{SC} works exactly as described in Figure 4.2. Since $i > 0$ corresponds to the case where the Sumcheck Protocol has been executed partially to the point where the prover sends the message $\tilde{\alpha}_{i+j}$, the full Sumcheck Protocol just continues the protocol from that point onwards. Hence the first message in the full Sumcheck Protocol is from the verifier to the prover, unlike the normal Sumcheck Protocol. Let's denote this protocol by

$$(\mathbf{P}_{\text{SC}}(y, f, \boldsymbol{\beta}, \boldsymbol{\tau}), \mathbf{V}_{\text{SC}}(y, f, \boldsymbol{\beta}, \boldsymbol{\tau})),$$

where $\boldsymbol{\tau} = \tilde{\alpha}_{j+1}, \beta_{j+1}, \dots, \tilde{\alpha}_{j+i}$ is the fixed partial transcript. In Theorem 6 below, we capture the fact that the protocol just described is an unambiguously sound interactive proof for \mathbf{L}_{SC} — the proof is similar to that of Theorem 5 and is omitted.

Theorem 6. *Let $f : \mathbb{F}^n \rightarrow \mathbb{F}$ be an n -variate polynomial of degree at most $d < |\mathbb{F}|$ in each variable. The Sumcheck Protocol $(\mathbf{P}_{\text{SC}}(y, f, \boldsymbol{\beta}, \boldsymbol{\tau}), \mathbf{V}_{\text{SC}}(y, f, \boldsymbol{\beta}, \boldsymbol{\tau}))$ described above is a $(d(n-i-j)/|\mathbb{F}|)$ -unambiguously sound interactive proof system for \mathbf{L}_{SC} .*

4.2.2 Non-Interactive Sumcheck Protocol

We consider the non-interactive version of the Sumcheck Protocol obtained by applying the Fiat-Shamir Transform to the protocols from Figure 4.2 and Theorem 6. To be exact, the verifier's "challenges" β_i in the non-interactive protocol are obtained by applying a hash function $H : \{0, 1\}^* \rightarrow \mathbb{F}$ to the transcript thus far, which is comprised of the instance and the prover's messages up to that round.

The non-interactive Sumcheck Protocol $(\mathbf{P}_{\text{FS}}(y, f, \boldsymbol{\beta}), \mathbf{V}_{\text{FS}}(y, f, \boldsymbol{\beta}))$ corresponding to Figure 4.2 is given in Figure 4.3. The algorithm \mathbf{P}_{FS} runs in time $\mathbf{poly}(|\mathbb{F}|^n)$. The verification algorithm \mathbf{V}_{FS} runs in time $\mathbf{poly}(n, |f|)$ and space $O(n \cdot \log_2(|\mathbb{F}|))$. The size of the

proof is $O(n \cdot \log_2(|\mathbb{F}|))$. The non-interactive Sumcheck Protocol $(\mathbf{P}_{\text{FS}}(y, f, \boldsymbol{\beta}, \boldsymbol{\tau}), \mathbf{V}_{\text{FS}}(y, f, \boldsymbol{\beta}, \boldsymbol{\tau}))$ corresponding to the protocol from Theorem 6 works similar to the protocol in Figure 4.2 and hence incurs similar costs.

Assumption on Fiat-Shamir. The assumption that underlies our main theorem pertains to the soundness of the collapsed protocol

$$(\mathbf{P}_{\text{FS}}(y, f, \boldsymbol{\beta}, \boldsymbol{\tau}), \mathbf{V}_{\text{FS}}(y, f, \boldsymbol{\beta}, \boldsymbol{\tau})).$$

In particular, we assume that the Fiat-Shamir Transform is adaptively *sound* when applied to the interactive Sumcheck Protocol and, as a result, that the collapsed protocol $(\mathbf{P}_{\text{FS}}(y, f, \boldsymbol{\beta}, \boldsymbol{\tau}), \mathbf{V}_{\text{FS}}(y, f, \boldsymbol{\beta}, \boldsymbol{\tau}))$ is an adaptively sound non-interactive proof system for the language L_{SC} . By Claim 5.1, it follows that the collapsed protocol is adaptively unambiguously sound as required for the application in §4.3. In Lemma 7 below, we show that the assumption holds for the protocol described in Figure 4.3 *with respect to random oracles*. This also holds for the protocol from Theorem 6 as claimed in Lemma 8. In fact we directly show the stronger property of unambiguous soundness.

Assumption 3. The Fiat-Shamir Transform is *unambiguously* sound for the Sumcheck Protocol from Theorem 6. In other words, there exists a family of hash functions \mathcal{H} such that when instantiated with (random) $H : \{0, 1\}^* \rightarrow \mathbb{F}$ from \mathcal{H} , the non-interactive Sumcheck Protocol $(\mathbf{P}_{\text{FS}}(y, f, \boldsymbol{\beta}, \boldsymbol{\tau}), \mathbf{V}_{\text{FS}}(y, f, \boldsymbol{\beta}, \boldsymbol{\tau}))$ from Theorem 6 is (δ, ϵ) -unambiguously sound for the language L_{SC} for some δ and ϵ that are negligible in n .

Lemma 7. *Let $f : \mathbb{F}^n \rightarrow \mathbb{F}$ be an n -variate polynomial of degree at most d in each variable and $\boldsymbol{\beta} = (\beta_1, \dots, \beta_j) \in \mathbb{F}^j$ be any prefix. Let $(\mathbf{P}_{\text{FS}}^H(y, f, \boldsymbol{\beta}), \mathbf{V}_{\text{FS}}^H(y, f, \boldsymbol{\beta}))$ denote the non-interactive Sumcheck Protocol obtained by instantiating the protocol described in Figure 4.3 with a random oracle H . Then $(\mathbf{P}_{\text{FS}}^H, \mathbf{V}_{\text{FS}}^H)$ is a $(Qd/|\mathbb{F}|)$ -unambiguously sound non-interactive proof system for prefixed 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.*

- **Prescribed Completeness:** For every y ,

$$\Pr [\mathbf{V}_{\text{FS}}^H((y, f, \boldsymbol{\beta}), \mathbf{P}_{\text{FS}}^H(y, f, \boldsymbol{\beta})) = L_{\text{SC}}(y, f, \boldsymbol{\beta}, \emptyset)] = 1.$$

- **Soundness:** For every (computationally unbounded) cheating prover strategy $\tilde{\mathbf{P}}$ that makes at most Q queries to the random oracle,

$$\Pr \left[\mathbf{V}_{\text{FS}}^H((y, f, \boldsymbol{\beta}), \tilde{\pi}) = 1 \mid \begin{array}{l} ((y, f, \boldsymbol{\beta}), \tilde{\pi}) \leftarrow \tilde{\mathbf{P}}^H \\ \sum_{z \in \mathbb{H}^{n-j}} f(\boldsymbol{\beta}, z) \neq y \end{array} \right] \leq \frac{Qd}{|\mathbb{F}|}.$$

- **Unambiguity:** For every (computationally unbounded) cheating prover strategy $\tilde{\mathbf{P}}$ that makes at most Q queries to the random oracle,

$$\Pr \left[\mathbf{V}_{\text{FS}}^H((y, f, \boldsymbol{\beta}), \tilde{\pi}) = 1 \mid \begin{array}{l} ((y, f, \boldsymbol{\beta}), \tilde{\pi}) \leftarrow \tilde{\mathbf{P}}^H \\ \pi \leftarrow \mathbf{P}_{\text{FS}}^H(y, f, \boldsymbol{\beta}) \\ \tilde{\pi} \neq \pi, \sum_{z \in \mathbb{H}^{n-j}} f(\boldsymbol{\beta}, z) = y \end{array} \right] \leq \frac{Qd}{|\mathbb{F}|}.$$

Non-Interactive β -Prefix Sumcheck Protocol

Parameters:

1. \mathbb{F} (field), n (dimension), d (individual degree) and $\mathbb{H} \subset \mathbb{F}$
2. hash function $H : \{0, 1\}^* \rightarrow \mathbb{F}$

$P_{\text{FS}}(y, f, \beta) :$

1. Let $\beta = (\beta_1, \dots, \beta_j)$ and set $y_j = y$.
2. For $i \leftarrow j + 1, \dots, n$:
 - (a) Compute the degree- d univariate polynomial

$$g_i(x) := \sum_{z_{i+1}, \dots, z_n \in \mathbb{H}} f(\beta, \beta_{j+1}, \dots, \beta_{i-1}, x, z_{i+1}, \dots, z_n).$$

Let $\alpha_i = \{\alpha_{i,\gamma} = g_i(\gamma)\}_{\gamma=0}^d$ be the values of g_i on the first $d + 1$ field elements.

- (b) Compute

$$\beta_i = H(\mathbb{F}, y, f, \beta, \{\alpha_{j+1,\gamma}\}_{\gamma=0}^d, \beta_{j+1}, \dots, \beta_{i-1}, \{\alpha_{i,\gamma}\}_{\gamma=0}^d)$$

and set $y_i = g_i(\beta_i)$.

3. Output $\pi = \left(\{\alpha_{j+1,\gamma}\}_{\gamma=0}^d, \dots, \{\alpha_{n,\gamma}\}_{\gamma=0}^d \right)$

$V_{\text{FS}}((y, f, \beta), \{\tilde{\alpha}_{j+1,\gamma}\}_{\gamma=0}^d, \dots, \{\tilde{\alpha}_{n,\gamma}\}_{\gamma=0}^d) :$

1. For $\beta = (\beta_1, \dots, \beta_j)$, set $y_j = y$.
2. For $i \leftarrow j + 1, \dots, n$:
 - (a) Use the $d+1$ field elements $\{\tilde{\alpha}_{i,\gamma}\}_{\gamma=0}^d$ to interpolate the (unique) degree- d polynomial \tilde{g}_i s.t. $\forall \gamma \in \{0, \dots, d\}, \tilde{g}_i(\gamma) = \tilde{\alpha}_{i,\gamma}$.
Check that: $\sum_{x \in \mathbb{H}} \tilde{g}_i(x) = y_{i-1}$. If not, then reject.
 - (b) Compute

$$\beta_i = H(\mathbb{F}, y, f, \beta, \{\tilde{\alpha}_{j+1,\gamma}\}_{\gamma=0}^d, \beta_{j+1}, \dots, \beta_{i-1}, \{\tilde{\alpha}_{i,\gamma}\}_{\gamma=0}^d)$$

and set $y_i = \tilde{g}_i(\beta_i)$.

3. If $y_n = f(\beta_1, \dots, \beta_n)$ then accept and otherwise reject.

Figure 4.3: Non-interactive version of the Sumcheck Protocol ($P_{\text{SC}}(y, f, \beta), V_{\text{SC}}(y, f, \beta)$) from Figure 4.2.

Lemma 8. *Let $f : \mathbb{F}^n \rightarrow \mathbb{F}$ be an n -variate polynomial of degree at most d in each variable, $\boldsymbol{\beta} = (\beta_1, \dots, \beta_j) \in \mathbb{F}^j$ be any prefix, and $\boldsymbol{\tau} = \tilde{\boldsymbol{\alpha}}_{j+1}, \beta_{j+1}, \dots, \tilde{\boldsymbol{\alpha}}_{j+i}$ be any fixed partial transcript. Let $(\mathbf{P}_{\text{FS}}^H(y, f, \boldsymbol{\beta}, \boldsymbol{\tau}), \mathbf{V}_{\text{FS}}^H(y, f, \boldsymbol{\beta}, \boldsymbol{\tau}))$ denote the non-interactive Sumcheck Protocol obtained by instantiating the protocol described in Theorem 6 with a random oracle H . Then $(\mathbf{P}_{\text{FS}}^H(y, f, \boldsymbol{\beta}, \boldsymbol{\tau}), \mathbf{V}_{\text{FS}}^H(y, f, \boldsymbol{\beta}, \boldsymbol{\tau}))$ 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.*

Proof (of Lemma 7). A query is a tuple of the form

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

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

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

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

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

is *bad* if

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

Since β_{ℓ} and $\tilde{\beta}_{\ell}$ are outputs of a random oracle and the polynomials $g_{\ell}(x)$, $\tilde{g}_{\ell}(x)$ 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 $\neg\text{bad}$, 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:

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

4.3 The Reduction

In this section, we present an RSVL instance constructed using the non-interactive Sumcheck Protocol (P_{FS}, V_{FS}) for the language L_{SC} from §4.2.2 as a building block. The proposed RSVL 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 main line in the RSVL instance starts at a fixed initial state s_0 (the source) and ends at a final state s_L (the sink) comprised of the sum

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

as well as a proof π of y 's correctness. The i -th (intermediate) state along the path from s_0 to s_L , which we denote by s_i , consists of appropriately-chosen prefix sums and associated proofs. (To be precise, each state also includes an index $t \in [d+1]^{<n}$ that is determined by its counter i .) The successor S performs single steps, receiving as input the current state s_i , and computing the next state s_{i+1} . The verification procedure V , which takes as input a state s and a counter i and accepts if 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 (s', i) such that $s' \neq s_i$ but V accepts 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 7 (Main Theorem). *For a parameter 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*

$$S := S_{f,H} : \{0,1\}^m \rightarrow \{0,1\}^m \text{ and } V := V_{f,H} : \{0,1\}^m \times [L]$$

be constructed as in Algorithm 7 with

$$m = m(n, d, p) = (d+1)n \log_2(p) \text{ and } L = L(n, d) = \sum_{j \in [n]} (d+2)^j.$$

Given an adversary A that solves instances from the RSVL family

$$\{(S, V, (0^n, f(0^n), \emptyset), L)\}_{n \in \mathbb{N}} \tag{4.2}$$

in polynomial time $T_A = T_A(n)$ and a non-negligible probability $\epsilon = \epsilon(n)$, it is possible to either

- *find the sum $\sum_{z \in \{0,1\}^n} f(z)$ in time $O(T_A)$ 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. The main technical component of our reduction are the successor circuit \mathbf{S} and the verifier circuit \mathbf{V} , described in §4.3.1 and §4.3.2. \mathbf{S} and \mathbf{V} , together, implement the incrementally-verifiable counter for statements of size 2^n . They are defined using a sequence of circuits

$$(\mathbf{S}_n, \mathbf{V}_n), \dots, (\mathbf{S}_0, \mathbf{V}_0),$$

where $(\mathbf{S}_{n-j+1}, \mathbf{V}_{n-j+1})$ is an incrementally-verifiable counter for statements of size 2^{n-j+1} and is implemented *recursively* using $(\mathbf{S}_{n-j}, \mathbf{V}_{n-j})$. At the base of the recursion, $(\mathbf{S}_0, \mathbf{V}_0)$ 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 (\mathbf{S}, \mathbf{V}) simply invoke $(\mathbf{S}_n, \mathbf{V}_n)$.

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 $(\mathbf{S}_{n-j}, \mathbf{V}_{n-j})$ takes L steps, uses m bits of memory, and generates a final proof of size P bits, then $(\mathbf{S}_{n-j+1}, \mathbf{V}_{n-j+1})$ 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 (\mathbf{S}, \mathbf{V}) runs for $2^{O(n)}$ steps, uses $\mathbf{poly}(n)$ space and has proof size of $\mathbf{poly}(n)$. But most importantly \mathbf{S} and \mathbf{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 $(\mathbf{S}_{n-j+1}, \mathbf{V}_{n-j+1})$ in our incrementally-verifiable counting procedure have hardwired 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 $\boldsymbol{\beta} = (\beta_1, \dots, \beta_{j-1}) \in \mathbb{F}^{j-1}$ (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 $(\mathbf{S}_{n-j+1}, \mathbf{V}_{n-j+1})$ is implemented using $(\mathbf{S}_{n-j}, \mathbf{V}_{n-j})$, we need to take a closer look at the non-interactive Sumcheck Protocol given in Figure 4.3. Suppose that the prover \mathbf{P}_{FS} has been invoked on the $\boldsymbol{\beta} = (\beta_1, \dots, \beta_{j-1})$ -prefix sum

$$\sum_{\mathbf{z} \in \{0,1\}^{n-j+1}} f(\boldsymbol{\beta}, \mathbf{z}) = y_{j-1}, \quad (4.3)$$

which is a statement of size 2^{n-j+1} . At the end of the first iteration, \mathbf{P}_{FS} 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

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

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

Now, suppose we are given incrementally-verifiable procedures $(\mathbf{S}_{n-j}, \mathbf{V}_{n-j})$ 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 $(\mathbf{S}_{n-j+1}, \mathbf{V}_{n-j+1})$ 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 $P(n-j) + (d+1) \log_2(p)$ 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 $(\mathbf{S}_n, \mathbf{V}_n)$ is procedure for computing sums of size 2^n with $2^{O(n)}$ steps and **poly**(n) space, and the final proof is of size **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 $\gamma \in [d]$, we sequentially run $(\mathbf{S}_{n-j}, \mathbf{V}_{n-j})$ to compute the valuations $\alpha_{j+1,\gamma}$ with a proof certifying the sum

$$\sum_{\mathbf{z} \in \{0,1\}^{n-j-1}} f(\boldsymbol{\beta}, \gamma, \mathbf{z}) = \alpha_{j+1,\gamma} = g_j(\gamma) \quad (4.4)$$

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

$$\sum_{\mathbf{z} \in \{0,1\}^{n-j-1}} f(\boldsymbol{\beta}, \sigma, \mathbf{z}) = y_j = g_j(\sigma) \quad (4.5)$$

for the next round can also be computed using $(\mathbf{S}_{n-j}, \mathbf{V}_{n-j})$ in an incrementally-verifiable manner. In other words, we have reduced computing the proof for the $\boldsymbol{\beta}$ -prefix sum given in eq.(4.3) to (i) $(d+1)$ *new* sumchecks given in eq.(4.4) concerning the computation of polynomial $g_j(x)$, and (ii) the second iteration of the *original* sumcheck given in eq.(4.5), 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 $(\mathbf{S}_{n-j}, \mathbf{V}_{n-j})$.

The working of the procedure $(\mathbf{S}_{n-j+1}, \mathbf{V}_{n-j+1})$ on an input a prefix $\boldsymbol{\beta} = (\beta_1, \dots, \beta_{j-1})$ 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 $(\mathbf{S}_{n-j}, \mathbf{V}_{n-j})$ 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 $(\mathbf{S}_{n-j}, \mathbf{V}_{n-j})$
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 \mathbf{S}_{n-j+1} and \mathbf{V}_{n-j+1} .

The recursive successor. Recall that $(\mathbf{S}_{n-j+1}, \mathbf{V}_{n-j+1})$, on input a prefix $\beta = (\beta_0, \dots, \beta_{j-1})$, calls the procedure $(\mathbf{S}_{n-j}, \mathbf{V}_{n-j})$ sequentially $d + 2$ times. This results in a sequence of states $s_0, \dots, s_{L(n-j+1)}$, where $s_{L(n-j+1)}$ is comprised of the sum

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

as well as a proof π of y 's correctness. Since the invocations of $(\mathbf{S}_{n-j}, \mathbf{V}_{n-j})$ are sequential, an intermediate state s_i along the path from s_0 to $s_{L(n-j+1)}$, is comprised of at most $(d+2)$ “sub-states”, one for each invocation of $(\mathbf{S}_{n-j}, \mathbf{V}_{n-j})$.

In more details, each state s_i is associated with an index $t \in [d+1]^{\leq(n-j+1)}$ which is determined by the counter i . Loosely speaking, the index t of the i -th state 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 $t = (t_1, \dots, t_\ell)$ (where $\ell \in [n-j+1]$) then s_i consists of t_1 sub-states, where

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

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

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

The resulting construction of \mathbf{S}_{n-j+1} is formally described in Algorithm 5. There we have also addressed a minor detail (which also applies to \mathbf{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, \mathbf{S}_{n-j+1} receives as an auxiliary argument the protocol transcript that serves as the input to H , denoted by τ . From the description of the non-interactive Sumcheck Protocol in Figure 4.3, τ should contain the following information:

1. the *original* statement $(\mathbb{F}, f, y, \beta)$, left empty if the sum y has not been computed yet, and
2. a *partial* proof π for β -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 \mathbf{P}_{FS}).

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

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

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

1. the prefix β' for s' , which is either (β, γ) for $\gamma \in [d]$, or (β, σ) for a challenge σ , and
2. the index t' for β' , which is either ε in case s' is final for $(\mathbf{S}_{n-j}, \mathbf{V}_{n-j})$, or (t_2, \dots, t_ℓ) otherwise.

Next, it checks the validity of the sub-state s' recursively by invoking \mathbf{V}_{n-j} .

The formal description of \mathbf{V}_{n-j+1} is given in Algorithm 6. Similarly to the successor \mathbf{S}_{n-j+1} , \mathbf{V}_{n-j+1} also receives the transcript as input to ensure that it possesses the necessary information to compute the challenges.

4.3.2 The RSVL Instance

The label of the i -th vertex v_i in the proposed RSVL instance is a tuple (t, s_i) , where $s_i \in \mathbb{F}^*$ is a state and $t \in [d+1]^{\leq n}$ its index determined by the counter i . To be precise, t is the (address of) i -th vertex in the perfect $(d+2)$ -dary tree⁶ that the depth-first search *leaves* (i.e. visits for the final time). To map a counter $i \in [L]$ to an index $t \in [d+1]^{\leq n}$, we use a bijective map $\text{DFS}(\cdot)$ (Note that this makes sense only if $T = |[d+1]^{\leq n}|$), which we show below.). Its inverse is denoted by $\text{DFS}^{-1}(\cdot)$. Thus, the main RSVL line consists of the sequence of labels

$$(0^n, s_1), (0^{n-1}1, s_2), (0^{n-1}, s_3), \dots, (11, s_{L-2}), (1, s_{L-1}), (\varepsilon, s_L).$$

The successor and verifier circuits (\mathbf{S}, \mathbf{V}) for the RSVL instance can now be implemented using $(\mathbf{S}_n, \mathbf{V}_n)$ as shown in Algorithm 7. In short, on input a counter i and a label (t, s) , the verifier circuit \mathbf{V} simply checks if t matches $\text{DFS}(i)$ and then invokes the

⁶A $(d+2)$ -ary tree is called *perfect* if all its interior nodes have $(d+2)$ children and all leaves have the same depth.

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

Output the next state

Base Case $S_0(\beta, \varepsilon, \emptyset, \emptyset)$: Return $(f(\beta), \emptyset)$

Recursion

1. If $t = \varepsilon$ and $s \neq \emptyset$ return s (already in final state: self-loop)
 2. If $t = d+1$ then return $\{y_\gamma\}_{\gamma=0}^d$ appended to π_{d+1} (Case A: merge)
 3. Compute sub-state s' by truncating $\{(y_\gamma, \pi_\gamma)\}_{\gamma=0}^{t_1-1}$ from s
 4. Set $t' := (t_2, \dots, t_\ell)$ as the index of the sub-state s'
 5. If $t = d$ or $t_1 = d+1$: increment/initialise $d+2$ -th sub-state
 - (a) If $\tau = \emptyset$ then initialise with statement $\tau := (\mathbb{F}, y_0 + y_1, f, \beta)$ (Case B)
 - (b) Compute updated transcript τ' by appending $\{y_\gamma\}_{\gamma=0}^d$ to τ (Case C)
 - (c) Compute the challenge $\sigma := H(\tau)$ and append σ to τ
 - (d) Increment/initialise $d+2$ -th sub-state: $s' := S_{n-j}((\beta, \sigma), t', s', \tau)$
 - (e) Append $\{(y_\gamma, \pi_\gamma)\}_{\gamma=0}^d$ back to s' to update s and return it
- Else $t \neq d$ and $t_1 \neq d+1$
- (a) If $t \in [d-1]$: initialise t_1+1 -th sub-state (Case B)
 - i. Return $\{(y_\gamma, \pi_\gamma)\}_{\gamma=0}^{t_1}$ appended to $S_{n-j}((\beta, t_1+1), \varepsilon, \emptyset, \emptyset)$
 - (b) Else: increment t_1 -th sub-state (Case C)
 - i. Return $\{(y_\gamma, \pi_\gamma)\}_{\gamma=0}^{t_1-1}$ appended to $S_{n-j}((\beta, t_1), t', s', \emptyset)$

Algorithm 5: The recursive successor circuit S_{n-j+1} .

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

Output a bit indicating accept (1) or reject (0)

Base Case $V_0(\beta, \varepsilon, (y, \emptyset), \emptyset)$: Accept if $y = f(\beta)$ and reject otherwise

Recursion

1. If $t = \varepsilon$ return the bit $b \leftarrow V_{\text{FS}}((y_\varepsilon, f, \beta), \pi_\varepsilon)$ (final state)
2. For $\gamma \in [t_1 - 1]$: verify all final sub-states
 Reject if $V_{n-j}((\beta, \gamma), \varepsilon, (y_\gamma, \pi_\gamma), \emptyset)$ rejects
3. Compute t_1 -th sub-state s' by truncating $\{(y_\gamma, \pi_\gamma)\}_{\gamma=0}^{t_1-1}$ from s
4. Set $t' := (t_2, \dots, t_\ell)$ as the index of the sub-state s'
5. If $t_1 = d + 1$
 - (a) If $\tau = \emptyset$ then initialise with statement $\tau := (\mathbb{F}, y_0 + y_1, f, \beta)$
 - (b) Compute updated transcript τ' by appending $\{y_\gamma\}_{\gamma=0}^d$ to τ
 - (c) Compute the challenge $\sigma := H(\tau)$ and append σ to τ
 - (d) Reject if $V_{n-j}((\beta, \sigma), t', s', \tau')$ rejects
 Otherwise $t_1 < d + 1$
 - (a) Reject if $V_{n-j}((\beta, t_1), t', s', \emptyset)$ rejects
6. Accept

Algorithm 6: The recursive verifier circuit V_{n-j+1} .

recursive verifier circuit V_n on the state s and the index t . On the other hand, on input a label (t, s) , the successor S first ensures that 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 $|S_{n-j+1}| < |S_{n-j}| + \mathbf{poly}(n)$ with $|S_0| = \mathbf{poly}(n)$, and consequently $|S| = \mathbf{poly}(n)$. 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 V_{FS} .

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 (S_{n-j+1}, 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

$$L = L(n, d) = \sum_{j \in [n]} (d+2)^j = \lceil [d+1]^{\leq n} \rceil.$$

For simplicity, assume that $t \in \mathbb{F}^{\leq n}$. From the description of (S_0, V_0) , it is clear that $m(0) = P(0) = \log_2(p)$ (where p , if you recall, denotes the size of the finite field \mathbb{F}). From the description of (S_{n-j+1}, V_{n-j+1}) , we have $m(n-j+1) \leq m(n-j) + (d+1)\log_2(p)$ and $P(n-j+1) = P(n-j) + (d+1)\log_2(p)$. On solving the recursion, we get $m = m(n) \leq (d+1)n\log_2(p)$ and $P = P(n) \leq (d+1)n\log_2(p)$.

4.3.3 Analysis

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

Theorem 7 (Main Theorem). *For a parameter 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*

$$S := S_{f,H} : \{0, 1\}^m \rightarrow \{0, 1\}^m \text{ and } V := V_{f,H} : \{0, 1\}^m \times [L]$$

be constructed as in Algorithm 7 with

$$m = m(n, d, p) = (d+1)n\log_2(p) \text{ and } L = L(n, d) = \sum_{j \in [n]} (d+2)^j.$$

Given an adversary A that solves instances from the RSVL family

$$\{(S, V, (0^n, f(0^n), \emptyset), L)\}_{n \in \mathbb{N}} \tag{4.2}$$

Verifier circuit $V_{f,H}(v, i)$ **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 $(t, s) \in [d + 1]^{\leq n} \times \mathbb{F}^*$ where t is the index and s the state
2. a counter $i \in [L]$

Output a bit indicating accept (1) or reject (0)

Procedure

1. Reject if $t \neq DFS(i)$
2. Return the bit $b \leftarrow V_n(\varepsilon, t, s, \emptyset)$

Successor circuit $S_{f,H}(v)$

Hardwired see V

Input a label v parsed as $(t, s) \in [d + 1]^{\leq n} \times \mathbb{F}^*$

Output the next label

Procedure

1. Set the index $i := DFS^{-1}(t)$
2. Return v if $V(v, i)$ rejects (self-loop)
3. Return $(DFS(i + 1), S_n(\varepsilon, t, s, \emptyset))$

Algorithm 7: The verifier V and successor S for the RSVL instance.

in polynomial time $T_A = T_A(n)$ and a non-negligible probability $\epsilon = \epsilon(n)$, it is possible to either

- find the sum $\sum_{z \in \{0,1\}^n} f(z)$ in time $O(T_A)$ 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 8. In the original paper [31], 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 [8], there exist hard instances of #SAT (since $\mathbf{P} \neq \#\mathbf{P}$). However, as pointed out in [11], 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 7 with $f = f_\Phi$, and Lemma 7 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. \square

Proof (of Theorem 7). Recall that by Definition 16 the adversary \mathbf{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. $\mathbf{V}(v, i) = 1$ while $\mathbf{S}^i((0^n, f(0^n), \emptyset)) \neq v$.

Finding a type (i) solution is tantamount to solving the #SAT instance f since the sink of the RSVL instance defined in eq.(4.2) is $(\varepsilon, s_L = (y_L, \pi_L))$ 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 $(t, \{(\tilde{y}_1, \tilde{\pi}_1), \dots, (\tilde{y}_\ell, \tilde{\pi}_\ell)\})$ and let

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

be the correctly computed vertex. Also, let $\{\tilde{\beta}_1, \dots, \tilde{\beta}_\ell\}$ and $\{\beta_1, \dots, \beta_\ell\}$ denote the associated prefixes. We first establish that there exists at least one index $k^* \in [1, \ell]$ such that the proof $\tilde{\pi}_{k^*}$ breaks the unambiguous soundness of the non-interactive Sumcheck Protocol.

Assume for contradiction that \mathbf{A} violated neither soundness nor unambiguity in the process of finding the type (ii) solution (v, i) s.t. $\mathbf{V}(v, i) = 1$ but $\mathbf{S}^i((0^n, f(0^n), \emptyset)) \neq v$.

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 $(\tilde{y}_k, \tilde{\pi}_k) = (y_k, \pi_k)$.

When V_n is invoked by V on $(\varepsilon, t, s, \emptyset)$ it recurses until $(\tilde{y}_1, \tilde{\pi}_1)$ is a final sub-state for some (S_{n-j}, V_{n-j}) . At this point the validity of $(\tilde{y}_1, \tilde{\pi}_1)$ is checked using a single call to V_{FS} . Recall that it is assumed that neither soundness nor unambiguity was broken. Since $(\tilde{y}_1, \tilde{\pi}_1)$ passes verification and $\tilde{\beta}_1 = \beta_1$, it is guaranteed that the proofs $(\tilde{y}_1, \tilde{\pi}_1)$ (y_1, π_1) are for the *same* statement. Therefore $(\tilde{y}_1, \tilde{\pi}_1) = (y_1, \pi_1)$ 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 β_k is correctly computed, for which there are two possibilities. If β_k corresponds to a challenge then, since y_1, \dots, y_{k-1} have been validated, β_k is also guaranteed to be computed by hashing the same arguments as in the protocol specification; otherwise, β_k is computed by a simple increment, which the verifier again checks for. Therefore, by the same argument as for $k = 1$, we get $\tilde{\pi}_k = \pi_k$ and $\tilde{y}_k = y_k$.

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

- $V_{FS}((\tilde{y}_{k^*}, f, \tilde{\beta}_{k^*}), \tilde{\pi}_{k^*}) = 1$, and $\sum_{z \in \{0,1\}^{n-j_{k^*}}} f(\tilde{\beta}_{k^*}, z) \neq \tilde{y}_{k^*}$; *or*
- $V_{FS}((\tilde{y}_{k^*}, f, \tilde{\beta}_{k^*}), \tilde{\pi}_{k^*}) = 1$, and $\tilde{\pi}_{k^*} \neq P_{FS}(\tilde{y}_{k^*}, f, \tilde{\beta}_{k^*})$ where P_{FS} 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 $(S, V, (0^n, f(0^n), \emptyset), L)$.
2. Let the output returned by A be of the form $(t, \{(\tilde{y}_1, \tilde{\pi}_1), \dots, (\tilde{y}_\ell, \tilde{\pi}_\ell)\})$.
3. Sample a random index $k^* \xleftarrow{\$} [1, \ell]$.
4. Return $\left((f, \tilde{\beta}_{k^*}, \tilde{y}_{k^*}), \tilde{\pi}_{k^*} \right)$, where $\tilde{\beta}_{k^*}$ is the prefix associated to $(\tilde{y}_{k^*}, \tilde{\pi}_{k^*})$.

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,

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

The first inequality follows from the fact that if \mathbf{A} succeeds, there is at least one index k^* such that $V_{\text{FS}}((\tilde{y}_{k^*}, f, \tilde{\beta}_{k^*}), \tilde{\pi}_{k^*}) = 1$, and $\sum_{z \in \{0,1\}^{n-j_{k^*}}} f(\tilde{\beta}_{k^*}, z) \neq \tilde{y}_{k^*}$. 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) . \square

4.4 Instantiating Fiat-Shamir

There has been exciting recent progress [25; 61; 26; 24; 26; 28; 77] on instantiating the Fiat-Shamir Transform using *correlation-intractable hash functions* (CIHFs) [27]. In this section, we discuss the applicability of these results to our setting. We observe that the results in [24] 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 [24] 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 [77]), 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 [24], with several (important) changes that are needed to obtain results in our setting. (The notation also has been adapted.) We highlight changes in the assumptions and theorem statements. For a comprehensive discussion, we refer the reader to [24].

4.4.1 Collision-Intractable Hash Functions

We begin with definitions, starting with the notion of *correlation-intractability* [27] that we require for our application. A function is *quasi-polynomial* in a parameter λ if it is of the form $\lambda^{\text{polylog}(\lambda)}$. We denote the set of all quasi-polynomial functions in λ by **quasipoly**(λ).

Definition 21. For a relation ensemble $\mathcal{R} = \{\mathcal{R}_\lambda \subseteq \mathcal{X}_\lambda \times \mathcal{Y}_\lambda\}_{\lambda \in \mathbb{N}}$, a hash family $\mathcal{H} = \{H_\lambda : \mathcal{I}_\lambda \times \mathcal{X}_\lambda \rightarrow \mathcal{Y}_\lambda\}_{\lambda \in \mathbb{N}}$ is said to be \mathcal{R} -correlation-intractable against *quasi-polynomial* adversaries if for every quasi-polynomial-size $\mathbf{A} = \{\mathbf{A}_\lambda\}_{\lambda \in \mathbb{N}}$, there exists an ϵ such that

$$\Pr [I \leftarrow \mathcal{I}_\lambda, x \leftarrow \mathbf{A}_\lambda(I) : (x, H_\lambda(I, x)) \in \mathcal{R}_\lambda] \leq \epsilon$$

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 22 (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 9. 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 23. A distribution P *multiplicatively ϵ -approximates* a distribution Q if for all outcomes ω , it holds that

$$\Pr[x \leftarrow P : x = \omega] \geq \epsilon \cdot \Pr[x \leftarrow Q : x = \omega].$$

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

Definition 24 (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 $\{\mathbf{R}_\lambda\}_{\lambda \in \mathbb{N}}$ such that for every $x \in \mathcal{X}_\lambda$, the distribution $\mathbf{R}_\lambda(x)$ 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 $\epsilon \geq 1/\text{poly}(\lambda)$.

For our application, we need relations where the sampling function \mathbf{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. [26]. 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 25. Let $\text{SKE} = \{(\mathcal{K}_\lambda, \mathbf{E}_\lambda, \mathbf{D}_\lambda)\}_{\lambda \in \mathbb{N}}$ 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}_{SKE} is

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

where a key is sampled (from \mathcal{I}) as the random ciphertext $C := \mathbf{E}_\lambda(K, M)$ 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 26. A secret-key encryption scheme $\text{SKE} = \{(\mathcal{K}_\lambda, \mathbf{E}_\lambda, \mathbf{D}_\lambda)\}_{\lambda \in \mathbb{N}}$ with message space \mathcal{M}_λ has *universal ciphertexts* if for any secret key $K \in \mathcal{K}_\lambda$, the distribution $\mathbf{E}_\lambda(K, U_{\mathcal{M}_\lambda})$ multiplicatively $1/\mathbf{poly}(\lambda)$ -approximates the distribution $\mathbf{E}_\lambda(\mathcal{K}_\lambda, U_{\mathcal{M}_\lambda})$, where $U_{\mathcal{M}_\lambda}$ denotes the uniform distribution over \mathcal{M}_λ .

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 27 (KDM-security). Let $\text{FHE} = \{(\mathcal{K}_\lambda, \mathbf{E}_\lambda, \mathbf{D}_\lambda, \mathbf{F}_\lambda)\}_{\lambda \in \mathbb{N}}$ be a secret-key fully-homomorphic *bit-encryption* scheme with message space \mathcal{M}_λ , let $f = \{f_\lambda : \mathcal{K}_\lambda \rightarrow \mathcal{M}_\lambda\}_{\lambda \in \mathbb{N}}$ 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 $\mathbf{A} = \{\mathbf{A}_\lambda\}_{\lambda \in \mathbb{N}}$, there exists a $\delta \in \boldsymbol{\delta}$ such that:

$$\Pr \left[\begin{array}{l} (K, E) \leftarrow \mathcal{K}_\lambda \\ (M_1, \dots, M_\ell) \leftarrow f_\lambda(K) : \mathbf{A}_\lambda(E, C_1, \dots, C_\ell) = K \\ \{C_i \leftarrow \mathbf{E}_\lambda(K, M_i)\}_{i \in [\ell]} \end{array} \right] \leq \delta(\lambda)$$

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 [24]. 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 [82], which we define below.

Definition 28 (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

$$\chi_{\text{sk}} = \{\chi_{\text{sk}}(\lambda)\}_{\lambda \in \mathbb{N}} \quad \text{and} \quad \chi_e = \{\chi_e(\lambda)\}_{\lambda \in \mathbb{N}} \quad (4.6)$$

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

- \mathcal{K}_λ is the distribution $\chi_{\text{sk}}^{n'}$.
- $\mathbf{E}_\lambda : \mathbb{Z}_q^{n'} \times \{0, 1\} \rightarrow \mathbb{Z}_q^{n'} \times \mathbb{Z}_q$ is defined so that $\mathbf{E}_\lambda(\mathbf{s}, m)$ is obtained by sampling a uniformly random vector $\mathbf{a} \leftarrow \mathbb{Z}_q^{n'}$, sampling $e \leftarrow \chi_e(\lambda)$, and outputting

$$\left(\mathbf{a}, \mathbf{s}^t \cdot \mathbf{a} + m \cdot \left\lceil \frac{q}{2} \right\rceil + e \right).$$

- $\mathbf{D}_\lambda : \mathbb{Z}_q^{n'} \times (\mathbb{Z}_q^{n'} \times \mathbb{Z}_q) \rightarrow \{0, 1\}$ is defined so that $\mathbf{D}_\lambda(\mathbf{s}, (\mathbf{a}, b))$ is the bit m for which $b - \mathbf{s}^t \cdot \mathbf{a}$ is closer to $m \cdot \left\lceil \frac{q}{2} \right\rceil$ than to $(1 - m) \cdot \left\lceil \frac{q}{2} \right\rceil$.

A pair $(\mathbf{a}, b) \in \mathbb{Z}_q^{n'} \times \mathbb{Z}_q$ is a *Regev encryption* of $m \in \{0, 1\}$ under $\mathbf{s} \in \mathbb{Z}_q^{n'}$ with B -bounded noise if $b - \mathbf{s}^t \cdot \mathbf{a} - m \cdot \left\lceil \frac{q}{2} \right\rceil$ 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 29 (Regev-Extractable Secret-Key Homomorphic Encryption). A secret-key fully-homomorphic bit-encryption scheme $\{(\mathcal{K}_\lambda, \mathbf{E}_\lambda, \mathbf{D}_\lambda, \mathbf{F}_\lambda)\}_{\lambda \in \mathbb{N}}$ is considered $(n', q, \chi_{\text{sk}})$ -*Regev-extractable* with $B(\lambda)$ -bounded noise if it satisfies the following structural properties.

1. The distribution of \mathbf{s} when sampling $(\mathbf{s}, E) \leftarrow \mathcal{K}_\lambda$ is $\chi_{\text{sk}}^{n'}$ with χ_{sk} as in eq.(4.6).
2. There is a **poly** (λ) -time evaluable extract function $\mathbf{X} = \{\mathbf{X}_\lambda\}_{\lambda \in \mathbb{N}}$ such that
 - (a) For any λ , any $\mathbf{s} \in \chi_{\text{sk}}^{n'}$, and any $m \in \{0, 1\}$, it holds that $\mathbf{X}_\lambda(\mathbf{E}_\lambda(\mathbf{s}, m))$ is a Regev encryption (\mathbf{a}, b) of m under \mathbf{s} with B -bounded noise, and where \mathbf{a} is uniformly random in $\mathbb{Z}_q^{n'}$.
 - (b) For any $m_1, \dots, m_{n'} \in \{0, 1\}$, any circuit $\mathbf{C} : \{0, 1\}^{n'} \rightarrow \{0, 1\}$, and any $(\mathbf{s}, E) \in \mathcal{K}_\lambda$, it holds with probability 1 that

$$\mathbf{X}_\lambda(\mathbf{F}_\lambda(E, \mathbf{C}, \mathbf{E}_\lambda(\mathbf{s}, m_1), \dots, \mathbf{E}_\lambda(\mathbf{s}, m_{n'})))$$

is a Regev encryption (\mathbf{a}, b) of $\mathbf{C}(m_1, \dots, m_{n'})$ under \mathbf{s} with B -bounded noise.

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

Definition 30 (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* (κ, ℓ, δ) -*CCO-secure* if for every ensemble of ℓ -bit messages $\{m_\lambda\}_{\lambda \in \mathbb{N}}$, FHE is f -KDM δ -secure against quasi-polynomial adversaries for the “augmented bit-by-bit circular security function”

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

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 [24; 26] 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 8 ([26]). *Let $\text{SKE} = \{(\mathcal{K}_\lambda, \text{E}_\lambda, \text{D}_\lambda)\}_{\lambda \in \mathbb{N}}$ be a secret-key encryption scheme with universal ciphertexts, message space $\{0, 1\}^\ell$, and key distribution \mathcal{K}_λ equal to the uniform distribution on $\{0, 1\}^\kappa$ for some $\kappa = \kappa(\lambda)$. If SKE is \mathcal{F} -KDM δ -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 $\delta \in \delta$ such that \mathcal{H}_{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 [24]. 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 [24].

Assumption 4 (Existence of quasi-polynomially CCO-secure FHE). For some n', q, χ_{sk} , there exists a quasi-polynomially (κ, ℓ, δ) -CCO secure secret key FHE scheme that is $(n', q, \chi_{\text{sk}})$ -Regev-extractable with B -bounded noise for $\kappa = \lambda^{\Theta(1)}, \ell = \lambda^{\Omega(1)}, B \leq q/\tilde{\Omega}(\lambda)$ and $\chi_{\text{sk}}^{n'}$ 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}^ℓ contains all functions, with output size ℓ , computable in quasi-polynomial time.

Claim 8.1. *If Assumption 4 is true, then there exist parameters $n' = n'(\lambda), q = q(\lambda)$, and $\chi_{\text{sk}} = \chi_{\text{sk}}(\lambda)$ such that for some $\ell = \lambda^{\Omega(1)}$, $\text{Regev}_{n', q, \chi_{\text{sk}}, \chi_e}$ is \mathcal{F}^ℓ -KDM δ -secure, where \mathcal{F}^ℓ 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 $\chi_{\text{sk}}^{n'}$.*

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 [24].

Let \mathbf{A}_{KDM} be the adversary that breaks the KDM security of the Regev encryption scheme. We will use \mathbf{A}_{KDM} to build an adversary \mathbf{A}_{CCO} that breaks Assumption 4. \mathbf{A}_{CCO}

is given as input the challenge $(E, c_1, \dots, c_{\ell+\kappa})$, where $c_{\ell+1}, \dots, c_{\ell+\kappa}$ is the encryption of the secret key K .

On initialization, \mathbf{A}_{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, \mathbf{A}_{CCO} uses the homomorphic evaluation function \mathbf{F} to homomorphically compute $f^*(K)$ as $\mathbf{F}(E, f^*, c_{\ell+1}, \dots, c_{\ell+\kappa})$. But given that \mathcal{F}^ℓ consists of all functions computable in quasi-polynomial time, the above \mathbf{F} computation may take quasi-polynomial time. The rest of the reduction remains unchanged.

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

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

Remark 10. 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 [27]. The first stage of our reduction can be seen as a straightforward adaptation of their argument (as provided in [24]) to our setting. For the first part, we restate the following claim from [24], 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 $i \in \{j+1, \dots, n\}$, an i -th *round partial transcript* consists of the statement $(\mathbb{F}, y, f, \beta)$, and $\tau_i := \alpha_{j+1}, \beta_{j+1}, \dots, \alpha_i, \beta_i$. Recall that $\alpha_i = \{\alpha_{i,\gamma}\}_{\gamma=0}^d$ defines a unique degree d polynomial g_i . Formally the relation \mathcal{R}_{SC} is defined as,

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

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

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

where $\beta_{j+1}, \dots, \beta_i$ and g_i are obtained from τ_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 $(\mathbb{F}, y, f, \boldsymbol{\beta}, \tau_i) \in \mathcal{G}$ (good) if

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

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

Claim 9.1. *Let $\Pi = (\text{P}_{\text{SC}}, \text{V}_{\text{SC}})$ be the $O(\text{polylog}(\lambda))$ -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 $\text{poly}(\lambda)$, then Fiat-Shamir Transform gives an adaptively-sound argument for L_{SC} .*

Proof. This follows in a straightforward manner as in the proof in [24], 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}_λ , that produces a string $(\mathbb{F}, y^*, f^*, \beta_1^*, \dots, \beta_j^*) \notin \text{L}_{\text{SC}}$. i.e

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

and $(\boldsymbol{\alpha}_{j+1}^*, \dots, \boldsymbol{\alpha}_n^*)$ such that V accepts the transcript derived using H . We shall use this cheating prover P^* to create an adversary $\text{A} = \{\text{A}_\lambda\}_{\lambda \in \mathbb{N}}$ 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 $(\mathbb{F}, y^*, f^*, \beta_1^*, \dots, \beta_j^*)$ and $(\boldsymbol{\alpha}_{j+1}^*, \dots, \boldsymbol{\alpha}_n^*)$.
2. Sample a random index $i^* \leftarrow \{j+1, \dots, n-1\}$.
3. Return $(\mathbb{F}, y^*, f^*, \beta_1^*, \dots, \beta_j^*, \tau_{i^*}, \boldsymbol{\alpha}_{i^*+1}^*)$, where $\forall k \in [i]$:

$$\beta_k = h(\mathbb{F}, y^*, f^*, \beta_1^*, \dots, \beta_j^*, \tau_{k-1}, \boldsymbol{\alpha}_k^*).$$

From Sumcheck Protocol, for every accepting transcript for $(\mathbb{F}, y^*, f^*, \beta_1^*, \dots, \beta_j^*) \notin \text{L}_{\text{SC}}$, there must exist at least one round k such that:

$$(\mathbb{F}, y^*, f^*, \beta_1^*, \dots, \beta_j^*, \tau_k) \in \mathcal{B} \quad \text{and} \quad (\mathbb{F}, y^*, f^*, \beta_1^*, \dots, \beta_j^*, \tau_k, \boldsymbol{\alpha}_{k+1}^*, \beta_{k+1}) \in \mathcal{G}.$$

Note that this follows from the fact that $(\mathbb{F}, f^*, y^*, \beta_1^*, \dots, \beta_j^*) \notin \text{L}_{\text{SC}}$, 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. \square

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 8. 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 β) to be sent in round $i + 1$, i.e.

$$g_{i+1}(x) := \sum_{z \in \{0,1\}^{n-i-1}} f(\beta, \beta_{j+1}, \dots, \beta_i, x, z).$$

Claim 9.2. \mathcal{R}_{SC} is a ρ -sparse relation for $\rho = d/|\mathbb{F}|$.

Proof. Given $(\mathbb{F}, y, f, \beta, \tau_i, \alpha)$, we compute the fraction of β such that

$$((\mathbb{F}, y, f, \beta, \tau_i, \alpha), \beta) \in \mathcal{R}_{\text{SC}}.$$

For $(\mathbb{F}, y, f, \beta, \tau_i, \alpha, \beta) \in \mathcal{G}$, we require β to be such that $\tilde{g}(\beta) = g_{i+1}(\beta)$, where $\tilde{g}(x)$ denotes the polynomial described by α , and $g_{i+1}(x)$ is as defined above. This follows from the definition of \mathcal{L}_{SC} . The polynomial $g_{i+1}(x) - \tilde{g}(x)$ has degree at most d (since $g(x)$ has degree at most d), and is non-zero (since \tilde{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 $\tilde{g}(\beta) = g_{i+1}(\beta)$. 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. \square

Claim 9.3. \mathcal{R}_{SC} is sampleable in $\mathbf{quasipoly}(\lambda)$ -time.

Proof. The algorithm for sampling a β given $(\mathbb{F}, y, f, \beta, \tau_i, \alpha)$ works in the following manner. Using the Cantor-Zassenhaus algorithm [29], 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 $g_{i+1}(x) - \tilde{g}(x)$. 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 10, the size of field is larger than any quasi-polynomial, and thus we do not know if we can do this deterministically in quasi-polynomial time. \square

Theorem 8 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 ℓ 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, $(\mathbb{F}, y, f, \beta, \tau_\ell)$, 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 ℓ 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 8, we need an appropriate encryption scheme with universal ciphertexts, and KDM security for all quasi-polynomial computable functions. Specifically, we require an encryption scheme $\text{SKE} = (\text{SKE.G}, \text{SKE.E}, \text{SKE.D})$ with keys of length $\kappa = \kappa(\lambda) \geq \lambda^{\Omega(1)}$ and universal ciphertexts that are δ -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 $[-\frac{q}{4}, \frac{q}{4})$. For the corresponding scheme n' is set to be such that $(2B + 1)^{n'} \in \{0, 1\}^{|\tau|}$, where $|\tau|$ 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 8, and thus invoking the result, the Fiat-Shamir Transform gives us an adaptively-sound argument system. From Claim 4.1, an adaptively-sound argument system is also an adaptively unambiguously-sound argument system, thus completing the proof. \square

5 Conclusions and 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 circuit 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 final Chapter. Our results and techniques motivate several natural research directions.

Hardness relative to random oracles. In [31], 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 8 this claim is wrong. Using further ideas from incrementally-verifiable computing, a recent result [11] 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.

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.

It is well worth pointing out 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 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 [60] to our settings, in particular to the second construction. In [60] 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 [91; 10] 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** [48] 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 [13] in the context of pseudo-random generators (PRG). The PRG in [13] 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.

$$LSB(x^2) \rightarrow LSB(x^{2^2}) \rightarrow LSB(x^{2^3}) \cdots \rightarrow LSB(x^{2^T}) \pmod{N}. \quad (5.1)$$

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 [13]. It is known that for elements $x \in QR_N$ it has a cyclic structure

$$\left\{ x, x^2, x^{2^2}, x^{2^3}, \dots, x^{2^{\pi(x)}} = x \pmod{N} \right\}, \quad (5.2)$$

with the length of the cycle $\pi(x)$ depending on 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 [81]. 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

$$x^{2^i} = x^{2^{i+\pi(x)}} \pmod{N}.$$

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 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 [80] or Brent’s [21] 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 would also be of interest if assumptions related to FACTORING, such as RSA and composite residuosity (CR) assumption [74], can be reduced to EOL (even in the random-oracle model). Since these problems both reduce to FACTORING (see Figure 5.1), they have a richer structure that could be exploited. It is also worth pointing out that RSA is widely believed not to be equivalent to FACTORING [16].

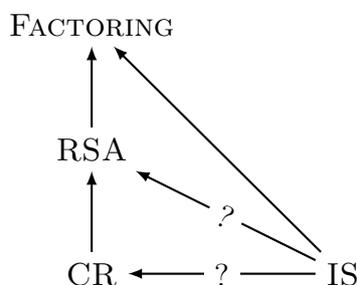


Figure 5.1: Factoring and its related assumptions. $A \rightarrow B$ denotes that problem A reduces to problem B . The arrows with a question mark denote unknown relationships.

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.

One such possibility could be to the computation of Lucas sequences [64], 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 primality 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 $s_0 = 4, s_1, \dots, s_{p-2} \bmod M_p$, 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 primality along with the computation, which would enable other parties to validate such primes more efficiently.

Round-complexity of interactive protocol. A question related to the one preceding is regarding the round complexity of interesting protocols (with statistical soundness). Note that Sumcheck Protocol requires polynomially-many rounds of interaction between the prover and the verifier. Pietrzak's protocol, on the other hand, enables statistical testing of $y = f(N, x, T) = x^{2^T} \bmod N$ in logarithmic number of rounds. Are there other problems that allow for such testing? Interestingly, even the best known interactive protocol for UNSAT \in **co-NP** (which is also the Sumcheck Protocol) requires polynomially-many rounds.

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>. (Cited on pages 14, 18 and 31.)
- [2] BABICHENKO, Y. Query complexity of approximate Nash equilibria. *J. ACM* 63, 4 (2016), 36:1–36:24. (Cited on page 11.)
- [3] BAN, F., JAIN, K., PAPADIMITRIOU, C. H., PSOMAS, C.-A., AND RUBINSTEIN, A. Reductions in ppp. *Information Processing Letters* 145 (2019), 48 – 52. (Cited on page 5.)
- [4] BARAK, B., GOLDREICH, O., IMPAGLIAZZO, R., RUDICH, S., SAHAI, A., VADHAN, S. P., AND YANG, K. On the (im)possibility of obfuscating programs. *J. ACM* 59, 2 (2012), 6:1–6:48. (Cited on page 14.)
- [5] 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. (Cited on pages 4, 5 and 9.)
- [6] 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. (Cited on page 15.)
- [7] BENNETT, C. H. Time/space trade-offs for reversible computation. *SIAM J. Comput.* 18, 4 (1989), 766–776. (Cited on pages 14 and 30.)
- [8] BENNETT, C. H., AND GILL, J. Relative to a random oracle A , $\mathbf{P}^A \neq \mathbf{NP}^A \neq \mathbf{co-NP}^A$ with probability 1. *SIAM Journal on Computing* 10, 1 (1981), 96–113. (Cited on page 88.)
- [9] 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. (Cited on page 24.)

- [10] 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. (Cited on page 99.)
- [11] BITANSKY, N., AND GERICHTER, I. On the cryptographic hardness of local search. In *11th Innovations in Theoretical Computer Science Conference (ITCS 2019)* (2020), Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. (Cited on pages 88 and 99.)
- [12] BITANSKY, N., PANETH, O., AND ROSEN, A. On the cryptographic hardness of finding a Nash equilibrium. In *56th Annual Symposium on Foundations of Computer Science* (Berkeley, CA, USA, Oct. 17–20, 2015), V. Guruswami, Ed., IEEE Computer Society Press, pp. 1480–1498. (Cited on pages 14, 18, 25, 27, 29, 30 and 31.)
- [13] BLUM, L., BLUM, M., AND SHUB, M. A simple unpredictable pseudo-random number generator. *SIAM Journal on Computing* 15, 2 (1986), 364–383. (Cited on page 100.)
- [14] BLUM, M. Coin flipping by telephone. In *Advances in Cryptology – CRYPTO’81* (Santa Barbara, CA, USA, 1981), A. Gersho, Ed., vol. ECE Report 82-04, U.C. Santa Barbara, Dept. of Elec. and Computer Eng., pp. 11–15. (Cited on page 41.)
- [15] BONEH, D., BÜNZ, B., AND FISCH, B. A survey of two verifiable delay functions. Cryptology ePrint Archive, Report 2018/712, 2018. <https://eprint.iacr.org/2018/712>. (Cited on pages 40 and 41.)
- [16] 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. (Cited on page 101.)
- [17] 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. (Cited on page 63.)
- [18] 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. (Cited on page 63.)
- [19] 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. (Cited on page 63.)
- [20] 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. (Cited on page 63.)
- [21] BRENT, R. P. An improved monte carlo factorization algorithm. *BIT Numerical Mathematics* 20, 2 (1980), 176–184. (Cited on page 101.)
- [22] BUHRMAN, H., FORTNOW, L., KOUCKÝ, M., ROGERS, J. D., AND VERESHCHAGIN, N. Does the polynomial hierarchy collapse if onto functions are invertible? *Theory of Computing Systems* 46, 1 (Dec 2008), 143. (Cited on page 13.)
- [23] BURESH-OPPENHEIM, J. On the TFNP complexity of factoring. Unpublished, <http://www.cs.toronto.edu/~bureshop/factor.pdf>, 2006. (Cited on page 13.)
- [24] 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. (Cited on pages 17, 25, 61, 62, 63, 90, 92, 94, 95 and 96.)
- [25] 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. (Cited on page 90.)
- [26] CANETTI, R., CHEN, Y., REYZIN, L., AND ROTHBLUM, R. D. Fiat-Shamir and correlation intractability from strong KDM-secure encryption. In *Advances in Cryptology – EUROCRYPT 2018, Part I* (Tel Aviv, Israel, Apr. 29 – May 3, 2018), J. B. Nielsen and V. Rijmen, Eds., vol. 10820 of *Lecture Notes in Computer Science*, Springer, Heidelberg, Germany, pp. 91–122. (Cited on pages 45, 62, 90, 91 and 94.)
- [27] CANETTI, R., GOLDREICH, O., AND HALEVI, S. The random oracle methodology, revisited. *J. ACM* 51, 4 (July 2004), 557–594. (Cited on pages 90 and 95.)
- [28] CANETTI, R., LOMBARDI, A., AND WICHS, D. Non-interactive zero knowledge and correlation intractability from circular-secure fhe. Cryptology ePrint Archive, Report 2018/1248, 2018. <https://eprint.iacr.org/2018/1248>. (Cited on pages 62 and 90.)
- [29] CANTOR, D. G., AND ZASSENHAUS, H. A new algorithm for factoring polynomials over finite fields. *Mathematics of Computation* 36 (1981), 587–592. (Cited on page 97.)

- [30] CHEN, X., DENG, X., AND TENG, S. Settling the complexity of computing two-player Nash equilibria. *J. ACM* 56, 3 (2009). (Cited on page 7.)
- [31] 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. 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. (Cited on pages 17, 25, 88 and 99.)
- [32] 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>. (Cited on page 25.)
- [33] CHUNG, F., DIACONIS, P., AND GRAHAM, R. Combinatorics for the east model. *Advances in Applied Mathematics* 27, 1 (2001), 192–206. (Cited on page 31.)
- [34] COVER, T. M., AND THOMAS, J. A. *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*. Wiley-Interscience, New York, NY, USA, 2006. (Cited on page 113.)
- [35] DASKALAKIS, C., GOLDBERG, P. W., AND PAPADIMITRIOU, C. H. The complexity of computing a Nash equilibrium. *SIAM J. Comput.* 39, 1 (2009), 195–259. (Cited on page 7.)
- [36] DASKALAKIS, C., AND PAPADIMITRIOU, C. H. Continuous local search. In *22nd Annual ACM-SIAM Symposium on Discrete Algorithms* (San Francisco, CA, USA, Jan. 23–25, 2011), D. Randall, Ed., ACM-SIAM, pp. 790–804. (Cited on pages 7 and 33.)
- [37] DENG, X., EDMONDS, J. R., FENG, Z., LIU, Z., QI, Q., AND XU, Z. Understanding PPA-completeness. In *31st Conference on Computational Complexity (CCC 2016)* (Dagstuhl, Germany, 2016), R. Raz, Ed., vol. 50 of *Leibniz International Proceedings in Informatics (LIPIcs)*, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, pp. 23:1–23:25. (Cited on page 6.)
- [38] EPHRAIM, N., FREITAG, C., KOMARGODSKI, I., AND PASS, R. Continuous verifiable delay functions. Cryptology ePrint Archive, Report 2019/619, 2019. <https://eprint.iacr.org/2019/619>. (Cited on page 57.)
- [39] 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. (Cited on page 7.)

- [40] FIAT, A., AND SHAMIR, A. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology – CRYPTO’86* (Santa Barbara, CA, USA, Aug. 1987), A. M. Odlyzko, Ed., vol. 263 of *Lecture Notes in Computer Science*, Springer, Heidelberg, Germany, pp. 186–194. (Cited on pages 16 and 44.)
- [41] 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. (Cited on page 6.)
- [42] FISCHLIN, R., AND SCHNORR, C.-P. Stronger security proofs for RSA and Rabin bits. *Journal of Cryptology* 13, 2 (Mar. 2000), 221–244. (Cited on page 39.)
- [43] 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. (Cited on page 14.)
- [44] 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. (Cited on page 14.)
- [45] 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. (Cited on page 63.)
- [46] 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. (Cited on page 24.)
- [47] 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. (Cited on page 9.)
- [48] 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. (Cited on pages 7 and 100.)

- [49] 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. (Cited on page 64.)
- [50] 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. (Cited on page 63.)
- [51] 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. (Cited on page 62.)
- [52] GOLDWASSER, S., MICALI, S., AND RACKOFF, C. The knowledge complexity of interactive proof systems. *SIAM J. Comput.* 18, 1 (1989), 186–208. (Cited on page 42.)
- [53] 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. (Cited on page 11.)
- [54] HOFHEINZ, D., AND KILTZ, E. The group of signed quadratic residues and applications. In *Advances in Cryptology – CRYPTO 2009* (Santa Barbara, CA, USA, Aug. 16–20, 2009), S. Halevi, Ed., vol. 5677 of *Lecture Notes in Computer Science*, Springer, Heidelberg, Germany, pp. 637–653. (Cited on page 39.)
- [55] HUBÁČEK, 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. (Cited on page 13.)
- [56] HUBÁČEK, 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. (Cited on pages 7, 14, 17, 25, 27, 33 and 34.)
- [57] JEŘÁBEK, E. Integer factoring and modular square roots. *J. Comput. Syst. Sci.* 82, 2 (2016), 380–394. (Cited on pages 7 and 13.)
- [58] JOHNSON, D. S., PAPADIMITRIOU, C. H., AND YANNAKAKIS, M. How easy is local search? *Journal of Computer and System Sciences* 37, 1 (1988), 79 – 100. (Cited on pages 6 and 13.)

- [59] KALAI, Y. T., KHURANA, D., AND SAHAI, A. Statistical witness indistinguishability (and more) in two messages. In *Advances in Cryptology – EUROCRYPT 2018, Part III* (Tel Aviv, Israel, Apr. 29 – May 3, 2018), J. B. Nielsen and V. Rijmen, Eds., vol. 10822 of *Lecture Notes in Computer Science*, Springer, Heidelberg, Germany, pp. 34–65. (Cited on page 45.)
- [60] 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. (Cited on page 99.)
- [61] KALAI, Y. T., ROTHBLUM, G. N., AND ROTHBLUM, R. D. From obfuscation to the security of Fiat-Shamir for proofs. In *Advances in Cryptology – CRYPTO 2017, Part II* (Santa Barbara, CA, USA, Aug. 20–24, 2017), J. Katz and H. Shacham, Eds., vol. 10402 of *Lecture Notes in Computer Science*, Springer, Heidelberg, Germany, pp. 224–251. (Cited on pages 45, 62 and 90.)
- [62] 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. (Cited on page 13.)
- [63] KOMARGODSKI, I., AND SEGEV, G. From minicrypt to obfustopia via private-key functional encryption. In *Advances in Cryptology – EUROCRYPT 2017, Part I* (Paris, France, Apr. 30 – May 4, 2017), J. Coron and J. B. Nielsen, Eds., vol. 10210 of *Lecture Notes in Computer Science*, Springer, Heidelberg, Germany, pp. 122–151. (Cited on page 15.)
- [64] LEHMER, D. H. An extended theory of lucas’ functions. *Annals of Mathematics* 31, 3 (1930), 419–448. (Cited on page 102.)
- [65] LEMKE, C. E., AND HOWSON, J. T. Equilibrium points of bimatrix games. *SIAM Journal on Applied Mathematics* 12, 2 (1964), 413–423. (Cited on page 11.)
- [66] 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. (Cited on page 63.)
- [67] LUND, C., FORTNOW, L., KARLOFF, H., AND NISAN, N. Algebraic methods for interactive proof systems. *J. ACM* 39, 4 (Oct. 1992), 859–868. (Cited on pages 16, 25, 61, 65, 70, 71 and 72.)
- [68] MAHMOODY, M., AND XIAO, D. On the power of randomized reductions and the checkability of SAT. In *2010 IEEE 25th Annual Conference on Computational Complexity* (June 2010), pp. 64–75. (Cited on page 13.)

- [69] MEGIDDO, N., AND PAPADIMITRIOU, C. H. On total functions, existence theorems and computational complexity. *Theor. Comput. Sci.* 81, 2 (1991), 317–324. (Cited on pages 3, 4, 13 and 27.)
- [70] MICALI, S. Computationally sound proofs. *SIAM Journal on Computing* 30, 4 (2000), 1253–1298. (Cited on page 24.)
- [71] 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. (Cited on page 24.)
- [72] NASH, J. Non-cooperative Games. *The Annals of Mathematics* 54, 2 (1951), 286–295. (Cited on page 3.)
- [73] 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. (Cited on page 43.)
- [74] 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. (Cited on page 101.)
- [75] PAPADIMITRIOU, C. H. On the complexity of the parity argument and other inefficient proofs of existence. *J. Comput. Syst. Sci.* 48, 3 (1994), 498–532. (Cited on pages 4, 7, 9, 11, 12 and 29.)
- [76] PAPADIMITRIOU, C. H. *Computational complexity*. Academic Internet Publ., 2007. (Cited on page 4.)
- [77] 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. (Cited on pages 62 and 90.)
- [78] 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. (Cited on pages 16, 17, 20, 25, 37, 41, 44, 45, 47 and 62.)
- [79] POINTCHEVAL, D., AND STERN, J. Security arguments for digital signatures and blind signatures. *Journal of Cryptology* 13, 3 (June 2000), 361–396. (Cited on page 44.)

- [80] POLLARD, J. M. A monte carlo method for factorization. *BIT Numerical Mathematics* 15, 3 (Sep 1975), 331–334. (Cited on page 101.)
- [81] RABIN, M. O. Digitalized signatures and public-key functions as intractable as factorization. Tech. rep., USA, 1979. (Cited on page 100.)
- [82] 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. (Cited on page 92.)
- [83] 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. (Cited on pages 41, 42, 69 and 74.)
- [84] RIVEST, R. L., SHAMIR, A., AND WAGNER, D. A. Time-lock puzzles and timed-release crypto. Tech. rep., Cambridge, MA, USA, 1996. (Cited on pages 15, 37 and 38.)
- [85] ROSEN, A., SEGEV, G., AND SHAHAF, I. Can PPAD hardness be based on standard cryptographic assumptions? In *TCC 2017: 15th Theory of Cryptography Conference, Part II* (Baltimore, MD, USA, Nov. 12–15, 2017), Y. Kalai and L. Reyzin, Eds., vol. 10678 of *Lecture Notes in Computer Science*, Springer, Heidelberg, Germany, pp. 747–776. (Cited on page 13.)
- [86] 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. (Cited on pages 14 and 30.)
- [87] 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. (Cited on page 11.)
- [88] 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. (Cited on page 5.)
- [89] 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. (Cited on page 2.)
- [90] TOVEY, C. A. A simplified NP-complete satisfiability problem. *Discrete Applied Mathematics* 8, 1 (1984), 85–89. (Cited on page 65.)

- [91] VALIANT, P. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In *TCC 2008: 5th Theory of Cryptography Conference* (San Francisco, CA, USA, Mar. 19–21, 2008), R. Canetti, Ed., vol. 4948 of *Lecture Notes in Computer Science*, Springer, Heidelberg, Germany, pp. 1–18. (Cited on pages 18, 22, 23, 24, 50, 51, 66 and 99.)
- [92] VON ZUR GATHEN, J., AND SHPARLINSKI, I. E. Generating safe primes. *J. Mathematical Cryptology* 7, 4 (2013), 333–365. (Cited on page 39.)
- [93] 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. (Cited on page 40.)

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 $\lambda/2$ -bit safe primes, the following holds: any adversary that makes at most Q queries to the random oracle H will make a bad query with probability at most $3 \cdot Q/2^{\lambda/2-1}$*

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 $x \in QR_N^*$ and moreover either

- (i) $x' \notin QR_N^*$; 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 $y' := \mu^r \circ y$.

In the rest of the proof and in Claims 9.4 and 9.4, to avoid cluttering, we drop the subscript in λ and denote it by λ . Since the range of the range of the random oracle H is $\{0, 1\}^{3\lambda}$ (cf. §3.2.2), its output is distributed $2^{-\lambda}$ -close to uniform over $\mathbb{Z}_{p'q'}$. In the analysis below we assume that it is actually uniform over $\mathbb{Z}_{p'q'}$ and then apply the data processing lemma [34] to get the desired bound. That is, we show that

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

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

$$\Pr_r[(y' = x'^{2^{T/2}}) \wedge (x' \in QR_N^*)] + \Pr_r[x' \notin QR_N^*] \leq 3/2^\lambda. \quad (\text{A.1})$$

We bound the two probabilities separately in Claims 9.4 and 9.5, and the lemma follows by a union bound over all the queries. \square

Claim 9.4. $\Pr_r[x' \notin QR_N^*] \leq 2/2^\lambda$.

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 $|QR_N^+| = p'q'$). As $x, \mu \in QR_N^+$, also $x' = x^r \circ \mu = x^{r+e_\mu}$ is in QR_N^+ ,

and $\langle x' \rangle = QR_N^+$ holds if $\text{ord}(x') = p'q'$, which is the case except if $(r + e_\mu) = 0 \pmod{p'}$ or $(r + e_\mu) = 0 \pmod{q'}$ or equivalently (using that $2^\lambda < \min(p', q')$) if

$$r \in \mathcal{B} := \{\mathbb{Z}_{2^\lambda} \cap \{(-e_\mu \pmod{p'}), (-e_\mu \pmod{q'})\}\} . \quad (\text{A.2})$$

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

Claim 9.5. $\Pr_r[(y' = x'^{2^{T/2}}) \wedge (x' \in QR_N^*)] \leq 1/2^\lambda$.

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 $x' \in QR_N^*$ 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

$$\Pr_r[y' = x'^{2^{T/2}} \wedge x' \in QR_N^*] = \Pr_r[y' = x'^{2^{T/2}} \mid x' \in QR_N^*] \cdot \Pr_r[x' \in QR_N^*] \quad (\text{A.3})$$

For the second factor in eq.(A.3) we have with \mathcal{B} as in eq.(A.2)

$$\Pr_r[x' \in QR_N^*] = \frac{2^\lambda - |\mathcal{B}|}{2^\lambda} . \quad (\text{A.4})$$

Conditioned on $x' \in QR_N^*$ the r is uniform in $\mathbb{Z}_{2^\lambda} \setminus \mathcal{B}$, so the first factor in eq.(A.3) is

$$\Pr_r[y' = x'^{2^{T/2}} \mid x' \in QR_N^*] = \Pr_{r \in \mathbb{Z}_{2^\lambda} \setminus \mathcal{B}}[y' = x'^{2^{T/2}}] . \quad (\text{A.5})$$

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

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

rearranging terms

$$r(e_\mu - 2^{T/2}) + e_y - e_\mu 2^{T/2} = 0 \pmod{p'q'} . \quad (\text{A.6})$$

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

$$e_y - 2^T = 0 \pmod{p'q'}$$

which does not hold as by assumption we have $y \neq x^{2^T}$. So from now on we assume $e_\mu \neq 2^{T/2} \pmod{p'q'}$. Then for $a = e_\mu - 2^{T/2} \neq 0 \pmod{p'q'}$ (and $b = e_y - e_\mu 2^{T/2}$) eq.(A.6) becomes

$$r \cdot a = b \pmod{p'q'}$$

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

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

and the claim follows from the above equation and eq.(A.3)-eq.(A.5) as

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

\square