Title
Efficient Secure Computation and Randomness

Author
Baron, Joshua William

Publication Date
2012

Peer reviewed|Thesis/dissertation

University of California
Los Angeles

Efficient Secure Computation and Randomness

A dissertation submitted in partial satisfaction
of the requirements for the degree
Doctor of Philosophy in Mathematics

by

Joshua William Baron

2012

Abstract of the Dissertation

Efficient Secure Computation and Randomness

by

Joshua William Baron

Doctor of Philosophy in Mathematics

University of California, Los Angeles, 2012

Professor Rafail Ostrovsky, Chair

The question of how to construct optimally efficient secure protocols is a central question in cryptography and in the computer security world at large. We focus in this work on several facets of this question, with a particular view towards the role of randomness in secure computation.

The use of random inputs is ubiquitous in cryptographic primitives. However, the ability to consistently draw from large random sources may be a barrier in practice. We examine the existence of optimally strong *pseudorandom sources* [81, 85] with particularly efficient implementations; namely, we construct exponentially hard pseudorandom generators that can be computed by circuits that have size linear in the generator output.

Conversely, we also examine efficient protocols that rely only minimally on their random inputs. In the *resettable security model* [16], parties may be forced to use the same random input across polynomially many interactions with other parties. We examine this security model in the case of zero knowledge proofs, which are a primitive frequently required in secure computation protocols when one party must prove that they have executed the protocol correctly to another party without revealing secret inputs and compromising security.

Finally, we examine the secure and efficient implementation of a specific functionality, including its various required cryptographic primitives (such as zero knowledge arguments of knowledge). In particular, we construct a protocol that securely realizes *pattern matching*, including single character wildcards and substring matching.

The dissertation of Joshua William Baron is approved.

Don Blasius

Eliezer Gafni

Igor Pak

Rafail Ostrovsky, Committee Chair

University of California, Los Angeles

2012

In memory of my grandmothers

Carole Baron z"l

and

Jennie Shinder z"l

Table of Contents

1 Introduction **1**

1 Overview 1

2 Asymptotically Optimal Pseudorandom Number Generation 2

2.1 Our Results 3

3 Zero Knowledge Protocols with Minimal Reliance on Random Inputs 5

3.1 Our Results 6

4 An Efficient Secure Protocol for Pattern Matching 6

4.1 Our Results 7

5 Outline 8

2 On Linear-Size Pseudorandom Generators and Hardcore Functions . . . **9**

1 Introduction 9

1.1 Our Contribution 11

1.2 Related Work 13

2 Bilinear Uniform-Output Hash Families are Hardcore 16

2.1 Definitions and Preliminaries 16

2.2 BLUO Hash Families are Hardcore 19

2.3 Linear-Size Hardcore Functions 20

3 PRGs Computable By Linear-Sized Circuits 22

3.1 Introduction 22

3.2 Definitions and Lemmas 22

3.3 PRGs for One-Way Functions with Lower-Bounded Min-Entropy . . . 23

3.4 Re-examining Previous Constructions 25

4 PRGs for One-Way Functions with Somewhat Upper-Bounded Min-Entropy 29

5 Candidate one-way functions with high min-entropy 30

5.1 Goldreich's One Way Function With One Predicate 30

5.2 Goldreich's One Way Function With n Predicates 31
6 The specific setting of [19] 34
7 PRGs with Arbitrary Stretch from PRGs with Linear Stretch 35
8 Not All Pairwise Independent Hash Families are Hardcore 36

3 Nearly Simultaneously Resettable Black-Block Zero Knowledge 38
1 Introduction 38
1.1 Overview of Our Contribution 40
1.2 Other Related Work 43
2 Preliminaries, Definitions and Tools 44
2.1 Zero-Knowledge Proof Systems and Resettability 47
3 Black-Box rZK with t-Resettable Soundness 52
3.1 Protocol Π Specification in Detail 54
4 From a rZK Proof System to a New rZK Proof System 60
4.1 Near Compatibility in More Detail 61
5 An Admissible, Near-Compatible rZK Proof System 63
5.1 π_2' Specification in More Detail 64
5.2 High-Level Simulator Strategy in the Proof of Theorem 5.1 66
6 Completeness and t-Resettable Soundness of Π 67
7 Proof of Theorem 3.3 69
8 Proof of Lemma 4.3 80
9 Proof of Lemma 5.3 84
10 Simulator Specification and Proof of Theorem 5.1 91
10.1 Notation and Variables 91
10.2 Simulator $\mathcal{S}$ Specification 92
10.3 Proof of Theorem 5.1 94

4 5PM: Secure Pattern Matching 108
1 Introduction 108

1.1 Our Contributions . . . 109
1.2 Comparison to Previous Work . . . 110
2 Preliminaries . . . 113
2.1 Insecure Pattern Matching (IPM) Algorithm . . . 114
2.2 Preliminary Cryptographic Tools . . . 117
2.3 Computing Linear Operations Using Additively Homomorphic Encryption Schemes . . . 118
3 5PM Protocol . . . 121
3.1 Converting IPM to Linear Operations. . . . 122
3.2 Honest-but-curious (HBC) 5PM Protocol . . . 123
3.3 Malicious Model 5PM Protocol . . . 126
4 Converting Σ protocols to Zero-Knowledge Arguments of Knowledge (ZK-AoKs) 133
4.1 Definitions . . . 133
4.2 Extractable Equivocable Commitment Schemes . . . 135
4.3 Construction of a ZK-AoK from Σ Protocols . . . 137
5 Required Σ protocols . . . 139
6 Detailed π_{5PM}^{M} Specification . . . 142
6.1 Arguments of Knowledge of Consistency . . . 142
6.2 π_{5PM}^{M} Protocol Specification . . . 144
7 Security Analysis . . . 151
7.1 Adversarial Model . . . 152
7.2 Simulator Constructions and Security for π_{5PM}^{H} . . . 154
7.3 Simulator Constructions and Security for π_{5PM}^{M} . . . 158

References . . . **186**

List of Figures

4.1 Example of IPM's operation 116

List of Tables

3.1 Outline of protocol $\Pi = (\pi_0, \pi_1, \pi_2)$. CHCom is statistically binding, while SHCom is statistically hiding. 53

3.2 Outline of nearly resettable coin-flipping subprotocol with output τ. f_*^{PRF} is a pseudorandom function family. (P executes P_L and V executes P_R.) 53

4.1 Comparison of previous protocol functionality, NB= Non-Binary HBC = Honest but Curious, M= Malicious, *=Using unary encoding, **=Can be extended 111

4.2 Detailed comparison with [57] for non-binary substring matching in HBC model with Text length=n, Pattern length=m, Security Parameter=k, Rds= Rounds. 111

4.3 Detailed comparison with [84] and [49] for single character wildcards and substring matching in malicious model with Text length=n, Pattern length=m, Security Parameter=k, Rds= Rounds. 111

4.4 Notation used for 5PM protocols 122

4.5 Overview of 5PM protocol for HBC adversary model, π_{5PM}^H. See Table 4.4 for notation. 124

4.6 Notation used in subprotocols in Tables 4.7 through 4.12 145

4.7 Subprotocol π_{encr} 145

4.8 Subprotocol $\pi_{C,AV}$ 147

4.9 Subprotocol $\pi_{S,AV}$ 147

4.10 Subprotocol π_{vec} 150

4.11 Subprotocol π_{rand} 151

4.12 Subprotocol π_{ans} 151

Acknowledgments

There are many people without whom this thesis would not have been possible.

I would first like to thank my advisor Rafail Ostrovsky. He introduced me to cryptography and to the myriad facets of the academic world, and he did it with endless patience and good cheer. Watching Rafi work has been a privilege and an inspiration, and I thank him for all the time he has spent teaching me and working with me.

I would like to thank Yuval Ishai, who has been there for me from the beginning and onwards. Working with Yuval has been a treat. I have been in constant awe of his endless insights and his deep intuition, and I thank him for the many months (and years) he spent teaching me when I was just starting out.

Working with Ivan Visconti has been such an enjoyable experience. Ivan combines brilliance with a positive, smooth attitude that has taught me much about the cryptographer that I would like to be. Thanks for everything, Ivan.

Karim El Defrawy has taught me a great deal about research. His intelligence combined with his deep work ethic have made him an ideal collaborator; working with him has been a constant learning experience.

I would like to thank all those at HRL Laboratories, LLC, that I have worked with over the years. I would particularly like to thank Roy Matic, who has been a mentor to me since I joined HRL. Special thanks goes to Son Dao, who has taught me a great deal about research and more in the commercial world. Finally, I would like to thank Kirill Minkovich, Aleksey Nogin and Jonathan Lynch, among many others, for the opportunity to work with them.

Without Don Blasius, I would not have been at UCLA. I thank him for bringing another aspiring number theorist to UCLA, even though that is not the field that I ended up in. I would also like to thank Igor Pak and Eliezer Gafni for being on my thesis committee; it is a privilege to have them supervising this thesis.

I would like to thank my parents, Andrew and Ruth, for absolutely everything. In particular, I would like to thank them for instilling in me the deep importance of effort; it

has been at the core of my academic career. I would like to thank my siblings, Zachary and Sarah, for being who they are, and their endless patience with a brother who they must have, only occasionally, found insufferable. My grandfather, Martin Baron, has been and will continue to maintain a standard that I can only hope to match in my life. I love you all.

Finally, I would like to thank my wife, Elizabeth. For her love, for her humor, for her patience, for her help, for saying yes, and for many, many other things, this thesis is dedicated also to her.

Vita

1984	Born, Sacramento, California, USA
Summer 2005 and 2006	Intern, MIT Lincoln Laboratory
2006	Elected to Phi Beta Kappa
2006	B.A. in Mathematics, University of California at Berkeley
2006	Graduated with High Distinction in general scholarship, Highest Honors in the major, UC Berkeley
2006-2009	Teaching Assistant, Mathematics Department, UCLA
2007	M.A. in Mathematics, UCLA
Summer 2007	Intern, NASA Jet Propopulsion Laboratory
2008–2012	Student Researcher, HRL Laboratories, LLC
2009-2010	RTG Fellow in Number Theory, Mathematics Department, UCLA
2010–2012	Teaching Assistant, Mathematics Department, UCLA

Publications

Nearly Simultaneously Resettable Black-Box Zero Knowledge. With Ivan Visconti and Rafail Ostrovsky. Proceedings of the 39th International Colloquium on Automata, Lan-

guages and Programming, LNCS volume 7391, pages 88-99, Springer Berlin / Heidelberg, 2012.

5PM: Secure Pattern Matching. With Karim El Defrawy, Kirill Minkovich, Rafail Ostrovsky, and Eric Tressler. Proceedings of the8th Conference on Security and Cryptography for Networks, LNCS volume 7485, Springer Berlin / Heidelberg, 2012.

CHAPTER 1

Introduction

1 Overview

The question of how to construct optimally efficient secure protocols is a central question in cryptography and in the computer security world at large. Such secure protocols range from pattern matching algorithms, anonymous reputation protocols, transaction authentications, and more generally are conceptualized as secure multiparty computation protocols.

These protocols are constructed by utilizing various primitives, and this thesis is largely an examination of such primitives.

The bulk of primitives in cryptography rely on the use of random inputs. A key reason for such inputs is to keep the strategy of one party unknown to any others. More concretely, encryption schemes that do not require random inputs can fall prey to frequency attacks [31]. In most of the literature, the length of this input is not a matter of concern; however, finding large sources of "pure" randomness (which is to say, having the ability to efficiently select a string uniformly at random from the set $\{0,1\}^{l(n)}$ for arbitrarily large polynomials l) may not be a simple matter. Since cryptographic protocols frequently require security against only computationally bounded adversaries, the question of efficiently constructing sources "random enough" to fool computationally bounded adversaries is important because it would reduce the amount of pure randomness required in many cryptographic applications.

Conversely, another important question in cryptography is the degree to which cryptographic primitives must rely on their random inputs. More specifically, in the resettable security model [16], parties may be forced to use the same random input across polynomially many interactions with other parties. We examine this security model in the case of zero

knowledge proofs, which is a primitive frequently required in secure computation protocols when one party must prove, without revealing secret inputs, that they have executed the protocol correctly to another party.

We finally turn to a very concrete examination of a secure functionality, namely that of secure pattern matching. In this setting, a Client with a pattern p and a Server with a text T attempt to jointly compute discover whether (and where) the pattern matches the text. Various related functionalities are examined, including pattern matching with single character wildcards and substring pattern matching. Security in this context means that the Server to learns nothing about the Client's pattern, and the Client learns nothing about the Server's text other than whether (and where) the pattern matches. In particular, the goal of our work here is to construct the most efficient possible secure protocol that computes all of these functionalities.

We now outline the results of this thesis in more detail.

2 Asymptotically Optimal Pseudorandom Number Generation

A *pseudorandom generator* (PRG) [81, 85] is a deterministic algorithm which stretches a short random seed into a longer output that looks random to any computationally bounded observer. PRGs have numerous applications in cryptography. In particular, they serve as useful building blocks for basic cryptographic tasks such as encryption, commitment, and message authentication.

A seemingly weaker primitive, which satisfies a much milder form of hardness requirement, is a *one-way function* (OWF). A OWF is an efficiently computable function which is hard to invert on a random input. We say that f is *hard to invert* if a polynomial time adversary has only a negligible chance of inverting f. We say that f is *exponentially hard* an exponential time adversary has only an exponentially small chance of inverting f.

A central question in cryptography has be concerned with constructing PRGs from OWFs, which culminated in the seminal result of Håstad, Impagliazzo, Levin and Luby (HILL) [47] that a PRG can be constructed from an *arbitrary* OWF. More recently, there

has been another fruitful line of work on simplifying and improving the efficiency of the HILL construction [51, 43, 44, 45, 46, 42, 83].

The main focus in the above works has been on optimizing efficiency under *minimal assumptions*. The present work is motivated by the following dual question: under which assumptions can we get *optimal efficiency*? Ideally, we would like to obtain a PRG $G : \{0,1\}^n \to \{0,1\}^{l(n)}$ satisfying the following requirements:

- G has *large stretch*; that is, $l(n) > cn$ or even $l(n) > n^c$ for some constant $c > 1$. A large stretch is crucial for most cryptographic applications of PRGs.
- G has *linear circuit complexity*; that is, the output of G can be computed by a uniform family of (bounded fan-in) boolean circuits of size $O(l(n))$.
- G has *exponential security*; that is, there exists a constant $\delta > 0$ such that any algorithm running in time $2^{\delta n}$ can distinguish between the output of G and a truly random string of length $l(n)$ with at most a $2^{-\delta n}$ advantage. In typical PRG applications, exponential security is useful for minimizing the asymptotic length of the secret keys or the amount of true randomness.

We refer to a PRG as above as an *asymptotically optimal PRG*. Using this terminology, the main question we pose in this work is the following:

Which types of one-way functions imply an asymptotically optimal PRG?

2.1 Our Results

We prove the above conjecture under the assumption there exists an exponentially hard OWF f whose output on a random input x has a high min-entropy, where the required amount of entropy depends on its hardness. (A distribution has min-entropy k if the most likely element in its support occurs with probability 2^{-k}.)

We also re-examine previous constructions of HILL and of Haitner, Harnik, and Reingold [45] and note that by using hash functions and hardcore functions that can be computed by linear-sized circuits, we can obtain asymptotically optimal PRGs for regular OWFs, which are OWFs whose every output has the same preimage size. HILL examined regular

OWFs with known preimage size, and [45] examined regular OWFs with unknown preimage size; when the OWFs used are exponentially hard, we prove that their constructions yield asymptotically optimal PRGs. More specifically, we obtain the following result.

Theorem 2.1 (Main Theorem 1 (Informal))**.** *Suppose that $f : \{0,1\}^n \to \{0,1\}^n$ is $2^{\beta n}$-hard to invert. Suppose that either f is regular* or *that the min-entropy of $f(x)$ is bigger than γn for some constant γ such that $\gamma > 1 - \beta/3$. Then there exists an exponentially strong PRG $G : \{0,1\}^n \to \{0,1\}^{2n}$ that can be computed by linear-sized circuits using $O(1)$ oracle calls to f.*

Hardcore functions. Our construction of asymptotically optimal PRGs is obtained via a technical lemma about hardcore functions that may be of independent interest. A hardcore predicate b is a function that outputs a single bit $b(x)$ which is hard to predict even given $f(x)$. A hardcore *function* for a one-way function f is a function h (which can output more than one bit) whose output $h(x)$ is hard to distinguish from random even when $f(x)$ is known. Hardcore functions are a fundamental cryptographic object, with applications to pseudoentropy and pseudorandomness. Goldreich and Levin [40] introduced the first hardcore predicates and functions for general OWFs, showing that a random linear function is hardcore, and so is a product with a random Toeplitz matrix.

We examine families of linear functions $\mathcal{H} = \{h_i : \{0,1\}^n \to \{0,1\}^m\}$ which we term *bilinear uniform output hash families* that satisfy two properties. First, for any $x \neq 0$, the random variable $\mathcal{H}(x)$ is uniformly distributed over $\{0,1\}^m$. Second, $\mathcal{H}$ forms an additive group. Building on the work of Holenstein, Maurer, and Sjodin [52], we prove the following theorem.

Theorem 2.2 (Main Theorem 2)**.** *Let $H = \{h : \{0,1\}^n \to \{0,1\}^{\alpha n}\}$ be a bilinear uniform output hash family and let $f : \{0,1\}^n \to \{0,1\}^n$ be a $2^{\beta n}$-hard one-way function. Then H is a family of exponentially strong hardcore functions for f if $\beta > 2\alpha$.*

3 Zero Knowledge Protocols with Minimal Reliance on Random Inputs

A zero knowledge proof is an interactive protocol between a prover and a verifier where, for a $\mathcal{NP}$ language L and a public instance x, the prover attempts to convince the verifier that x is in L without revealing the witness for such membership. For instance, a prover might wish to prover to a verifier that a particular number n is a product of two primes p_1 and p_2 without revealing any information about p_1 or p_2.

We examine simultaneously resettable zero-knowledge proof systems. Such proof systems are of interest as examples of proof systems that are secure under very relaxed constraints on the re-use of the same randomness in multiple executions. In the case of *resettable zero knowledge (rZK)*, a malicious verifier may cheat against an honest prover who must use the same random tape polynomially many times. Further, *resettably sound* zero knowledge constrains the randomness used by the verifier: a malicious prover may try to cheat against an honest verifier who must use the same random tape polynomially many times. The former property was introduced and instantiated by Canetti, Goldreich, Goldwasser and Micali [16]; the later property was introduced by Micali and Reyzin [67] and later instantiated by Barak, Goldreich, Goldwasser and Lindell [9].

A question opened by [9] and resolved by Deng, Goyal and Sahai [26] is *"Does there exist a resettably sound rZK proof system for all $\mathcal{NP}$?"* [26] answered this question in the affirmative, but they required a construction with a security proof that required a non-black-box simulator strategy, which utilize the specific strategy of a cheating verifier in its specification. Currently, there is no known practical protocol that relies on a non-black-box simulation strategy, while for instance there *do* exist efficient constructions for cZK and concurrent non-malleable zero knowledge that rely on black-box simulation strategies [68, 73], which work against any verifier strategy.

It is proved in [9] that resettably sound black-box zero-knowledge arguments can be constructed for languages in $\mathcal{BPP}$. Instead, we study whether there exist *t-bounded resettably sound rZK proof systems with black-box simulation*, and more generally, with only a black-

box use of the code of the adversary (i.e., both the simulation and the proof of soundness do not rely on non-black-box uses of the code of the adverary). Such proof systems are rZK but also allow a malicious prover to conduct at most $t(n)$ resets against an honest verifier, where t is any fixed polynomial and n is the security parameter.

3.1 Our Results

For all $\mathcal{NP}$ and for any polynomial t, we construct a t-resettably sound rZK proof system with black-box use only of the code of the adversary and round complexity $O(n^\epsilon)$ for security parameter n and for any constant $\epsilon > 0$. We require standard assumptions as the existence of enhanced trapdoor permutations and collision-resistant hash functions.

Theorem 3.1 (Main Theorem 1 (Informal))**.** *Assuming the existence of enhanced trapdoor permutations and collision-resistant hash functions, protocol there exists a black-box t-bounded resettably sound rZK proof system for any L in $\mathcal{NP}$.*

4 An Efficient Secure Protocol for Pattern Matching

Pattern matching is fundamental to computer science. It is used in many areas, including text processing, database search [2], networking and security applications [70] and recently in the context of bioinformatics and DNA analysis [72, 82, 11]. It is a problem that has been extensively studied, resulting in several efficient (although insecure) techniques to solve its many variations, e.g,[81, 1, 64, 58]. The most common interpretation of the pattern matching problem is the following: given a finite alphabet Σ, a text $T \in \Sigma^n$ and a pattern $p \in \Sigma^m$, the *exact pattern matching decision problem* requires one to decide whether or not a pattern appears in the text. The *exact pattern matching search problem* requires finding all indices i of T (if any) where p occurs as a substring starting at position i. If we denote by T_i the ith character of T, the output should be the set of matching positions $MP := \{i \mid p \text{ matches } T \text{ beginning at } T_i\}$. The following generalizations of the exact matching problem are often encountered, where the output in all cases is the set MP:

- *Pattern matching with single character wildcards*[1]*:* There is a special character "$*$" $\notin \Sigma$ that matches any single character of the alphabet, where $p \in \{\Sigma \cup \{*\}\}^m$ and $T \in \Sigma^n$. Using such a "wildcard" character allows one pattern to be specified that could match several sequences of characters. For example the pattern "$TA*$" , would match any of the following character sequence in a text[2]: TAA, TAC, TAG, and TAT.
- *Substring pattern matching:* Fix some $l \leq m$; a match for p is found whenever there exists in T an m-length string that differs in l characters from p (i.e., has Hamming distance l from p). For example, the pattern "TAC" has $m = 3$. If $l = 1$, then any of the following words would match: $*AC$, $T * C$, or $TA*$; note that this is an example of non-binary substring matching.

A secure version of pattern matching has many applications. For example, secure pattern matching can help secure databases containing medical information, such as DNA records, while still allowing one to perform pattern matching operations on such data. The need for privacy-preserving DNA matching has been highlighted in recent papers [8, 59, 80]. In addition to the case of DNA matching, where substring matching may be particularly useful, Hamming distance-based approximate matching has also been demonstrated in the case of secure facial recognition [72]. We note that both of these settings require computation over non-binary alphabets.

4.1 Our Results

This paper presents a new protocol for arbitrary alphabets, **5**ecure **P**attern **M**atching (or 5PM), that addresses, in addition to exact matching, more expressive search queries including single character wildcards and substring pattern matching, in addition to providing the ability to hide pattern length.

5PM performs exact, single character wildcard, and substring pattern matching in the honest-but-curious and malicious (static corruption) models. Our malicious model protocol

[1]Such wildcards are also called "do not cares" and "mismatches" in the literature.

[2]Here and throughout, we use the DNA alphabet ($\Sigma = \{A, C, G, T\}$) for examples.

requires $O((m+n)k^2)$ bandwidth complexity. Further, our protocol can be specified to require 2 (one-way) rounds of communication in the semi-honest model and 8 (one-way) rounds of communication in the malicious model.

We construct our protocols by reducing the problems of Hamming distance and pattern matching, including single character wildcards and substring matching, to a sequence of linear operations. We then rely on the observation that such linear operations, such as the inner products and matrix multiplication, can be efficiently computed in the malicious model using additively homomorphic encryption schemes.

5 Outline

In Chapter 2, we examine the construction of asymptotically optimal pseudorandom generators as well as construct hardcore functions with linear circuit complexity. In Chapter 3, we construct nearly simultaneously resettable black-box zero knowledge proofs for any language L in $\mathcal{NP}$. Finally, in Chapter 4, we examine efficient protocols to securely evaluate various forms of pattern matching.

CHAPTER 2

On Linear-Size Pseudorandom Generators and Hardcore Functions

1 Introduction

A *pseudorandom generator* (PRG) [81, 85] is a deterministic algorithm which stretches a short random seed into a longer output which looks random to any computationally bounded observer[1]. PRGs have numerous applications in cryptography. In particular, they serve as useful building blocks for basic cryptographic tasks such as (symmetric) encryption, commitment, and message authentication.

A seemingly weaker primitive, which satisfies a much milder form of hardness requirement, is a *one-way function* (OWF). A OWF is an efficiently computable function which is hard to invert on a random input. We say that f is $t(n)$-*hard to invert* (or $t(n)$-hard for short) if every algorithm running in time $t(n)$ can find a preimage of $f(x)$ for a random $x \in \{0,1\}^n$ with at most $1/t(n)$ probability, for all sufficiently large n. We say that f is *exponentially hard* if it is $2^{\beta n}$-hard for some constant $\beta > 0$.

Every PRG which significantly stretches its seed is also a OWF. However, because of its crude form of security, a OWF is easier to construct heuristically than a PRG. There are many natural candidates for a OWF (even an exponentially strong OWF) which do not immediately give rise to a similar PRG. This motivated a line of work on constructing PRGs from different types of OWFs, which culminated in the seminal result of Håstad, Impagliazzo, Levin and Luby (HILL) [47] that a PRG can be constructed from an *arbitrary* OWF. More recently,

[1]We would like to thank Benny Applebaum, Andrej Bogdanov, Iftach Haitner, and Salil Vadhan for helpful discussions and comments on this work.

there has been another fruitful line of work on simplifying and improving the efficiency of the HILL construction [51, 43, 44, 45, 46, 42, 83].

The main focus in the above works has been on optimizing efficiency under *minimal assumptions*. The present work is motivated by the following dual question: under which assumptions can we get *optimal efficiency*? Ideally, we would like to obtain a PRG $G : \{0,1\}^n \to \{0,1\}^{l(n)}$ satisfying the following requirements:

- G has *large stretch*; that is, $l(n) > cn$ or even $l(n) > n^c$ for some constant $c > 1$. A large stretch is crucial for most cryptographic applications of PRGs.
- G has *linear circuit complexity*; that is, the output of G can be computed by a uniform family of (bounded fan-in) boolean circuits of size $O(l(n))$. This implies linear-time computation also in other, more liberal, models such as RAM.
- G has *exponential security*; that is, there exists a constant $\delta > 0$ such that any algorithm running in time $2^{\delta n}$ can distinguish between the output of G and a truly random string of length $l(n)$ with at most a $2^{-\delta n}$ advantage. In typical PRG applications, exponential security is useful for minimizing the asymptotic length of the secret keys or the amount of true randomness.

We refer to a PRG as above as an *asymptotically optimal PRG*. Using this terminology, the main question we pose in this work is the following:

Which types of one-way functions imply an asymptotically optimal PRG?

A natural conjecture is that an asymptotically optimal PRG can be constructed from any OWF $f : \{0,1\}^n \to \{0,1\}^n$ which can be computed by linear-size circuits and is exponentially hard to invert. However, this conjecture does not seem to follow from the current state of the art. A recent result of Vadhan and Zheng [83] (improving on [44, 46]) comes close to proving the conjecture, implying a PRG construction which satisfies the first two requirements but falls short of the third. More concretely, the construction adds a $polylog(n)$ multiplicative overhead to the seed length. A recent result of Applebaum [5] can be used to get an asymptotically optimal PRG from an exponentially strong variant of an assumption

due to Goldreich [38]. Roughly speaking, the assumption asserts that certain randomized choices of a "local" function, in which each bit of the output depends on a constant number of input bits, are one-way with high probability. The question of constructing asymptotically optimal PRGs under more general types of OWFs remained open.

1.1 Our Contribution

We prove the above conjecture under the additional assumption that the output of f on a random input x has a high min-entropy, where the required amount of entropy depends on its hardness. (A distribution has min-entropy k if the most likely element in its support occurs with probability 2^{-k}.)

We also re-examine previous constructions of HILL and of Haitner, Harnik, and Reingold [45] and note that by using hash functions and hardcore functions that can be computed by linear-sized circuits, we can obtain asymptotically optimal PRGs for regular OWFs, which are OWFs whose every output has the same preimage size. HILL examined regular OWFs with known preimage size, and [45] examined regular OWFs with unknown preimage size; when the OWFs used are exponentially hard, we prove that their constructions yield asymptotically optimal PRGs. More specifically, we obtain the following result.

Theorem 1.1 (Main Theorem 1 (Informal))**.** *Suppose that $f : \{0,1\}^n \to \{0,1\}^n$ is $2^{\beta n}$-hard to invert. Suppose that either f is regular* or *that the min-entropy of $f(x)$ is bigger than γn for some constant γ such that $\gamma > 1 - \beta/3$. Then there exists an exponentially strong PRG $G : \{0,1\}^n \to \{0,1\}^{2n}$ that can be computed by linear-sized circuits using $O(1)$ oracle calls to f.*

Using a standard tree-based construction for PRG extension (see Section 7), the above theorem yields an asymptotically optimal PRG with an arbitrary polynomial stretch.

We note that the above entropic requirement seems quite mild and in some cases of interest it can be proved unconditionally. In particular, there are natural variants of Goldreich's OWF candidate that can be shown to have fractional entropy that tends to 1 with the locality degree (see [19] and Section 5.2), whereas the expected hardness of inverting does

not seem to decrease (and possibly even grows) with the degree. This is the case even for families of local OWF candidates to which the results from [5] do not seem to apply.

Hardcore functions. Our construction of asymptotically optimal PRGs is obtained via a technical lemma about hardcore functions that may be of independent interest. Recall that a hardcore predicate b is a function that outputs a single bit $b(x)$ which is hard to predict even given $f(x)$. A hardcore *function* for a one-way function f is a function h (which can output more than one bit) whose output $h(x)$ is hard to distinguish from random even when $f(x)$ is known. More precisely, we allow h to be picked at random from a collection of functions H and provide a description of h as an additional input to the distinguisher. Hardcore functions are a fundamental cryptographic object, with applications to pseudoentropy and pseudorandomness. Goldreich and Levin [40] introduced the first hardcore predicates and functions for general OWFs, showing that a random linear function is hardcore, and so is a product with a random Toeplitz matrix.

We examine families of linear functions $\mathcal{H} = \{h_i : \{0,1\}^n \to \{0,1\}^m\}$ which we term *bilinear uniform output hash families* that satisfy two properties. First, for any $x \neq 0$, the random variable $\mathcal{H}(x)$ is uniformly distributed over $\{0,1\}^m$. Second, $\mathcal{H}$ forms an additive group. Building on the work of Holenstein, Maurer, and Sjodin [52], we show that any such family of functions is hardcore for any sufficiently hard OWF. In fact, our work simplifies the proof in [52] that the set of all matrices and the set of all Toeplitz matrices are families of hardcore functions.

Theorem 1.2 (Main Theorem 2)**.** *Let $H = \{h : \{0,1\}^n \to \{0,1\}^{\alpha n}\}$ be a bilinear uniform output hash family and let $f : \{0,1\}^n \to \{0,1\}^n$ be a $2^{\beta n}$-hard one-way function. Then H is a family of exponentially strong hardcore functions for f if $\beta > 2\alpha$.*

A construction of linear-size computable pairwise independent hash functions was given by Ishai, Kushilevitz, Ostrovsky and Sahai [55]. Observing that the construction can be instantiated so that each function in the family is affine, and constructing linear uniform-output hash families from such families, we can use the above theorem to obtain linear-

size computable hardcore functions with a long output. Using such a hardcore function, the construction of an asymptotically optimal PRG proceeds in a simple way. We first extract the randomness from the output of f by applying a (linear-size) pairwise independent hash function (appealing to the Leftover Hash Lemma [47]). Then, we extract sufficient pseudorandomness from the input of f by applying the (linear-size) hardcore function to the input x. If f has sufficiently high min-entropy and is hard enough to invert, these techniques combine so that the output has length cn for $c > 1$. From this PRG, we can construct PRGs with an arbitrary polynomial stretch that can be computed by circuits of size linear in their outputs, and with hardness exponential in the seed length.

1.2 Related Work

Pseudorandom Generators. Following the pioneering works of Blum and Micali [81] and Yao [85], who constructed a PRG from a one-way *permutation*, Goldreich, Krawczyk and Luby [39] constructed a PRG from any *regular* OWF with unknown preimage size (a OWF is regular if every output of f has the same preimage size). Håstad, Impagliazzo, Levin and Luby [47] gave the first construction of a PRG from any OWF. The first effort towards simplifying and improving the HILL construction was made by Holenstein [51], who also explicitly considered the case of exponentially strong OWFs. Haitner, Harnik and Reingold [43, 44, 45] improved the construction of [39] by relying only on pairwise independent hash functions ([39] had required n-wise independent hash functions) and by reducing the seed length. We will demonstrate that some of the constructions of [47] and [45] can in fact be instantiated to be computable by linear-sized circuits, but for different classes of OWFs (regular rather than high min-entropy) than examined here. More recently, Haitner, Reingold and Vadhan [46] further improved the seed length of PRGs from general OWFs. The most efficient general constructions to date are given in the aforementioned work of Zheng and Vahdan [83], who also noted that combining their construction with the pairwise independent hash functions of [55] gives a linear-stretch linear-size PRG from *any* exponentially hard OWF. (This construction does not depend on the hash functions being

affine.) As discussed above, this construction still falls short of our main goal because of its polylogarithmic overhead to the seed length, but otherwise it is stronger in several important aspects. In particular, it does not require f to satisfy any entropy requirement.

Constructions of PRGs in NC^0 (i.e., ones in which each output bit depends on a constant number of input bits) were first given by Applebaum, Ishai, and Kushilevitz [6] under standard assumptions; however, these constructions have sublinear stretch. Linear-stretch constructions were given by the same authors in [7] under an indistinguishability assumption due to Alekhnovich [3]. A construction under a one-wayness assumption was recently given by Applebaum [5], who under similar assumptions obtained a PRG with polynomial stretch in NC^0. However, the underlying OWFs in these constructions are restricted to be (special distributions over) NC^0 functions.

Goldreich's One-Way Function. Goldreich conjectured [38] the following graph-based one-way function: Consider a d-ary predicate P and a bipartite graph $G = (V, E)$, here assumed to be bipartite with right degree d (denoted *d-regular*), n left vertices (nodes) and n right vertices. Define $f : \{0,1\}^n \to \{0,1\}^n$ by labeling the left (input) nodes of the bipartite graph with the bits of x. We define the jth output bit of f on x, $f(x)_j$, as $P(x_{i_1}, \ldots, x_{i_d})$, where $u_{i_1}, \ldots, u_{i_d}$ are the input nodes that are in the neighborhood of v_j, the jth output node of G. We note that f can be computed by linear-sized circuits as long as d is a constant. We will examine the min entropy of two different variants of this function.

Recent work on this type of functions [74, 4, 19, 13] may be viewed as supporting the possibility that they are exponentially hard; however, Bogdanov and Qiao [13] have shown that for variants where the output stretch is a large constant (at least exponential in the input degree), there exist instantiations that are invertible in polynomial time. Applebaum's construction of a linear-stretch PRG in NC^0 [5] uses a variant of Goldreich's one-way function with large constant stretch. He demonstrates that if such a function, using any single predicate for all of the output bits, is one-way, then the output has sufficiently good pseudoentropy from which to construct a PRG. By contrast, our one-way function is from n bits to n bits, and it is not clear that the technique from [5] is applicable in this case.

Finally, in Section 5.2, we show that a random d-local one-way function from n bits to n bits, instantiated with a random and independent d-ary predicate for *each* output bit of the function, has high min-entropy except with exponentially small probability over the choice of graph and predicates. Previous work (for example, [19]) have examined more concrete predicates and proved them also to have min-entropy except with exponentially small probability over the choice of the function.

Hardcore Functions. Goldreich and Levin [40] demonstrated that the set of all inner product functions constitutes a family of hardcore predicates for any one-way function. More generally, they proved that the set of all linear functions with input in $\{0,1\}^n$ and the set of Toeplitz matrices with input $\{0,1\}^n$ are families of hardcore functions for any one-way function (for appropriately sized outputs). The central idea of their proof is that if a random XOR of a candidate hardcore function output is hard to distinguish, then the function is indeed hardcore; they constructed such an argument for the set of all matrices and Toeplitz matrices, respectively, by direct calculation. By contrast, we will need to use more general structural arguments to achieve our claim.

Näslund [71] showed that the family of all affine functions over $GF[2^n]$ and the family of all linear functions over the integers modulo a prime are families of hardcore functions for any one-way function.

Holenstein, Maurer and Slodin [52] gave a complete classification of all so-called bilinear hardcore function families over arbitrary fields; that is, the hardcore functions are additively homomorphic both in their function inputs and in the strings that represent each function (e.g. the set of hardcore functions form an additive group). Their work therefore also obtains the results of [40] as a corollary. We will prove our result by showing that all bilinear uniform-output hash families are full rank, and are therefore hardcore, using the results of [52].

2 Bilinear Uniform-Output Hash Families are Hardcore

We demonstrate that a group of linear functions H whose output for any x, $H(x)$ is uniformly distributed over its range, is hardcore. We begin with some definitions.

2.1 Definitions and Preliminaries

We denote by U_n the random variable uniformly distributed over $\{0,1\}^n$.

Definition 2.1. *Let $\mathbb{F}$ be a field. Then a function $h : \mathbb{F}^n \times \mathbb{F}^k \to \mathbb{F}^m$ is* bilinear *if h can be specified by $h_i(x,r) = x^T M r$ where M is some $n \times k$ matrix with coefficients in $\mathbb{F}$.*

In the course of this paper, $\mathbb{F}$ will be $\mathbb{Z}_2$.

Definition 2.2. *Let $h : \mathbb{F}^n \times \mathbb{F}^k \to \mathbb{F}^m$ be a bilinear function over $\mathbb{F}$. Then h is* full rank *if for every linear map $l : \mathbb{F}^m \to \mathbb{F}$, $\text{rank}(l \circ h) = n$, where $l \circ h : \mathbb{F}^n \times \mathbb{F}^k \to \mathbb{F}$ and the rank of $l \circ h$ is the rank of the matrix $M'_{l \circ h}$ specifying this mapping.*

Definition 2.3. *Let $H = \{h_i : \{0,1\}^n \to \{0,1\}^m\}$ be a multiset of functions. We say that H is a* family of uniform-output hash functions *if for every non-zero $x \in \{0,1\}^n$, the random variables $H(x)$ induced by a uniform choice of h from H is uniformly distributed over $\{0,1\}^m$. If every $h_i \in H$ is an linear function over the binary field (i.e., a function of the form $A_i x$), we call H a* linear uniform-output (LUO) hash family*.*

Further, a LUO hash family that can be expressed as a bilinear function $h : \{0,1\}^n \times \{0,1\}^k \to \{0,1\}^m$ is denoted a bilinear uniform-output (BLUO) hash family*.*

We will typically consider infinite collections of families $H_{n,m}$ parameterized by the input and output length. In such a case we require the existence of a representation length $\ell_{n,m} = poly(n,m)$ such that $H_{n,m}$ contains $2^{\ell_{n,m}}$ (not necessarily distinct) functions h_i indexed by all binary strings of length $\ell_{n,m}$ (which equals k in the above definitions). For convenience, we will abuse notation and refer to $h_i \in H_{n,m}$ as both a function and the string representing it. We assume that there is a polynomial-time evaluation algorithm that, given h_i and x, outputs $h_i(x)$. In fact, we will rely on families for which this algorithm can be implemented by linear-size circuits.

We also introduce the more standard notion of pairwise independent hash families.

Definition 2.4. *Let $H' = \{h_i : \{0,1\}^n \to \{0,1\}^m\}$ be a multiset of functions. We say that H' is a* family of pairwise independent hash functions *if for every distinct $x, y \in \{0,1\}^n$, the random variables $h(x)$ and $h(y)$ induced by a uniform choice of h from H' are independently and uniformly distributed over $\{0,1\}^m$. If every $h_i \in H'$ is an affine function over the binary field (i.e., a function of the form $A_i x + b_i$), we call H' an* affine pairwise independent (API) hash family.

We note that a uniform-output hash family can generally be constructed from an affine pairwise-independent hash family.

Claim 2.5. *Let H' be an API hash family. Then the multiset $H = \{h_i : h_i(x) = h'_i(x) - h'(0), h'_i \in H'\}$ is an LUO hash family.*

The proof of the claim is immediate from the fact that for any $x \neq 0$, $H'(x)$ and $H'(0)$ are distributed uniformly and independently at random, and that each function in H is linear.

Definition 2.6. *A polynomial-time computable function $f : \{0,1\}^n \to \{0,1\}^{l(n)}$ is* $(t(n), \epsilon(n))$ one-way *if for every probabilistic algorithm A' running in time at most $t(n)$ and all sufficiently large n,*

$$\Pr[x \leftarrow U_n : A'(f(x), 1^n) \in f^{-1}(f(x))] < \epsilon(n).$$

For n sufficiently large, a function $f : \{0,1\}^n \to \{0,1\}^{l(n)}$ is β-exponential one-way *if it is $(2^{\beta n}, 2^{-\beta n})$ one-way and simply* one-way *if f is $(p(n), 1/p(n))$ one-way for every polynomial p.*

Definition 2.7. *For a one-way function $f : \{0,1\}^n \to \{0,1\}^{l(n)}$, a collection of polynomial-time computable functions $\mathcal{H}_{n,m} = \{h_i : \{0,1\}^n \to \{0,1\}^m\}$ is a* family of $(t(n), \epsilon(n))$ hardcore functions of f *if for every probabilistic algorithm D running in time at most $t(n)$ and for all sufficiently large n,*

$$|\Pr[x \leftarrow U_n, i \leftarrow I : D(f(x), h_i, h_i(x)) = 1] - \Pr[x \leftarrow U_n, i \leftarrow I, y \leftarrow U_m : D(f(x), h_i, y) = 1]| < \epsilon(n),$$

where I is a random variable distributed uniformly over the index set of $\mathcal{H}_{n,m}$.

A collection of functions $\mathcal{H}_{n,m}$ is a β-exponential hardcore family of f *if it is a* $(2^{\beta n}, 2^{-\beta n})$ hardcore family of f *and simply a* hardcore family of f *if $\mathcal{H}_{n,m}$ is a $(p(n), 1/p(n))$ hardcore family of f for every polynomial p.*

We state the hardcore function result of [52] specified for a full rank bilinear function. We slightly modify their statement to take the initial distinguisher time into account.

Theorem 2.8 (Theorem 25 in [52])**.** *Let $f : \{0,1\}^n \to \{0,1\}^n$ be any efficiently computable function. Let $h : \{0,1\}^n \times \{0,1\}^k \to \{0,1\}^m$ be any efficiently computable bilinear function with $k \in poly(n)$. Suppose there exists a distinguisher D running in time $t(n)$ and $\epsilon, \delta > 0$ such that $\Pr[x \leftarrow U_n, r \leftarrow U_k : D(f(x), r, h(x,r)) = 1] = \delta$ and $\Pr[x \leftarrow U_n, r \leftarrow U_k, y \leftarrow U_m : D(f(x), r, y) = 1] = \delta(1+\epsilon)$. Then there exists an algorithm A such that $\Pr[x \leftarrow U_n : f(A(f(x))) = f(x)] \geq \frac{\delta\epsilon^2}{4 \cdot 2^{2m}} = \epsilon'_{\delta,\epsilon}(n)$. A runs in* expected *time $t'_{\delta,\epsilon}(n) = \frac{2^{2m}}{\delta\epsilon^2} \cdot poly(n) \cdot t(n)$.*

Using Markov's inequality, we obtain the following result that does not run in expected time.

Theorem 2.9. *Let $f : \{0,1\}^n \to \{0,1\}^n$ be any efficiently computable function. Let $h : \{0,1\}^n \times \{0,1\}^k \to \{0,1\}^m$ be any efficiently computable bilinear function with $k \in poly(n)$. Suppose there exists a distinguisher D running in time $t(n)$ and $\epsilon, \delta > 0$ such that $\Pr[x \leftarrow U_n, r \leftarrow U_k : D(f(x), r, h(x,r)) = 1] = \delta$ and $\Pr[x \leftarrow U_n, r \leftarrow U_k, y \leftarrow U_m : D(f(x), r, y) = 1] = \delta(1+\epsilon)$. Let $\epsilon'_{\delta,\epsilon}(n)$ and $t'_{\delta,\epsilon}(n)$ be defined as in Theorem 2.8. Then for every function $\alpha(n)$ there exists an algorithm A_α that runs in time $t''_{\delta,\epsilon}(n) = \alpha(n) \cdot t'_{\delta,\epsilon}(n)$ such that $\Pr[x \leftarrow U_n : f(A_\alpha(f(x))) = f(x)] \geq \epsilon'_{\delta,\epsilon}(n) - \frac{1}{\alpha(n)}$.*

Proof of Theorem 2.9. Let A_α be identical to the algorithm A from Theorem 2.8 except that it halts after time $\alpha(n) \cdot t'_{\delta,\epsilon}(n)$. Then the probability that A_α fails to invert f equals the probability that A runs in time greater than $\alpha(n) \cdot t'_{\delta,\epsilon}(n)$ plus the probability that A_α does not halt A but still fails to invert. By Markov's inequality, the former probability is $\leq \frac{1}{\alpha(n)}$ and by Theorem 2.8, the latter probability is $\leq 1 - \epsilon'_{\delta,\epsilon}(n)$; the theorem follows.

2.2 BLUO Hash Families are Hardcore

We now state and prove our main result about hardcore function families.

Theorem 2.10. *Let $H_{n,l(n)}$ be a BLUO hash family and let $f : \{0,1\}^n \to \{0,1\}^n$ be a $(t(n), 1/t(n))$ one-way function. Then $H_{n,l(n)}$, is a $(t'(n), 1/t'(n))$ family of hardcore functions of f if $2\theta(l(n) + \log t'(n)) < \log t(n)$ for any constant $\theta > 1$.*

Corollary 2.11. *Let $H_{n,l(n)}$ be a BLUO hash family and let $f : \{0,1\}^n \to \{0,1\}^n$ be a one-way (resp. β exponential one-way) function. Then $H_{n,l(n)}$ is a family of hardcore functions (resp. is a $(2^{\Omega(n)}, 2^{-\Omega(n)})$ family of hardcore functions) of f for any $l(n) \in O(\log n)$ (resp. $l(n) < \frac{\beta n}{2\theta}$ for any constant $\theta > 1$).*

Before proceeding to the proof, we require the following technical lemma.

Lemma 2.12. *Let $\mathcal{H}$ be a BLUO hash family specified by the bilinear function h. Then h is full rank.*

Proof of Lemma 2.12. Let $h : \{0,1\}^n \times \{0,1\}^k \to \{0,1\}^m$ be the bilinear function that specifies $\mathcal{H}$. Then, by definition of LUO hash families, for any $0 \neq x \in \{0,1\}^n$, the distribution $\{h(x,r)\}_{r \leftarrow U_k}$ is distributed uniformly over $\{0,1\}^m$. Let $l : \{0,1\}^m \to \{0,1\}$ be an arbitrary non-zero linear function. Then for any $0 \neq x \in \{0,1\}^n$, $\{l \circ h(x,r)\}_{r \leftarrow U_k}$ is also distributed uniformly over $\{0,1\}$ and, in particular, the linear map $r \mapsto h(x,r)$ is surjective onto $\{0,1\}$ for any non-zero x.

Let M_l be the $n \times k$ matrix denoting $l \circ h : \{0,1\}^n \times \{0,1\}^k \to \{0,1\}$. We would like to prove that $\text{rank}(M_l) = n$. This follows from the fact that for every non-zero $x \in \{0,1\}^n$, there equals some r_x such that $x^T \cdot M \cdot r_x = 1$, which implies that every non-trivial linear combination of the rows of M_l is non-zero, and the lemma follows.

We now proceed to prove Theorem 2.10.

Proof of Theorem 2.10. We will proceed by contradiction, and assume that there exists a probabilistic algorithm D running in time $t'(n)$ such that $|\Pr[x \leftarrow U_n, i \leftarrow I :$

$D(f(x), h_i, h_i(x)) = 1] - \Pr[x \leftarrow U_n, i \leftarrow I, y \leftarrow U_{l(n)} : D(f(x), hi, y) = 1]| > \epsilon'(n) = 1/t'(n)$ for infinitely many n, where I is the random variable uniformly distributed over $H_{n,l(n)}$.

More specifically, let $\Pr[x \leftarrow U_n, i \leftarrow I : D(f(x), h_i, h_i(x)) = 1] = \delta$ and $\Pr[x \leftarrow U_n, i \leftarrow I, y \leftarrow U_{l(n)} : D(f(x), hi, y) = 1] = (1 + \epsilon)\delta$. Without loss of generality, let $\delta \geq 1/2$ (otherwise, let D' be the algorithm that outputs the opposite bit that D does and use D'). Then $\epsilon\delta \geq \epsilon'(n)$.

Let $\alpha(n)$ be such that for some $\theta > 1$, $2\theta(l(n) + \log t'(n)) + \log \alpha(n) < \log t(n)$. Since by Lemma 2.12, $\mathcal{H}$ can be specified by a full rank bilinear function, Theorem 2.9 implies that there exists an algorithm A_α and some $c > 0$ that inverts f in time $\alpha(n) \cdot \frac{2^{2l(n)}}{\delta\epsilon^2} \cdot n^c \cdot t'(n) \leq \alpha(n) \cdot \frac{2^{2l(n)}}{\epsilon'(n)^2} \cdot n^c \cdot t'(n)$, which, by assumption, is less than $t(n)$. Further, A_α inverts f with probability $\geq \frac{\delta\epsilon^2}{4 \cdot 2^{2l(n)}} - \frac{1}{\alpha(n)} \geq \frac{\epsilon'(n)^2}{4 \cdot 2^{2l(n)}} - \frac{1}{\alpha(n)}$, which, by assumption, is larger than $\epsilon(n) = 1/t(n)$, which contradictions the one-wayness of f and completes the proof.

2.3 Linear-Size Hardcore Functions

Ishai et al in [55] provide a construction for an API hash family that can be computed by linear-sized circuits; from this family, we construct a BLUO hash family. We sketch their construction here, but first require a definition first given by Canetti et al in [15]. Let $L \subset \{1, ..., n\}$, and let $y \in \{0,1\}^n$. Define $[y]_L$ to be the vector of length $|L|$ consisting of the bits of y at the locations in L.

Definition 2.13. *A function $f : \{0,1\}^n \rightarrow \{0,1\}^m$ is a l-*exposure resilient function *(l-ERF) if for any $L \subset \{1, ..., n\}$ of size $|L| = n - l$, and for $r \in \{0,1\}^n$ and $R \in \{0,1\}^m$ selected uniformly and independently at random, the following distributions are identical:*

$$([r]_L, f(r)) \equiv ([r]_L, R). \tag{2.1}$$

Proposition 2.14 (Theorem 3.2 in [55])**.** *For sufficiently large n, there exists a family of l-ERF's from $\{0,1\}^n$ to $\{0,1\}^m$ that are computable by linear-sized circuits where $l = \Theta(n)$ and $m = \Theta(n)$.*

From their construction of linear-size l-ERFs, [55] obtain linear-size pairwise-independent hash families.

Proposition 2.15 (Theorem 3.3 in [55])**.** *For sufficiently large n, there exists a family of pairwise independent hash functions from $\{0,1\}^n$ to $\{0,1\}^m$ where $l = \Theta(n)$ and $m = \Theta(n)$ and where each hash function is computable by linear-sized circuits.*

The hash functions in [55] for Proposition 2.15 are constructed as the composition $H = e \circ \mathcal{H} \circ c$, where e is a suitably chosen affine l-ERF, c is a linear error correcting code, and $\mathcal{H} = (H_{\mu,\mu})^{O(m)}$ is a collection of $O(m)$-length sequences of constant-sized hash functions, where μ is a constant. Once e and c are fixed, the hash functions in [55] rely solely upon the input of the $O(m)$ hash functions $h_i : \{0,1\}^\mu \to \{0,1\}^\mu$.

We can select these h_i from the family of all affine transformations from $\{0,1\}^\mu \to \{0,1\}^\mu$, which itself is a family of pairwise independent hash functions. Then the corresponding family of pairwise independent hash functions is affine. Each h_i consists of a pair (A, b) for a μ by μ matrix A and a μ-length vector b ($h_i(x) = Ax + b$). Further, a string representation of $\mathcal{H} = (H_{\mu,\mu})^{O(m)}$ consists of the ordered list of A_i and b_i. Under this representation, the distribution of the collection of all affine transformations from μ bits to μ bits is identical to $U_{\mu^2+\mu}$, and so the distribution of $h \in H = e \circ \mathcal{H} \circ c$ is identical to $U_{[O(m)(\mu^2+\mu)]}$. We note that this multiset can be converted into a LUO hash family in the following way: Denote by e' the linear part of e and c' the linear part of c. Construct H' as the set $e' \circ \tilde{\mathcal{H}} \circ c'$ without the corresponding additive vectors, where $\tilde{\mathcal{H}}$ is the set of matrices from $\mathcal{H}$ (without repetition). Elements of H' may be repeated, but if so, each element is repeated equally; this is because $\phi(h) = e' \circ h \circ c'$ is a group homomorphism on $\tilde{\mathcal{H}}$; without loss of generality, let H' be the corresponding set without repetitions. Note that the hash family is quantified by the set of all binary strings of length $O(m\mu^2)$ by construction. Then, by the construction in Claim 2.5 (and taking into account the fact that each matrix in $\mathcal{H}$ was repeated an equal number of times by construction), we have the following result.

Lemma 2.16. *For sufficiently large n, there exists a BLUO hash family from $\{0,1\}^n$ to $\{0,1\}^m$, where $l = \Theta(n)$ and $m = \Theta(n)$ and where each hash function is computable by*

linear-sized circuits.

Using these hash functions as input for Theorem 2.10 yields the following result.

Corollary 2.17. *Let $f : \{0,1\}^n \to \{0,1\}^n$ be a β-exponential one-way function. For any $l(n)$ and $\theta > 1$ such that $2l(n) < \beta n\theta$, there exists a BLUO hash family $H_{n,l(n)}$ such that $H_{n,l(n)}$, is a $(2^{\Omega(n)}, 2^{-\Omega(n)})$ family of hardcore functions of f, and where each $h \in H_{n,l(n)}$ can be computed by linear-sized circuits.*

3 PRGs Computable By Linear-Sized Circuits

3.1 Introduction

We discuss how hardcore functions that can be computed by linear-sized circuits can be used to construct linear-stretch PRGs that can be computed by linear-sized circuits. When f is an exponentially hard one-way function, various assumptions about the min-entropy of the output of f can be used to construct such PRGs. We first construct a PRG from a restriction of f of our own before examining previously made restrictions on f that have been used in the past to construct linear-stretch PRGs.

3.2 Definitions and Lemmas

We first introduce some definitions and lemmas needed for the results in this section.

Definition 3.1. *A $(t(n), \epsilon(n))$ pseudorandom generator (PRG) is a polynomial-time computable function $G : \{0,1\}^n \to \{0,1\}^{l(n)}$ such that:*

(i) $l(n) > n$ for all $n \in \mathbb{N}$.

(ii) For every probabilistic algorithm A running in time at most $t(n)$ (thought of as a distinguisher) and for all sufficiently large n,

$$|\Pr[A(G(U_n), 1^n) = 1] - \Pr[A(U_{l(n)}, 1^n) = 1]| < \epsilon(n).$$

A linear-stretch PRG is one where $l(n) = cn$ for some $c > 1$.

Definition 3.2. *For a random variable X distributed over $\mathcal{X}$, the min-entropy H_∞ of X is* $H_\infty(X) = -\log(\max_{x\in\mathcal{X}} \Pr[X = x])$.

Definition 3.3. *For random variables X and Y defined over $\mathcal{Z}$, the statistical distance δ between X and Y is* $\delta(X,Y) = \frac{1}{2}\sum_{z\in\mathcal{Z}} |\Pr[X = z] - \Pr[Y = z]|$. *If $\delta(X,Y) = 0$, then X and Y are said to be identical.*

Definition 3.4. *Two random variables X and Y defined over $\mathcal{Z}$ are said to be $(t(n), \epsilon(n))$ computationally indistinguishable if for every probabilistic algorithm D running in time at most $t(n)$ and for all sufficiently large n,*

$$|Pr[D(X, 1^n) = 1] - Pr[D(Y, 1^n)]| < \epsilon(n).$$

We recall the Leftover Hash Lemma as introduced by Impagliazzo, Levin and Luby [54].

Lemma 3.5 (Leftover Hash Lemma)**.** *Let X be a random variable over $\{0,1\}^n$ with $H_\infty(X) > m$, and let $l = m - 4e$. Let H be a pairwise independent family of hash functions from $\{0,1\}^n$ to $\{0,1\}^l$, where the distribution over the strings of H is identical to U_k for some $k = poly(n, l)$. Then $\delta((h, h(X)), U_{l+k}) < 3/2^e$, where h is drawn uniformly at random from H.*

3.3 PRGs for One-Way Functions with Lower-Bounded Min-Entropy

We demonstrate that there exist a linear stretch pseudorandom number generators that can be computed by linear-sized circuits provided that there exists a suitable class of exponentially hard one-way functions. In Section 5, we will discuss the plausibility of these assumptions.

Assumption 3.6. *There exists $\beta > 0, \theta > 1$, γ such that $\gamma > 1 - \frac{\beta}{2\theta}$ and a function $f : \{0,1\}^n \to \{0,1\}^n$, with the following properties:*

(i) f is a β-exponential one-way function.

(ii) f can be computed by linear-size circuits.

(iii) $H_\infty(f(U_n)) > \gamma n$.

Under this assumption, the following theorem can be proved.

Theorem 3.7. *If Assumption 3.6 holds, there exists a* $(2^{\Omega(n)}, 2^{-\Omega(n)})$ *linear-stretch PRG* G *that can be computed by linear-sized circuits with a single oracle call to* f.

Using either the construction in [55] (see Section 4.1 footnote 3; see Section 7 for more details) or the construction in [37] (see Section 3.3.2) with the PRG G of Theorem 3.7, we obtain the following corollary.

Corollary 3.8. *If Assumption 3.6 holds, then for any polynomial* $l(n) > n$ *there exists a* $(2^{\Omega(n)}, 2^{-\Omega(n)})$ *PRG* $G : \{0,1\}^n \to \{0,1\}^{l(n)}$ *such that* G *can be computed by circuits of size* $O(l(n))$ *with* $O(l(n)/n)$ *oracle calls to* f.

We describe the construction of an algorithm that will be proven to satisfy Theorem 3.7.

It has been shown in Section 2.3 that it is possible to specify a BLUO hash family (and also an API hash family) $h \in H_{m,\alpha m}$ by specifying a string from $\{0,1\}^{\mu m}$ for some constant μ. Let $f : \{0,1\}^n \to \{0,1\}^n$ be a β-exponential one-way function. Set $c_0 = \gamma$ and $c_1 = 1 - \gamma + \epsilon_2$ for some $\epsilon_2 > 0$ to be determined later. Construct an API hash family, $H_{n,c_0 n}$, and a BLUO hash family, $H_{n,c_1 n}$, which are represented as elements in $\{0,1\}^{k_0 n}$ and $\{0,1\}^{k_1 n}$ for some constants k_0 and k_1, respectively.

Construction 3.9. *Let* $H_{n,c_0 n}$ *be an API hash family and* $H_{n,c_1 n}$ *be a BLUO hash family with* h_0 *and* h_1 *drawn from* $H_{n,c_0 n}$ *and* $H_{n,c_1 n}$*, respectively. Let* $f : \{0,1\}^n \to \{0,1\}^n$ *satisfy Assumption 3.6. Then set:*

$$G(x, h_0, h_1) = (h_0(f(x)), h_0, h_1(x), h_1).$$

Note that $|(x, h_0, h_1)| = (1 + k_0 + k_1)n$ and $|G(x, h_0, h_1)| = c_0 n + k_0 n + c_1 n + k_1 n = (1 + \epsilon_2 + k_0 + k_1)n$, so G has linear stretch. G can also be computed by linear-sized circuits becauseh_0, h_1, and f can all be computed by linear-sized circuits.

Proof of Theorem 3.7. We must prove the $(2^{\Omega(n)}, 2^{-\Omega(n)})$ computational indistinguishability of $X = (H_{n,c_0n}, U_{c_0n}, H_{n,c_1n}, U_{c_1n})$, where each of the distributions are drawn from independently, from $Y = (H_{n,c_0n}, H_{n,c_0n}(f(U_n)), H_{n,c_1n}, H_{n,c_1n}(U_n))$, where the same function is drawn for each instance of H_{n,c_0n} and H_{n,c_1n}, respectively, and U_n yields the same element as input for both hash families (for simplicity, notation is abused and, for instance, H_{n,c_0n} is the random variable distributed uniformly over its functions). This will be accomplished using the hybrid distribution $Z = (H_{n,c_0n}H_{n,c_0n}(f(U_n)), H_{n,c_1n}, U_{c_1n})$.

We first prove that for a β-exponential one-way function f with $H_\infty(f(U_n)) > \gamma n = c_0 n$, the distribution X and Z are $(2^{\Omega(n)}, 2^{-\Omega(n)})$ computationally indistinguishable. Set $\epsilon < \epsilon_1$ where $H_\infty(f(U_n)) - \epsilon_1 = \gamma n$. Applying Lemma 3.5, we obtain that $\delta((H_{n,c_0n}, H_{n,c_0n}(U_n)), U_{(c_0+k_0)n}) < 3 \cdot 2^{-\epsilon n/4}$ and therefore $\delta(Y, Z) < 3 \cdot 2^{-\epsilon n/4}$. Since statistical indistinguishability is stronger than computational indistinguishability, Z and Y are $(t(n), 3 \cdot 2^{-\epsilon n/4})$ computationally indistinguishable for any $t(n)$.

We finish the proof by showing that Y and Z are $(2^{\Omega(n)}, 2^{-\Omega(n)})$ computationally indistinguishable. This claim is equivalent to the claim that H_{n,c_1n} is a (exponentially hard) hardcore family of f. Since $\beta > 2\theta(1-\gamma)$, c_1 can be fixed to be greater than $1-\gamma$. Then there exists a constant α with $2(c_1 + \alpha) < \beta$ and $c_0 + c_1 > 1$. Therefore, for appropriate c_1, Theorem 2.10 implies Y and Z are $(2^{\alpha n}, 2^{-\alpha n})$ computationally indistinguishable, which completes the proof.

3.4 Re-examining Previous Constructions

We have so far presented a construction of a PRG for exponentially hard one-way functions with certain preimage constraints. We now demonstrate that using API and BLUO hash families that can be computed by linear-sized circuits as inputs in previous constructions also yield linear-stretch PRGs that can be computed by linear-sized circuits. These constructions will use different assumptions than have so far been employed. We first recall the definition of Shannon entropy.

Definition 3.10. *For a random variable X, define the Shannon entropy of X as*

$H(X) = \mathrm{E}_{x \xleftarrow{R} X}[-\log \Pr[X = x]]$.

We now define a regular one-way function and also discuss the situation where the preimage size(s) of f is efficiently computable.

Definition 3.11. *A regular one-way function f is a one-way function $f : \{0,1\}^n \to \{0,1\}^m$, $|f^{-1}(f(x))| = |f^{-1}(f(y))|$ for all $x, y \in \{0,1\}^n$.*

Definition 3.12. *For $f : \{0,1\}^n \to \{0,1\}^m$, define $D_f(z) = \log |f^{-1}(z)|$. D_f can be thought of as a random variable distributed over $\{0,1\}^m$.*

Definition 3.13. *Let $f : \{0,1\}^n \to \{0,1\}^m$. f is a (t, ϵ) pseudoentropy generator with pseudoentropy s_n if there exists a random variable X distributed over $\{0,1\}^m$ such that $H(X) \geq n + s_n$ and X and $f(U_n)$ are (t, ϵ) computationally indistinguishable.*

3.4.1 PRGs from regular one-way functions with known preimage size

We will show that using the hardcore and hash families above that can be computed by linear-sized circuits, modifying a construction given by HILL [47] yields, for any exponentially hard one-way function f with D_f a polynomial-time computable ensemble, a linear-stretch PRG that can be computed by linear-sized circuits. An immediate corollary is a linear-stretch PRG that can be computed by linear-sized circuits for any regular one-way function with known preimage size.

Let $f : \{0,1\}^n \to \{0,1\}^n$ and let $h : \{0,1\}^n \to \{0,1\}^{n+2}$ be a BLUO hash function with description length p_n. Define $f'(x, y) = (f(x), h_y(x)_1, ..., h_y(x)_{D_f(f(x))+2}, y)$. The following is a simple adaptation of results from [47].

Lemma 3.14 (Lemma 5.1.2. in [47])**.** *Let f, f' be as above. If f is a $(2^{\Omega(n)}, 2^{-\Omega(n)})$ one-way function, f' is a $(2^{\Omega(n)}, 2^{-\Omega(n)})$ one-way function. Further, $H(f'(X, Y)) \geq n + p_n + 1/2$.*

Our main observation is as follows.

Lemma 3.15. *Let f, f' be as above. Let $H_{n,l(n)}$ be a BLUO hash family with $l(n) < \beta/2$. For $h \in H_{n,l(n)}$, define $g(x, y, h) = (f'(x, y), h, h(x))$. Then g is a $(2^{\Omega(n)}, 2^{-\Omega(n)})$ pseudoentropy generator with pseudoentropy $l(n) - 1/2$.*

Proof of Lemma 3.15. Because h is a hardcore function by Theorem 2.10, the the distribution of the output of g is $(2^{\Omega(n)}, 2^{-\Omega(n)})$ computationally indistinguishable from the output of $g'(x, y, h, \beta) = (f'(x, y), h, \beta)$, where $\beta \leftarrow U_{l(n)}$. By Lemma 3.14, the output of g has Shannon entropy $\geq |f'(x, y)| + |h| + l(n) - 1/2$, which completes the proof.

Using this observation, we obtain the following.

Corollary 3.16. *If $f : \{0,1\}^n \to \{0,1\}^n$ is a regular β exponential one-way function with polynomial time computable function ensemble D_f, then there exists a $(2^{\Omega(n)}, 2^{-\Omega(n)})$ PRG G with linear stretch that can be computed by linear size circuits.*

The corollary follows by letting all pairwise independent hash functions be API hash functions, and all hardcore functions be BLUO has functions, that can each be computed by linear-sized circuits as discussed in Section 2.3 and by Lemma 3.15 for an exponentially hard one-way function (since this implies pseudoentropy $\Theta(n)$).

3.4.2 PRGs from regular one-way functions with unknown preimage size

We modify a result of Haitner, Harnik and Reingold in [45, 42] to yield a linear stretch PRG that can be computed by linear-sized circuits from any regular exponentially-hard one-way function with unknown preimage size. Note that a very similar construction to [43, 45] exists in [39], but that construction requires n-wise independent hash families, whereas [43, 45] requires only pairwise-independent hash families.

One can slightly modify Theorem 5.2.2 in [42] to get the following modification of Proposition 2.10 above.

Proposition 3.17. *For some constants $\kappa > 0$ and $c \in (0,1)$ and for a polynomial p, there exists a family of functions $\{hc_i : \{0,1\}^{\kappa n} \to \{0,1\}^i\}_{i \in [n]}$ such that the following holds: Let $L_n \subseteq \{0,1\}^n$, let f and g be two polynomial-time computable functions $f, g : \{0,1\}^n \to \{0,1\}^n$, where in time at most $2^{\beta n}$ it holds that*

$$\Pr_{x \leftarrow L_n}[A(f(x)) = g(x)] \leq 2^{-\beta n}.$$

Then for every t' satisfying $p(n,t'(n)) < 2^{\beta n}$ and for every $l \in \{1,...,\lfloor c\beta n \rfloor\}$, the function $hc : \{0,1\}^{\kappa n} \to \{0,1\}^l$ defined as $hc(x,r) = hc_l(g(x),r)$ is a $(t'(n), 1/t'(n))$ hardcore function of $f'(x,r) = (f(x),r)$ over L_n. Furthermore, each hc_i can be computable by linear-sized circuits.

Remark 3.18. The prime difference between Proposition 3.17 and Proposition 2.10, namely the inclusion of the terms L_n and $g(x)$, are basic modifications of the Goldreich Levin theorem (see the proofs of the Golreich Levin theorem in [37] or Corollary 1 in [40]).

We now construct the following PRG, modified from Theorem 5.3.10 in [42].

Theorem 3.19. *Let $f : \{0,1\}^n \to \{0,1\}^n$ be a β exponential regular one-way function, and let $\mathcal{H}$ be an efficient family of pairwise-independent length preserving hash functions. Define $G : \{0,1\}^n \times \mathcal{H}^{d-1} \times \{0,1\}^{\kappa n} \to \{0,1\}^{dl} \times \mathcal{H}^{d-1} \times 0,1^{\kappa n}$ as*

$$G(x,\bar{h},r) = (hc_l(f^1(x,\bar{h}),r),...,hc_l(f^d(x,\bar{h}),r),\bar{h},r), \tag{2.2}$$

where:

- *$l = \lfloor \frac{c\beta}{4} \rfloor$ and $d = \lceil n/l \rceil + 1$, where c is the constant that appears in Proposition 3.17.*
- *hc_l is as in Proposition 3.17.*
- *$f^1(x,\bar{h}) = f(x)$ and for all $i \in [d-1]$ it holds that $f^{i+1}(x,\bar{h}) = f(\bar{h}_i(f^i(x,\bar{h})))$.*

There exists a polynomial p such that G is a $(t'(n),t'(n))$ PRG for any t' satisfying $p(n,t'(n)) < 2^{\beta n}$. The input length of G is $\Theta(n)$ and it stretches its input by $\Omega(n)$.

Using the API hash families as discussed in Section 2.3 which can be computed by linear-sized circuits and have description length $O(n)$, and using for the hardcore functions the family of BLUO hash functions as also discussed in Section 2.3 (as stated in Corollary 2.17) for Proposition 3.17, Theorem 3.19 then yields the following corollary.

Corollary 3.20. *If $f : \{0,1\}^n \to \{0,1\}^n$ is a β exponential regular one-way function (with possibly unknown preimage size), then there exists a $(2^{\Omega(n)}, 2^{-\Omega(n)})$ pseudorandom generator G with linear stretch that can be computed by linear-sized circuits with $O(1)$ oracle calls to f.*

4 PRGs for One-Way Functions with Somewhat Upper-Bounded Min-Entropy

The PRGs provided so far have depended on the fact that the choice of one-way function always satisfies Assumption 3.6. However, it might be the case that one can only generate "good" one-way functions with constant probability; for instance, random bipartite graphs will only be expander graphs, which are typically assumed for the Goldreich one-way function, with constant probability (see Proposition 6.6). We will construct a new family of PRGs based on the following slightly modified assumption:

Assumption 4.1. *There exists a collection of functions $\mathcal{F}$ such that all $f \in \mathcal{F}$ satisfy Assumption 3.6 except with probability at most 1/100 over the choice of f.*

Lemma 4.2. *If Assumption 4.1 holds then there exists a collection of functions, $\mathcal{G}_n$, where each $G \in \mathcal{G}_n$ is, except with probability $2^{-\Omega(\sqrt{n})}$, a $(2^{\Omega(\sqrt{n})}, 2^{-\Omega(\sqrt{n})})$ linear-stretch pseudorandom generator that can be computed by linear-sized circuits.*

Note that this collection of PRGs is not asymptotically optimal as security is not exponential in n. We obtain this collection by defining each PRG as follows.

Construction 4.3. *Denote the set*

$$\mathcal{G}_n = \{G_{f_1}...G_{f_{\sqrt{n}}} \mid f_1, ..., f_{\sqrt{n}} \in \mathcal{F}, G_{f_i} : \{0,1\}^{\sqrt{n}} \to \{0,1\}^{(1+c)\sqrt{n}}\}$$

for some $c > 0$, where each G_{f_i} is constructed using Construction 3.9 utilizing a different "one-way function" $f_i \in \mathcal{F}$. Then for each $G \in \mathcal{G}_n$, $G(x) = G_{f_1}(x_1)...G_{f_{\sqrt{n}}}(x_{\sqrt{n}})$ where $x = x_1...x_{\sqrt{n}}$ and $x_i \in \{0,1\}^{\sqrt{n}}$.

Let F be an $\frac{n}{50} - ERF$ from $(1+c)n$ bits to $(1+c')n$ bits, where $0 < c' < c$. . Then for $x \in \{0,1\}^n$, $x = x_1...x_{\sqrt{n}}$ where each x_i has length $\sqrt{n}$, set $G(x) = F(G_1(x_1), ..., G_{\sqrt{n}}(x_{\sqrt{n}}))$.

Since each of the G_{f_i}s can be computed by $O(\sqrt{n})$-sized circuits and F can be computed by linear-sized circuits, G can be computed by linear-sized circuits. Also, since each G_{f_i}

exhibits stretch that is linear in $\sqrt{n}$ and there are $O(\sqrt{n})$ of them, the output of G has linear stretch (namely by a factor of $1+c'$). Finally, we note that G_{f_i}'s constructed with an f_i that satisfies Assumption 4.1 are $(2^{\Omega(\sqrt{n})}, 2^{-\Omega(\sqrt{n})})$ PRGs.

Proof of Lemma 4.2. By a Chernoff bound, the number of output bits of G that due to a PRG G_{f_i} that is based on a bad $f \in \mathcal{F}$ is less than $98n/100$ except with probability $2^{-\Omega(\sqrt{n})}$. Since $l > n/100$, the output of G will be $(2^{\Omega(\sqrt{n})}, 2^{-\Omega(\sqrt{n})})$ pseudorandom by Definition 2.13.

Note that the constructions mentioned before Corollary 3.8 can be utlilized to modify the above construction to obtain a family of PRGs with arbitrarily large linear stretch.

5 Candidate one-way functions with high min-entropy

In the above section, it was demonstrated that if a one-way function existed satisfying Assumption 3.6 then a linear-stretch PRG could be constructed that could be computed by linear-sized circuits. While there are no proven exponentially hard one-way functions, there do exist one-way functions with conjectured exponential hardness. It has been shown [19] that a specific class of these functions have very high min-entropy, which would be very useful to satisfy Assumption 3.6. Here, we show that a more general class of one-way functions also has very high min-entropy (except with small probability).

5.1 Goldreich's One Way Function With One Predicate

Goldreich conjectured in [38] the following graph-based one-way function: Consider a d-ary predicate P and a bipartite graph $G = (V, E)$, here assumed to be bipartite with right degree d (denoted *d-regular*), n left vertices (nodes) and n right vertices. Define $f : \{0,1\}^n \to \{0,1\}^n$ by labeling the left (input) nodes of the bipartite graph with the bits of x. We define the jth output bit of f on x, $f(x)_j$, as $P(x_{i_1}, \ldots, x_{i_d})$, where $u_{i_1}, \ldots, u_{i_d}$ are the input nodes that are in the neighborhood of v_j, the jth output node of G. We note that f can be computed by linear-sized circuits as long as d is a constant. We will examine the min-entropy of two different variants of this function.

In [19], the following predicate is suggested:

$$P_d(x) = x_1 \oplus \ldots \oplus x_{d-2} \oplus (x_{d-1} \wedge x_d). \tag{2.3}$$

Lemma 5.1. *For every degree d, define P_d as in equation 2.3. Choose a random d-regular bipartite graph for f by connecting each output to d inputs chosen uniformly at random (with replacement). Then $E[|\{(x,y) : f(x) = f(y)\}|] = 2^{(1+2^{-\Omega(d)})n}$, where the expectation is over the choice of graph.*

Note that the expectation is bounded below by 2^n because it includes all instances (x, x).

Markov's inequality then yields that for any $\epsilon > 0$, the distribution $f(U_n)$ has min-entropy $> (1 - 2^{-\Omega(d)})n = \gamma n$ except with $2^{-\Omega(n)}$ probability over the choice of f.

The algorithm suggested in [19] is shown there to take exponential time to invert f (it is also needed that f is based on an expander graph). No other attacks exists in the literature for this function when utilizing this particular P_d for functions from n bits to n bits.

5.2 Goldreich's One Way Function With n Predicates

Instead of fixing a particular predicate P for each one-way function, one can instead choose a predicate P_i for each output node of the bipartite graph G. Let $\mathcal{F}$ denote the set of candidate Goldreich one-way functions from n bits to n bits consisting of a d-right regular bipartite graph on $2n$ vertices with n randomly chosen d-ary predicates P_i. Therefore, to select a $f \in \mathcal{F}$ is to select uniformly at random a d-right regular bipartite graph G and a tuple $(P_1, ..., P_n)$, where the P_is are chosen from the set of all functions from $\{0,1\}^d$ to $\{0,1\}$. We can then state the following result about the min-entropy of the proposed function. We note that Applebaum in [5] uses an assumption about the hardness of Goldreich's one-way function using a single random predicate to obtain a pseudoentropy result about the output of that one-way function; in contrast, we provide a statistical entropy result without any one-wayness assumptions, although our function would need to be one-way to be used in the PRGs outlined in this paper.

Theorem 5.2. *For $f : \{0,1\}^n \to \{0,1\}^n$ uniformly chosen at random from $\mathcal{F}$ consisting of a d-right regular bipartite graph on $2n$ vertices with n randomly chosen d-ary predicates P_i, $H_\infty(f(U_n)) > (1 - 2^{-\Omega(d)})n$ except with negligible probability over the choice of f.*

The theorem is proved by applying Markov's inequality to the following Lemma, using the reasoning discussed after Lemma 5.1.

Lemma 5.3. *For $f \in \mathcal{F}$ uniformly chosen at random and $\mathcal{F}$ defined as above. Then*

$$E_{f\in\mathcal{F}}[|\{(x,y)|f(x) = f(y)\}|] \leq 2^{(1+2^{-\Omega(d)})n}.$$

Proof of Lemma 5.3. This proof will largely follow a similar proof Theorem 4.1 in [19]. For $x, y \in \{0,1\}^n$, denote by n_{ij} the number of places where $x = i$ and $y = j$; then $n_{00} + n_{10} + n_{01} + n_{11} = n$. Let $\alpha_{ij} = n_{ij}/n$. Denote by $PE(\alpha_{00}, \alpha_{10}, \alpha_{01}, \alpha_{11})$ the probability over the choice of f that $f(x)_i = f(y)_i$. Note that since the choice is over the set of all bipartite graphs and the set of all n-tuples of d-ary predicates, the P_is are uncorrelated over the choices of x and y . Let $q = \alpha_{00} + \alpha_{11}$; then

$$PE = \frac{1 + q^d}{2}. \tag{2.4}$$

One then can derive the following, where H is the base-2 entropy.

$$\begin{aligned}
&E_{f\in\mathcal{F}}[|\{(x,y)|f(x) = f(x)\}|] \\
&= \sum_{x,y\in\{0,1\}^n} \Pr_{f\in\mathcal{F}}[f(x) = f(y)] \\
&= \sum_{n_{00}+n_{10}+n_{01}+n_{11}=n} \binom{n}{n_{00}, n_{10}, n_{01}, n_{11}} \Pr_{f\in\mathcal{F}}[f(x) = f(y)] \\
&\leq n^4 \max_{\alpha_{00}+\alpha_{10}+\alpha_{01}+\alpha_{11}=1} \binom{n}{n\alpha_{00}, n\alpha_{10}, n\alpha_{01}, n\alpha_{11}} PE(\alpha_{00}, \alpha_{10}, \alpha_{01}, \alpha_{11})^n \\
&= \max_{\alpha_{00}+\alpha_{10}+\alpha_{01}+\alpha_{11}=1} (2^{H(\alpha_{00},\alpha_{10},\alpha_{01},\alpha_{11}} PE(\alpha_{00}, \alpha_{10}, \alpha_{01}, \alpha_{11}))^{n(1+o(1)}
\end{aligned}$$

It therefore remains to show that there is a constant $\epsilon > 0$ such that for d large enough

$$\forall \alpha_{ij} : H(\alpha_{ij}) + \log_2 PE(\alpha_{ij}) \leq 1 + 2^{-\epsilon d}. \tag{2.5}$$

Let $p = 1 - q$. Having $\alpha_{00} = \alpha_{11} = q/2$, $\alpha_{10} = \alpha_{01} = p/2$ can only increase H, so it remains to show that

$$\forall p \in [0,1] : H(p/2, p/2, q/2, q/2) + \log_2(\frac{1 + (1-p)^d}{2}) \leq 1 + 2^{-\epsilon d}. \tag{2.6}$$

Equivalently, we want to demonstrate that

$$\forall p \in [0,1] : H(p) + \log_2(1 + (1-p)^d)) \leq 1 + 2^{-\epsilon d}. \tag{2.7}$$

The problem will be broken into four cases, choosing constants as needed.

Case 1: $p > \epsilon_1$

$$H(p) + \log_2(1 + (1-p)^d) \leq 1 + \log_2 e(1 - \epsilon_1)^d \leq 1 + 2^{-\epsilon d} \tag{2.8}$$

for $\epsilon < -\log_2(1 - \epsilon_1)$ and sufficiently large d.

For the remaining cases, since p is small one can employ the Taylor expansion of $\log_2$ around 2 to derive

$$\log_2(1 + (1-p)^d) \leq 1 + \frac{(1-p)^d - 1}{2 \ln 2} \leq \frac{e^{-pd} - 1}{2 \ln 2}. \tag{2.9}$$

Case 2: $\epsilon_1 \geq p > \epsilon_2/d$

$$H(p) + \log_2(1 + (1-p)^d) \leq H(\epsilon_1) + 1 + \frac{e^{-\epsilon_2} - 1}{2 \ln 2} \leq 1 \tag{2.10}$$

if ϵ_1 is chosen small enough so that $H(\epsilon_1) < \frac{1-e^{-\epsilon_2}}{2\ln 2}$.

Set $\epsilon_2 = 1/2$. Note then that $pd \leq 1/2$, and one can use the approximation

$$H(p) + 1 + \frac{e^{-pd} - 1}{2\ln 2} \leq (p\log_2(1/p) + 2p) - \frac{pd}{4\ln 2} = p(\log_2(1/p) - \Theta(d)) + 1. \qquad (2.11)$$

Case 3: $\epsilon_2/d \geq p > 2^{-\epsilon_3 d}$

For $\epsilon_3 < \frac{1}{4\ln 2}$ and sufficiently large d, $\log_2(1/p) + \Theta(d) < 0$.

Case 4: $2^{-\epsilon_3 d} \geq p$

For $\epsilon < \epsilon_3$ and sufficiently large d, $p\log(1/p) \leq \epsilon_3 d 2^{-\epsilon_3 d} \leq 2^{-\epsilon d}$.

Having exhausted all possible cases, the proof is complete.

6 The specific setting of [19]

In trying to meet part (iii) of Assumption 3.6 in the particular case that P_d is as in Equation 2.3, we have the following definitions and result from [19].

Let $G = (V, E)$ be a bipartite graph with $2n$ vertices and right-degree d. The left vertices (or nodes) are called input nodes, and we denote them by I. The right nodes are called output nodes, and we denote them by U.

Definition 6.1. *Let I be a set of input nodes. Its* ***boundary,*** *∂I, is the set of all $j \in U$ such that there is exactly one edge from j to I. The* ***neighborhood of*** *I, $\Gamma(I) \subseteq U$, is the set of all nodes connected to I.*

Definition 6.2. *G is an (r, d, c)-**boundary expander** if for all $I \subseteq V$, $|I| \leq r$ implies that $|\partial i| \geq c|I|$. G is an (r, d, c)-**expander** if for all $I \subseteq V$, $|I| \leq r$ implies that $|\Gamma(I)| \geq c|I|$.*

Proposition 6.3 (Lemma 2.10 in [19])**.** *Every (r, d, c)-expander is an $(r, d, 2c-d)$-boundary*

expander.

Definition 6.4. *$P : \{0,1\}^d \to \{0,1\}$ is (h,ϵ)-**balanced** if, after fixing all variables but $h+1$ of them, $|\Pr[P(x)=0] - \frac{1}{2}| \leq \epsilon$.*

Proposition 6.5. *Suppose G is a d-regular bipartite graph that is also an $n \times n$ (r,d,c)-boundary expander. Define $F = \lceil 2c - d - h \rceil - 1$ and $s = F/(F + d(d-1))$. With the definition of f on G as above with $P_d(x) = x_1 \oplus \ldots \oplus x_{d-2} \oplus (x_{d-1} \wedge x_d)$, there exists an algorithm A that chooses $x \in \{0,1\}^n$ uniformly, with $f(x) = b$, and solves $f(x) = b$ in time $2^{O(r(c-2))}$ with probability at most*

$$2^{n2^{-\Omega(d)}} 2^{-s\lfloor \frac{cr}{4dK} \rfloor}. \tag{2.12}$$

We assume for the moment that $r = \frac{n}{\alpha^{10}}$ for some absolute constant $\alpha > 1$ and $c = \Theta(d)$. Equation 2.12 together with part (iii) of Assumption 3.6 can therefore be formulated as

$$2^{-\Omega(d)} < \frac{1}{\Theta(d^2)} \tag{2.13}$$

which is true for d large enough. To show that our sample parameters r, and c are realistic we have the following result from [53].

Proposition 6.6. *Let $d \geq 3$. The probability that a randomly chosen d-regular bipartite graph on n vertices G is an $(\alpha^{-10}n, d, c')$ -expander graph with $c' \geq 9/10d - 1$ is $1 - o(1)$.*

This corresponds by Lemma 6.4 to G being an $(\alpha^{-10}n, d, 4/5d - 2)$ boundary expander graph. In particular, we can require n to be large enough so that the probability that a randomly chosen right-degree d bipartite graph on n vertices will be a $(\alpha^{-10}n, d, 4/5d - 2)$ boundary expander graph would be greater than $1/100$. This fits in with our notion of collections of PRGs.

7 PRGs with Arbitrary Stretch from PRGs with Linear Stretch

Here we examine in more details how to construct a PRG with polynomial stretch from a PRG with linear stretch, in particular, we consider the following construction (we follow the

reasoning of [37, 55]):

Construction 7.1. *Let G be a $(t(n), \epsilon(n))$ PRG with stretch $2n$ than can be computed by circuits of size cn. We can view the output of G as consisting of a left n-bit string concatenated with a right n-bit string. By reapplying G to each of these n-bit strings, we get a tree-like construction. Repeating through depth k yields an output of length $2^k n$; denote this function G_k. In particular, $G_{\log n^\alpha}$ has stretch n^α.*

We first verify that $G_{\log n^\alpha}$ can be computed by circuits with size $O(n^\alpha)$. Let $T(m)$ denote the circuit size of G with output size m (without loss of generality, we assume that $m = 2^i k$ for some i. By assumption, $T(2n) = cn$, and we further have that $T(2^k n) = T(2^{k-1} n) + 2^{k-1} cn$. This implies that $T(2^k n) = cn \sum_{i=1}^{k-1} 2^i < c(2^k)n$.

We now have the following lemma about the security of $G_{\log n^\alpha}$.

Lemma 7.2. *G is a $(t(n), \epsilon(n))$ PRG with stretch $2n$ that is computable by circuits of size cn, then $G_{\log n^\alpha}$ is a $(t(n) \cdot O(n^{-\alpha}), \epsilon(n) \cdot O(n))$ PRG with stretch n^α that is computable by circuits of size $O(n^\alpha)$.*

The proof of the lemma follows from the proof of Theorem 3.6.6 in [37]; in particular, the distinguisher D for $G_{\log n^\alpha}$ reduces to a distinguisher D' that executes G for each query of D and stores appropriate responses.

Finally, we note that if G' does not have stretch $2n$, but rather stretch an for some $a > 1$, we can iterate G' $\frac{1}{a-1}$ times to obtain a PRG G with stretch $2n$ and proceed as above.

8 Not All Pairwise Independent Hash Families are Hardcore

Kiltz [63] showed that if one-way permutations exist, then there exists a family of *universal* hash functions H and a one-way function f such that H is not a family of hardcore functions for f. Here we use a similar argument to show a similar result for *pairwise independent* hash functions.

Lemma 8.1. *Suppose a one-way permutation exists. Then there exists a pairwise independent hash family that is not hardcore.*

Proof. Let $H = \{h_1, ..., h_K : \{0,1\}^n \to \{0,1\}\}$ be a family of pairwise independent hash functions and let $f : \{0,1\}^n \to \{0,1\}^n$ be a one-way permutation. Consider the set of functions $S = \{h_1 \circ f, ..., h_K \circ f\}$.

We claim that S is a pairwise independent hash family. To prove the claim, let $x \neq y$ and consider the pair of distributions $(S(x), S(y))$. This pair equals the pair $(H(f(x)), H(f(y))$ and since f is a permutation, $x \neq y$ implies $f(x) \neq f(y)$. Therefore the pair of distributions $(S(x), S(y)) = (H(f(x)), H(f(y)))$ are distributed independently and uniformly at random over $\{0,1\}$ by definition of H being a pairwise independent hash family.

We define the one-way function $g(x, r) = (f(x), r)$ where $|r| = \log_2 K$. Then, define the hardcore bit $b(x, r) = h_r \circ f(x)$. Therefore, the question is whether the family of functions S is hardcore. However, given $g(x, r) = (f(x), r)$, a distinguisher D can simply compute $b(x, r) = h_r \circ f(x)$ from $g(x, r) = (f(x), r)$. □

CHAPTER 3

Nearly Simultaneously Resettable Black-Block Zero Knowledge

1 Introduction

In this work, we study the feasibility of achieving efficient zero-knowledge proof systems in the presence of physical attacks. Specifically, we examine the role of the black-box use of the code of the adversary with respect to simultaneously resettable proof systems. Such proof systems are of interest as examples of proof systems that are secure under very relaxed constraints on the re-use of the same randomness in multiple executions. In the case of *resettable zero knowledge (rZK)*, a malicious verifier may cheat against an honest prover who must use the same random tape polynomially many times. Further, *resettably sound* zero knowledge constrains the randomness used by the verifier: a malicious prover may try to cheat against an honest verifier who must use the same random tape polynomially many times. The former property was introduced and instantiated by Canetti, Goldreich, Goldwasser and Micali [16]; the later property was introduced by Micali and Reyzin [67] and later instantiated by Barak, Goldreich, Goldwasser and Lindell [9]. Because rZK is a generalization of concurrent zero knowledge (cZK) [30, 77, 62, 17, 76], every rZK proof system is a cZK proof system.

A question opened by [9] and resolved by [26] is *"Does there exist a resettably sound rZK proof system for all $\mathcal{NP}$?"* [26] answered this question in the affirmative, but they required a construction with a security proof that required a non-black-box simulator strategy, which utilize the specific strategy of a cheating verifier in its specification. Currently, there is

no known practical protocol that relies on a non-black-box[1] simulation strategy, while for instance there *do* exist efficient constructions for cZK and concurrent non-malleable zero knowledge that rely on black-box simulation strategies [68, 73], which work against any verifier strategy.

It is proved in [9] that resettably sound black-box zero-knowledge arguments can be constructed for languages in $\mathcal{BPP}$. Instead, we study whether there exist *t-bounded resettably sound rZK proof systems with black-box simulation*, and more generally, with only a black-box use of the code of the adversary (i.e., both the simulation and the proof of soundness do not rely on non-black-box uses of the code of the adverary). Such proof systems are rZK but also allow a malicious prover to conduct at most $t(n)$ resets against an honest verifier, where t is any fixed polynomial and n is the security parameter. Such a security setting has practical applications (indeed, in [12] it has been considered for the case of e-passports) because real hardware that may be reset to break security, such as smart cards or stateless devices, have certain wear costs; after enough resets, the hardware may simply break, a simple counter may run out, or a built-in battery may become depleted. Our black-box construction is also of theoretical interest, and moreover may lead to more efficient near-simultaneously rZK protocols. Indeed, while all known non-black-box constructions based on standard assumptions are inefficient, there are in several cases alternative efficient black-box constructions [68, 73]. Further, unlike [9], we obtain unconditional soundness, a property that is hard to achieve when the simulator is non-black-box.

We remark that constructing t-resettably sound rZK proof systems for any language $L \in \mathcal{NP}$ with black-box simulation is quickly accomplished if round complexity is allowed to be t-dependent. For any t, take any rZK proof system with black-box simulation and repeat it sequentially with independent randomness $t + 1$ times; a verifier then accepts only if he accepts for each of the $t + 1$ protocol runs. What we desire is to construct a t-resettably sound rZK proof system where the round complexity is *t-independent*.

[1] We ignore controversial non-black-box extraction assumptions.

1.1 Overview of Our Contribution

For all $\mathcal{NP}$ and for any polynomial t, we construct a t-resettably sound rZK proof system with black-box use only of the code of the adversary and round complexity $O(n^\epsilon)$ for security parameter n and for any constant $\epsilon > 0$. We require standard assumptions as the existence of enhanced trapdoor permutations and collision-resistant hash functions.

We re-examinine the rZK protocol of [16]. Their protocol involves an adaptation of the cZK construction of [77]. We first give a high-level description of the protocol of [16]. V commits to a set of n^ϵ strings of length n. Then, in the next $2n^\epsilon$ rounds, P first commits to an n-bit string and V subsequently decommits to the next string, eventually decommitting to the entire set. The protocol concludes by P giving a rWI proof that either $x \in L$ or that at least one of the strings committed by P is identical to the subsequent decommitment of V. Clearly, such a protocol is not t-resettably sound (for any $t \geq 1$), as a resetting prover can simply obtain one of V's committed strings, rewind a round and then commit to that string. However, we use this protocol as a basis to construct our protocol.

We think of the initial commitment by V to n^ϵ strings of length n as a *database*. The idea for our protocol is that V should commit to a database of $poly(n, t)$ strings of length n; then, in each of the next $2n^\epsilon$ rounds, P asks for n entries of V's database, which V then reveals. Finally, P provides a t-resettably sound, rWI proof that either $x \in L$ or that P can commit to a large constant fraction of V's database. The idea is that even if P was able to successfully ask for $tn^{\epsilon+1}$ indices of the database, P would still not know a large constant fraction of the database; in this way, the protocol will be t-resettably sound.

We overcome several challenges to accomplish such a protocol. First, we require the simulator to discover significantly more indices of V's database than a t-resetting P^* possibly could. We note that we can modify the (black-box) simulator strategy given in [16]; there, at each prover commitment phase, the simulator executes an independent look-ahead subprotocol call to t discover the string that V would decommit to. In fact, these look-ahead subprotocol calls are independent from one round to the next. We take advantage of this independence by having our simulator execute *polynomially* many look-ahead subprotocol calls

for each round and proving that such a strategy produces more than half of V's database. On the other hand, we will show that for suitable parameters, a t-resetting P^* will only be able to discover at most $1/16$ of V's database except with negligible probability. Therefore, our protocol starts by V committing to a large database followed by $2n^\epsilon$ rounds where V decommits to the n (distinct) random indices in each round that P asks for. Finally, P commits to a guess of the *entire* database and proves that either $x \in L$ or that a large constant fraction of the guess correctly corresponds to V's database.

However to have a meaningful statement for the proof given by P, it seems that V should reveal the entire database, but this exposes again V to reset attacks. Therefore, a second challenge is that V will reveal a small fraction of the database and P will prove that it committed to a large portion of this fraction. The challenge of such a strategy is that a cheating V^* might skew the distribution of what it reveals to be used for the rWI proof at the conclusion of our protocol such that the simulator might not discover enough entries of the database. Therefore, we require a special coin-flipping protocol by which V reveals n uniformly random elements of its database, at which point P proves, using a t-resettably sound rWI proof, that either $x \in L$ or that P's database guess, committed to before the final proof, contains a correct guess for at least $1/4$ of the n indices that V last decommitted to. Since the n revealed indices are uniformly distributed over V's database, only if P's initial database guess matches at least $1/4$ of V's database will P be able to give a correct WI proof without using the witness for $x \in L$ (except with negligible probability).

A first attempt for such a coin-flipping protocol between left player P_L and right player P_R might be for P_L to a apply pseudorandom function family, f_*^{PRF}, with a previously committed seed, s, on input a random string sent by P_R. The string sent by P_R would be constructed by applying a $(t+1)$-wise independent hash function on the transcript thus far (P_R cannot use a pseudorandom function because P_L^* is unbounded and could distinguish the output). However, a cheating P_L^* might commit to a seed in such a way that even on random input, the output of the pseudorandom function has skewed distribution. One can also try to modify the protocol by having P_R hash and output the pair (R, R') and send it

to P_L, who then computes $f_s^{PRF}(R) \oplus R'$. However, cheating P_R^* could then simply rewind, keep R fixed and send whatever R' he wished. Instead, we solve our problem as follows: P_R hashes to obtain the triple (R, R', r'), computes a (statistically hiding) commitment to R', denoted c, using randomness r', and sends (R, c) to P_L. Then, using previous committed seed s, P_L applies f_s^{PRF} to the concatenation of R and c, which also binds the output of the pseudorandom function to R' before R' has been revealed. Finally, P_R opens the decommitment of R' to P_L, and both set the random string τ to be the sum of the output of the pseudorandom function and R'. We remark that an adversary (resetting or not) may always guess $O(\log n)$ bits of the coin-flipping output; however, since the output length is n, an adversary will have only a negligible chance of correctly guessing a constant fraction of the coin-flipping output.

A final note is that while P_R^* may construct R and R' as he wishes, it is very important that a cheating P_L^* formats his pseudorandom function outputs correctly; otherwise, upon discovering P_R's R and R' , P_L^* could simply rewind and lie about the output of f_s^{PRF}. Therefore, P_L must send a rWI proof that either $x \in L$ or the function output was formatted correctly. In this way, only in the case that P_L^* is cheating with $x \notin L$ the correct formatting will need to be assured; we can bootstrap the witness for $x \in L$ to make the rWI proof of consistency witness hiding.

A third challenge is that our coin-flipping protocol makes the larger protocol not admissible. In particular, the simulator in [16] was given in the so-called hybrid model, where a cheating V^* is somewhat constrained. Therefore to prove rZK for our protocol, we must demonstrate that our protocol is not meaningfully different enough from the protocol of [16] even though their simulator no longer applies to our setting. To accomplish this task, we prove a theorem based on the observation that the only place where the simulators might differ are where they execute identically to the honest prover against the verifier but using a different witness. We therefore construct a variant of our own protocol that more explicitly models the protocol of [16] but is only rZK (and *not* bounded resettably sound). We then construct the simulator for the intermediate protocol. Finally, we prove that the existence of

a simulator for this intermediate protocol implies the existence of a simulator for the protocol that we desire. We believe that such a proof strategy is of independent interest.

We note that our protocol has communication complexity that is dependent of t; finding a protocol using standard assumptions with communication complexity independent of t is an interesting open question[2], as is improving the round complexity of our construction.

1.2 Other Related Work

The notion of rZK was first introduced by [16]; later constructions of black-box rZK protocols have better round complexity. In [62], a rZK proof system with black-box simulation and $\omega(\log^2 n)$ rounds is constructed. The protocol improved the round complexity of [77], by examining a static simulator rewind schedule and showing that such a schedule produced a successful single extraction, except with negligible probability. It is not clear how to adapt such a scheduling strategy to the polynomial many successful extractions that we require. The results of [76] can also be used to construct a black-box rZK proof system. Their protocol requires $\tilde{O}(\log n)$ rounds and also build upon the protocol of [77]. The simulator strategy of [76] relies on a careful analysis of the random tapes used by the simulator throughout its run together with the oblivious simulator strategy of [62] to obtain a single successful extraction, while our approach relies on segmenting the simulator of [16] and running various of its subprotocols in parallel to obtain polynomially many successful look-aheads. Finding compatibility between the two approaches is an interesting open question.

The first resettably sound (non-black-box) zero-knowledge argument was constructed by [9]. Deng and Lin [27] constructed a zero-knowledge argument secure in a weakened notion of simultaneous resettability: both cheating prover and verifier can reset the other polynomially many times, but can only reset a particular party with a fixed random tape (e.g., an incarnation) a bounded number of times. Their protocol requires only a constant number of rounds and also required non-black-box simulation in the proof.

The construction of a simultaneously rZK argument was first provided by [26] and requires

[2]Using less standard assumptions like complexity leveraging, constructing a protocol t-independent communication complexity seems to be more easily accomplished.

polynomial round complexity. Their protocol relies on the prover initially committing to his challenges for the extraction stage and using the resettably sound zero-knowledge argument of [9] to prove that either $x \in L$ or that the decommitted challenges are correct. Because the protocol heavily relies on the non-black-box zero-knowledge argument of [9], the simulator used for the security proof is non-black-box. Recently, it has been shown in [18] how to obtain a constant-round resettably sound resettable witness indistinguishable argument of knowledge.

We note that all the protocols listed here are in the standard model. In particular, [16, 9, 27, 22, 23, 78] also provide constructions in the bare public-key model. In addition, [34] give a resettable black-box *statistical* zero-knowledge proof for several non-trivial languages, but they do not have a construction for all $\mathcal{NP}$. As such research is incomparable to our work, we do not consider it here.

2 Preliminaries, Definitions and Tools

We denote by n the security parameter throughout our discussion., by $[m]$ the set $\{1, ..., m\}$, and by $x|y$ the concatenation of x and y, by U_n the uniform distribution over $\{0,1\}^n$.

Definition 2.1. *Two random variables* $\{X_n\}_{n\in\mathbb{N}}$ *and* $\{Y_n\}_{n\in\mathbb{N}}$ *distributed over* $\{0,1\}^{poly(n)}$ *are* computationally indistinguishable *if for all polynomials* q *and* p*, for all sufficiently large* n*, for every distinguishing circuit* D *of size* $p(n)$*,*

$$|\Pr_{x\leftarrow X_n}[D(x)=1] - \Pr_{y\leftarrow Y_n}[D(y)=1]| < \frac{1}{q(n)}.$$

A *perfectly binding non-interactive commitment scheme* consists of a pair of protocols CHCom and CHVer. The commitment stage of this protocol consists of C computing, for a message m and randomness r, $Com_{CH} \leftarrow \mathsf{CHCom}(m, r)$ and sending Com_{CH} to R. The decommitment stage of this protocol consists of C sending (m, r) to R who computes $\{0,1\} \leftarrow \mathsf{CHVer}(Com_{CH}, m, r)$ and accepts if 1 is output and rejects otherwise.

Definition 2.2. *A* perfectly binding non-interactive commitment scheme *for a PPT pair*

(C, R) *running* CHCom *and* CHVer *as described above satisfies:*

Correctness: *For all messages m and random strings r, if $Com_{CH} \leftarrow \mathsf{CHCom}(m, r)$ then $\mathsf{CHVer}(Com_{CH}, m, r) = 1$.*

Computationally Hiding: *For every two polynomials p, q, for all sufficiently large n, and for all $m, m' \in \{0,1\}^{p(n)}$ and every distinguishing circuit D of size $q(n)$,*

$$|\Pr_{r \leftarrow U_n}[D(\mathsf{CHCom}(m, r) = 1] - \Pr_{r' \leftarrow U_n}[D(\mathsf{CHCom}(m', r')) = 1]| < \frac{1}{q(n)}.$$

Perfectly Binding: *For every pair of messages m, m' and for any α, if both*

$$\Pr_{r \leftarrow U_n}[\mathsf{CHCom}(m, r) = \alpha], \quad \Pr_{r' \leftarrow U_n}[\mathsf{CHCom}(m', r') = \alpha]$$

are positive, then $m = m'$.

A *statistically hiding 2-round commitment scheme* consists of three protocols, $\mathsf{SHSetup}$, SHCom, and SHVer. The first round of the protocol consists of R computing, for randomness r, $SH_R \leftarrow \mathsf{SHSetup}(r)$ and sending SH_R to C. We will assume that SH_R is *public coin*; that is, SH_R is a string chosen uniformly at random over the set $\{0,1\}^{poly(n)}$. For message m and randomness r, C then computes $Com_{SH} \leftarrow \mathsf{SHCom}(m, r, SH_R)$ and sends Com_{SH} to R. To decommit to m, C sends (m, r) to R. To verify the decommitment, R computes $\{0,1\} \leftarrow \mathsf{SHVer}(m, r, SH_R, Com_{SH})$, accepts if 1 is output and rejects otherwise.

Definition 2.3. *A* statistically hiding 2-round commitment scheme *for a PPT pair (C, R) running* $\mathsf{SHSetup}$, SHCom *and* SHVer *as described above satisfies:*

Completeness: *For every $SH_R \leftarrow \mathsf{SHSetup}(r')$ for some random input r', every message m and any random string r, if $Com_{SH} \leftarrow \mathsf{SHCom}(m, r, SH_R)$ then $\mathsf{SHVer}(m, r, SH_R, Com_{SH}) = 1$.*

Statistically Hiding: *For all sufficiently large n, all SH_R, for all messages $m, m' \in$*

$\{0,1\}^{p(n)}$ *and for every polynomial* $q(n)$,

$$\sum_{\alpha} |\Pr_{r \leftarrow U_n}[\mathsf{SHCom}(m,r,SH_R)=\alpha] - \Pr_{r' \leftarrow U_n}[\mathsf{SHCom}(m',r',SH_R))=\alpha]| \leq \frac{1}{q(n)}.$$

Computationally Binding: *If* $Com_{SH} \leftarrow \mathsf{SHCom}(m,r,SH_R)$*, a cheating committer* C^* *tries, to find* m' *and* r'*, with* $m' \neq m$*, such that* $1 = \mathsf{SHVer}(m',r',SH_R,Com_{SH})$*. We require that for all polynomials* p *and* q*, for every circuit* C^* *(representing a cheating committer) of size* $p(n)$*, and all sufficiently large* n*,*

$$\Pr_{r \leftarrow U_n}[m' \neq m \wedge (\mathsf{SHVer}(m,r,SH_R,Com_{SH}) = 1 = \mathsf{SHVer}(m',C^*(m,r,SH_R),SH_R,Com_{SH}))] < \frac{1}{q(n)}$$

Definition 2.4. *Let* $l : \mathbb{N} \to \mathbb{N}$*. Consider a set of functions* $f^{PRF} = \{f_s^{PRF} : \{0,1\}^{l(|s|)} \to \{0,1\}^{l(|s|)}\}_{s \in \{0,1\}^*}$*. We then define an ensemble of random variables* $F = \{F_n\}_{n \in \mathbb{N}}$ *such that* F_n *is distributed uniformly over the multi-set* $\{f_s^{PRF}\}_{s \in \{0,1\}^n}$*. We say that* f^{PRF} *is a* pseudorandom function family *if the following holds:*

(a) There exists a polynomial time algorithm that on input s *and* $x \in \{0,1\}^{l(|s|)}$ *returns* $f_s^{PRF}(x)$*.*

(b) Let $G = \{G_n\}_{n \in \mathbb{N}}$*, where* G_n *is the random variable uniformly distributed over the set of all functions from* $l(n)$ *bits to* $l(n)$ *bits. Then for every polynomials* p *and* q*, for every distinguishing circuit* D *of size* $p(n)$ *given oracle access*[3] *to* F_n *and* G_n*, denoted* D^{F_n} *and* D^{G_n}*, respectively, and for all sufficiently large* n*,*

$$|\Pr_{r \leftarrow U_n}[D^{F_n}(r)=1] - \Pr_{r' \leftarrow U_n}[D^{G_n}(r')=1]| < \frac{1}{q(n)}.$$

Definition 2.5. *Let* $H_{n,m} = \{h_i : \{0,1\}^n \to \{0,1\}^m\}$*. We call* $H_{n,m}$ *a* family of k-wise independent hash functions *if it satisfies the following conditions:*

(a) For any distinct $x_1, ..., x_k \in \{0,1\}^n$*, the random variables* $H_{n,m}(x_1), ..., H_{n,m}(x_k)$ *are*

[3]We say that a distinguisher has oracle access to a distribution X_n if D, on input r, interacts with X_n by sending queries q and receiving back responses $X_n(q)$. This process continues iteratively per the specification of D. We note that D does not know the identity of X_n but rather only sees X_n's responses to the various queries q.

independent and uniformly distributed over $\{0,1\}^m$. *In other words, for every distinct* $x_1, ..., x_k \in \{0,1\}^n$ *and for every* $y_1, ..., y_k \in \{0,1\}^m$, $\Pr_{h \in H_{n,m}}[h(x_1) = y_1 \wedge ... \wedge h(x_k) = y_k] = 2^{-mk}$.

(b) There exists a polynomial-time algorithm that on input i *and* $x \in \{0,1\}^n$ *outputs the corresponding* $h_i(x)$.

2.1 Zero-Knowledge Proof Systems and Resettability

Definition 2.6. *For a pair of PPT algorithms* P_1 *and* P_2, *the interaction of* P_1 *using input* x *with* P_2 *using input* y, *where (without loss of generality)* P_1 *sends a message,* P_2 *responds, and so forth, and each algorithm can use previous message to compute its next message, is called an* interactive protocol *and denoted by* $(P_1(x), P_2(y))$.

We denote by $P_1(x) \leftrightarrow P_2(y)$ the transcript of public messages sent between P_1 and P_2 for the interactive protocol $(P_1(x), P_2(y))$. For an interactive protocol $\pi = (P_1(x), P_2(y))$, We denote by $View_{P_1}(\pi)$ as P_1's view (on input x) of the interaction with P_2 (on input y), which includes the transcript $P_1(x) \leftrightarrow P_2(y)$ as well as the input and auxiliary input of P_1, P_1's random tape, and the set of outputs that P_1 privately computes. We define $View_{P_2}(\pi)$ similarly.

We say that $(P_1(x), P_2(y))$ *accepts* if P_2 does not abort at any point in the execution (if P_1 sends the last message, then P_2 is able to abort after this message is sent). Otherwise, we say that $(P_1(x), P_2(y))$ *rejects*.

Definition 2.7. *Let* π_1,..., π_l *be a sequence of* l *interactive protocols; that is,* π_1,..., π_l *is a sequence of* l *pairs of PPT algorithms* $\{P_i^1(x_i), P_i^2(y_i)\}_{1 \leq i \leq l}$. *Such a sequence is called a* sequential composition of interactive protocols *when each protocol is run by the respective parties using the following triple of inputs:*

- P_1^1 *is given input* x *and* P_1^2 *is given input* y.
- *For* $2 \leq i \leq l$, *each* P_i^1 *is given as input the output of* P_{i-1}^1, *and each* P_i^2 *is given as input the output of* P_{i-1}^2.

Such a sequential composition will be written as $(\pi_1, ..., \pi_l)$. *We say that the sequential*

composition of interactive protocols $(\pi_1, ..., \pi_l)$ *is a pair of PPT algorithms* (P_1, P_2) *if* P_1 *on input* x *specifies subprotocols* $P_1^1, ..., P_l^1$ *and* P_2 *on input* y *specifies subprotocols* $P_1^2, ..., P_l^2$ *that, considered as respective pairs, execute the sequential composition of interactive protocols as described above.*

For an NP language L let R_L be the witness-relation consisting of the set all pairs (x, w) such that $x \in L$ and w is a witness for x in L. We say that $w \in W_L(x)$ if $R_L(x, w) = 1$.

Definition 2.8. *A pair of algorithms* (P, V) *for a language* L *with witness relation* R_L *is an* interactive proof system *if:*

Completeness: *For all* $x \in L$ *and all* $w \in W_L(x)$*, the probability that* $(P(x, w), V(x))$ *rejects is negligible in the length of* x*.*

Soundness: *If* $x \notin L$*, then for any prover* P^**, the probability that* $(P^*(x), V(x))$ *accepts is negligible in the length of* x*.*

We adapt the definition of resettable zero knowledge due to [16] to incorporate the fact that a black-box simulator needs a bound on sessions (for further discussion, see [17, 76]).

Definition 2.9. *An interactive proof system* (P, V) *for a language* L *is said to be* resettable zero-knowledge (rZK) *if for every polynomial* δ *there exists a PPT simulator* M_δ^* *such that for every PPT adversary* V^**, the distribution ensembles* D_1 *and* D_2 *described below are computationally indistinguishable: Let each distribution be indexed by a sequence of distinct common inputs* $\bar{x} = x_1, ..., x_{poly(n)} \in L \cap \{0,1\}^n$ *and a corresponding sequence of* P*'s auxiliary inputs* $\bar{y} = y_1, ..., y_{poly(n)}$*.*

Distribution D_1 *is defined by the following random process which depends on* P *and* V^**:*

(a) Randomly select and fix $t = poly(n)$ *random tapes* $\omega_1, ..., \omega_t$ *for* P*, resulting in deterministic strategies* $P^{(i,j)} = P_{x_i, y_i, \omega_j}$ *defined by* $P(\alpha) = P(x_i, y_i, \omega_j, \alpha)$ *for* $i, j \in \{1, ..., t\}$*. Each* $P^{(i,j)}$ *is called an* incarnation *of* P*.*

(b) Machine V^* *is allowed to run* $\delta(n)$ *many sessions with the* $P^{(i,j)}$*'s. Throughout these sessions,* V^* *is required to complete its current interaction with the current copy of*

$P^{(i,j)}$ before starting a new interaction with any $P^{(i',j')}$, regardless if $(i,j) = (i',j')$ or not. Thus, the activity of V^* proceeds in rounds. In each round, V^* selects one of the $P^{(i,j)}$'s and conducts a complete interaction with it.

(c) Once V^* decides it is done interacting with the $P^{(i,j)}$'s, V^* produces an output depending on its view of these interactions. This output is denoted by $(P(\bar{x},\bar{y}), V^*(\bar{x}))$ and is the output of the distribution D_1.

Distribution D_2 is the output of $M_\delta^{V^*}(\bar{x})$.

Such a proof system is black-box resettable zero-knowledge *if for every polynomial δ there exists a PPT M_δ so that $M_\delta^{V^*}$ with the above properties can be implemented by letting M_δ have oracle-access to V^*.*

In what follows, all references to rZK are for black-box rZK.

Definition 2.9 is in the sequential model, where V^* must complete its interaction with an incarnation before moving on to its next interaction (though V^* can move back to that same incarnation at will). [16] showed that rZK in the sequential model is equivalent to rZK in the interleaved model, where V^* does not have to complete its interaction with one incarnation but rather can interleave the various interactions.

Definition 2.10 ([16])**.** *An interactive proof system (P,V) for L and witness relation R_L is said to be* resettable witness indistinguishable (rWI) *if the following holds: Fix $\bar{x} = x_1, ..., x_{poly(n)} \in L \cap \{0,1\}^n$, and let $aux^{(1)}(\bar{x}) = y_1^{(1)}, ..., y_{poly(n)}^{(1)}$ and $aux^{(2)}(\bar{x}) = y_1^{(2)}, ..., y_{poly(n)}^{(2)}$, where $y_j^{(i)} \in W_L(x_j)$ for $1 \leq j \leq poly(n)$, be potentially different sequences of prover's auxiliary inputs. Then for any PPT verifier strategy[4] V^*, $\{(P(\bar{x}, aux^{(1)}(\bar{x})), V^*(\bar{x}))\}_{\bar{x}}$ and $\{(P(\bar{x}, aux^{(2)}(\bar{x})), V^*(\bar{x}))\}_{\bar{x}}$ are computationally indistinguishable.*

The following definition weakens the notion of resettable soundness is originally due to [9]. In fact, it was shown in [9] there that if there is a resettably sound black-box zero-knowledge proof system for a language L, then it must be that $L \in \mathcal{BPP}$. By contrast,

[4]Here, as in Definition 2.9, for each verifier strategy V^* we also assume the existence of a bound $\delta(n)$ of resets that V^* can execute. We do not bound δ, but its existence is needed for proving rWI below.

we will demonstrate the existence of a t-bounded resettably sound[5] black-box resettable zero-knowledge proof system for any $\mathcal{NP}$ language L.

Definition 2.11. *A* t-resetting attack *of a cheating prover P^* on a* t-resettable verifier V *is defined by the following two-step random process, indexed by a security parameter n.*

(a) Uniformly select and fix $l = poly(n)$ random tapes, denoted $r_1, ..., r_l$ for V, resulting in deterministic strategies $V^{(j)}(x) = V_{x,r_j}$ defined by $V_{x,r_j}(\alpha) = V(x, r_j, \alpha)$ where $x \in \{0,1\}^n$ and $j \in [l]$. Each $V^{(j)}(x)$ is called an incarnation *of V.*

(b) On input 1^n, machine P^* is allowed to initiate $poly(n)$ many interactions with the other $V^{(j)}(x)$'s. P^* chooses $j \in [l]$ and $x_j \in \{0,1\}^n$, thus defining $V^{(j)}(x_j)$ with random tape r_j. During P^*'s interaction with any $V^{(j)}(x_j)$, P^* may choose to interact with any other $V^{(j')}(x_{j'})$ during his interaction with $V^{(j)}(x_j)$ in an interleaved fashion. A t-complete session *with $V^{(j)}(x_j)$ means that for all pairs (j,t) specifying $V^{(j)}(x_j)$, P^* may only reset $V^{(j)}(x_j)$ at most t times over all its interactions. In other words, while P^* may have multiple sessions with a particular incarnation $V^{(j)}(x_j)$ over the course of the t-resetting attack, the total number of resets over all such sessions must be at most t.*

Let (P, V) be an interactive proof system where V is a PPT algorithm. We say that (P, V) is t-resettably sound *for L if the following two conditions hold:*

t-Resettable Completeness *Consider an arbitrary t-resetting attack and suppose that in some session after selecting an incarnation $V^{(j)}(x)$ the attacker follows the strategy P. Then, if $x \in L$, $V^{(j)}(x)$ accepts except with negligible probability.*

t-Resettable Soundness *For every t-resetting attack, the probability that in some session the corresponding $V^{(j)}(x)$ has accepted and $x \notin L$ is negligible.*

[5]In discussing how V^* interacts with P, Definition 2.11 is in the interleaved model while Definition 2.9 is in the sequential model. For resettable soundness, [9] showed that completeness and soundness in the interleaved model are equivalent in the sequential model (where sequential and interleaved prover interaction for resettable soundness are defined similarly as for resettable zero-knowledge). However, as pointed out in [16], doing so requires more calls to a given interaction in the sequential model than in the interleaved model and therefore it a stronger statement to discuss t-resettable soundness in the interleaved model.

Let (P, V) be an interactive proof system where V is a PPT algorithm. We say that (P, V) is resettably sound *for L if it is t-resettably sound for all $t \in poly(n)$.*

We will utilize the following construction[6] of Dwork and Naor [29].

Theorem 2.12 (zaps)**.** *If enhanced trapdoor permutations exist, then for every language L there exists a two-round simultaneously resettable WI proof system.*

2.1.1 Admissibility and the Hybrid Model

We introduce definitions used by [16] and many later works to construct rZK protocols (and adapted by [9] for resettably sound ZK protocols); we do not require the somewhat generalized notions of admissibility used by [26]. Admissible proof systems are protocols where the verifier's actions are essentially determined by its initial messages; typically, the later messages only consist of decommitments or other messages where the verifier has a fixed response that cannot impact the prover's subsequent messages. While the protocol that we ultimately construct will *not* be admissible in order to satisfy the bounded resettable soundness property, we will adapt our protocol from an admissible protocol (which has no resettable soundness properties).

Definition 2.13. *A proof system (P, V) is called* admissible *if the following requirements hold:*

(a) The prover P consists of two modules P_1 and P_2. Similarly, the random input w is partitioned into two disjoint parts $w^{(1)}$ and $w^{(2)}$, where $w^{(i)}$ is given to P_i. The prover initialization message is sent by P_1.

(b) Each verifier message (other than the first one) is first received by P_1 and is interpreted as consisting of two parts, called main *and* authenticator*. P_1 decides whether to accept the message or abort. If P_1 accepts, it forwards the* main *part of the message to P_2, who generates the next prover message.*

[6] In fact, zaps are not inherently resettable WI, though they are resettably sound, as noted by [9]. However, when the prover's zap message is computed using a random tape that is a pseudorandom function applied to V's initial message and P's random tape, as is done here, zaps are rWI. We will therefore refer to zaps here and implicitly assume that their instantiation in our protocol constructions utilize the appropriate PRF-random tape construction.

(c) Let V^* be an arbitrary (deterministic) polynomial-size circuit representing a possible strategy for the verifier in the interactive proof system (P, V). Then, except with negligible probability, V^* is unable to generate two different messages for some round l that specify the same session determining message in their corresponding prefixes, and such that P_1 accepts both.

Admissible proof systems are often proved secure in the so-called hybrid model. Essentially, the hybrid model restricts the verifier to only interacting with an incarnation that possesses a fixed random string $w^{(2)}$ once instead of polynomially many times. For further discussion, see [16].

Definition 2.14. *A* hybrid cheating verifier V^* *works against admissible proof systems defined in Definition 2.13 in the following manner. V^* proceeds as in Distribution D_1 of Definition 2.10 with the exception that no two sessions started by V^* may interact with incarnations $P^{(i,j,k)}$ and $P^{(i',j',k')}$ such that $k = k'$. An admissible proof system is hZK (respectively, hWI) if it satisfies Definition 2.10 with respect to hybrid cheating verifiers.*

Lemma 2.15 (Corollary 9 in [16])**.** *If a proof system (P, V) is hZK (respectively, hWI) for a language L, then, assuming the existence of pseudorandom functions, a new proof system $(\boldsymbol{P}, \boldsymbol{V})$ can be constructed to be rZK (respectively, rWI) for L as follows: $\mathbf{V}$ is identical to V; $\mathbf{P}$ uses the same random tape $w^{(1)}$ as P, and therefore $\mathbf{P}_1$ is identical to P_1; $\mathbf{P}_2$ acts as P_2 except that he computes and uses random tape $w^{(2)} = f_s^{PRF}(x, msg)$, where f_*^{PRF} is a pseudorandom function chosen from the pseudorandom function ensemble f^{PRF} using P's random tape, $x \in L$ is the statement being proved and msg is the first message(s) sent by V. Before $\mathbf{P}$ receives the determining message msg from $\mathbf{V}$, $\mathbf{P}_2$ acts exactly as P_2 does (with identical random tapes).*

3 Black-Box rZK with t-Resettable Soundness

Before giving the exact protocol specification for our candidate construction $\Pi = (P, V)$, we first outline its crucial steps. We consider our protocol as the composition of three

subprotocols, π_0, π_1, and π_2, for two reasons. The first reason is that the purpose of each of the subprotocols is distinct and so discussing them separately is natural. The second reason is that in order to prove rZK of Π, we will construct another protocol that will rely on the first two subprotocols but will require a different third subprotocol, π_2'. In what follows, fix a language $L \in \mathcal{NP}$, let n be the security parameter, let $\epsilon > 0$ be any constant, and let t be the polynomial resetting bound of the prover.

π_0 :	*Setup Phase* 1) P sets up SHCom 2) V constructs DB, $\lvert DB\rvert = 16tn(n^\epsilon + 1)$ 3) V sets up zap, V sends $\mathsf{SHCom}(DB)$
π_1 :	*Extraction Phase* 1) n^ϵ Iterations: i) P asks V for n indices of DB ii) V decommits to the n indices
π_2 :	*rWI Proof Phase* 1) P guesses DB as γ; P sends $\mathsf{PBCom}(\gamma)$ 2) P and V jointly generate n random indices, τ, of DB (coin flipping) 3) V decommits to the indices τ of DB 4) P sends zap for "$x \in L$ or γ agrees with at least $1/4$ of the n values of DB in positions τ"

Table 3.1: Outline of protocol $\Pi = (\pi_0, \pi_1, \pi_2)$. CHCom is statistically binding, while SHCom is statistically hiding.

Coin Flipping Subprotocol

1) P_L sends $\mathsf{CHCom}(s)$ to P_R
2) a) P_R applies $(t+1)$-wise independent hash function h to transcript to obtain (R, R', r')
   b) P_R computes $c \leftarrow \mathsf{SHCom}(R')$ using randomness r'
   c) P_R sends R, c to P_L
3) P_L computes $r \leftarrow f_s^{PRF}(R|c)$, and sends r to P_R
4) P_L sends zap for "$x \in L$ or r formatted correctly"
5) P_R decommits R'
6) P_R and P_L output $\tau = r \oplus R'$

Table 3.2: Outline of nearly resettable coin-flipping subprotocol with output τ. f_*^{PRF} is a pseudorandom function family. (P executes P_L and V executes P_R.)

For subprotocol π_0, P and V instantiate the proof system. P sends the setup message for a 2-round statistically hiding, computationally binding commitment scheme. V then constructs an ordered database, DB, consisting of $16tn(n^\epsilon + 1)$ random distinct strings of length n, and sends a statistically hiding commitment of DB to P. V also sends the setup message used by P to execute zap proofs. At this point, P applies a pseudorandom function ($f_*^{P,1}$, with seed chosen using P's initial random tape) to the current transcript and uses the output as his random tape for the rest of the protocol.

For subprotocol π_1, for each of the n^ϵ sequential iterations, P asks for a random sequence

of n entries of DB, which V then decommits to. Note that for this subprotocol, a resetting P can discover at most $tn^{1+\epsilon}$ entries of DB. We note that this protocol is very similar to the protocol in [16]; where their protocol requires $O(n)$-length (random) P commitments, our protocol requires $O(n \log n)$-length random index requests, where both protocols require corresponding V decommitments.

For subprotocol π_2, P guesses V's database and commits to the guess (which we call γ) using a non-interactive perfectly binding commitment scheme. P and V then attempt to jointly compute an $(n \log |DB|)$-length random string as follows: P commits to a seed s using a non-interactive perfectly binding commitment scheme. V uses a $(t+1)$-wise independent hash function h with input the transcript (of π_1 and π_2 thus far) to output a random triple (R, R', r'). V then computes c, a statistically hiding commitment to R' using randomness r', and sends R and c to P. P sends back, using the PRF family $f_*^{P,2}$, $r = f_s^{P,2}(R|c)$. P proves using a zap that either $x \in L$ or that r is properly formed from R, c and the commitment of s. Note that P's commitment to s earlier in π_2 can now be viewed as a partial commitment to a random string. V then decommits to R' and sets $\tau = R' \oplus r$ as the set of n indices of DB. V decommits to the n indices of DB corresponding to τ. Therefore, a resetting P^* can discover at most $tn(n^\epsilon + 1)$ entries, or $1/16$ of DB, including protocol π_1. P provides a zap that either $x \in L$ or that $1/4$ of the entries of P's guess γ corresponds to the final n decommitments from DB (that correspond to τ). We denote this language by Λ_2. We note that only in the case of $x \notin L$ we have that the property that τ is distributed randomly is important; we use this fact to "bootstrap" well-formedness of τ in the unbounded, cheating P^* case.

3.1 Protocol Π Specification in Detail

Before specifying protocol Π, we first introduce required notation.

3.1.1 Required Notation

We begin by fixing the terminology used in the protocol description. We will denote the interactive proof system constructed below by $\Pi = (P, V)$.

Per Section 2, our perfectly binding non-interactive commitment scheme consists of a pair of protocols CHCom and CHVer. Our statistically hiding 2-round commitment scheme consists of the three protocols $\mathsf{SHSetup}$, SHCom, and SHVer.

We denote by $(P_1(x), P_2(y))$ as the interactive protocol between party P_1 with input x and party P_2 with input y. We denote by $P_1(x) \leftrightarrow P_2(y)$ the transcript of public messages for $(P_1(x), P_2(y))$. For an interactive protocol $\pi = (P_1(x), P_2(y))$, we denote by $View_{P_1}(\pi)$ as P_1's view (on input x) of the interaction with P_2 (on input y), which includes the transcript $P_1(x) \leftrightarrow P_2(y)$ as well as the input and auxiliary input of P_1, P_1's random tape, and the set of outputs that P_1 privately computes. We define $View_{P_2}(\pi)$ similarly. We will denote the composition of protocols $\pi_i = (P_1^i, P_2^i)$ and $\pi_j = (P_1^j, P_2^j)$ as $(\pi_i, \pi_j) = ((P_1^i, P_1^j), (P_2^i, P_2^j)$. For an NP language L let R_L be the witness-relation consisting of the set all pairs (x, w) such that $x \in L$ and w is a witness for x in L. We define W_L to be the set such that if $w \in W_L(x)$ then $R_L(x, w) = 1$.

We denote for a language L and $x \in L$ the following two-round zap protocol, which is a two-round simultaneously resettable WI proof system between verifier ZV and prover ZP: ZV on input randomness r computes $\mathsf{zap}_V \leftarrow ZV(r)$ and sends zap_V to ZP. ZP computes $\mathsf{zap}_P \leftarrow ZP_L(x, y, \mathsf{zap}_V)$, where y is the auxiliary input of ZP. ZP sends zap_P to ZV. ZV computes $\{0, 1\} \leftarrow \mathsf{ZVVer}_{\mathsf{V}L}(x, r, \mathsf{zap}_V, \mathsf{zap}_P)$, *accepts* if 1 is output and *rejects* otherwise.

Let n be the security parameter, let ϵ be any positive constant, and let t be the bound on the total number of resets a malicious prover, P^*, can attempt against V.

We denote by $f_*^{P_1}$ and $f_*^{P_2}$ two distinct fixed families of pseudorandom functions used by P.

$\mathcal{H} = \{h_n\}_{n \in \mathbb{N}}$ is a family of $(t+1)$-wise independent hash functions (with appropriate input and output length as described below). We define an ordered set DB consisting of $16tn(n^\epsilon + 1)$ strings of length n. We denote by $DB \leftarrow \mathsf{DBGen}(m_1, m_2, r)$ as the algorithm

that partitions string r of length $m_1 m_2$ into the ordered set of m_1 strings of length m_2. We denote by DB_j the jth entry of DB. Let ind be a string of length $n \log |DB|$ considered as n concatenated indices of DB. Then $DB_{(ind)} \subseteq DB$ corresponds to the n entries of DB given by the indices ind. We construct a set γ as a set of $16tn(n^\epsilon + 1)$ copies of 0^n. We denote by $\gamma \leftarrow \mathsf{GamGen}(m_1, m_2)$ as the algorithm that constructs an ordered set of m_1 strings 0^{m_2}.

We will need two languages, Λ_1 and Λ_2, for the ZK proof system below. Fix language L, a pseudorandom function family $f_*^{P,2}$ and a perfectly binding, computationally hiding non-interactive commitment scheme CHCom. Then $(x, (r, r_1, r_2, s')) \in \Lambda_1$ if either $x \in L$ or there exists some message s and randomness r' such that $s' \leftarrow \mathsf{CHCom}(\mathsf{s}, \mathsf{r}')$ and r is the first $n \log |DB|$ bits of $f_s^{P,2}(r_1|r_2)$. Second, fix a language L and a perfectly binding, computationally hiding non-interactive commitment scheme CHCom. Let μ be a set of $16tn(n^\epsilon + 1)$ computationally hiding, perfectly binding commitments and $\{(DB'_{i_j}, i_j)\}_{j=1,\ldots,n}$ be a set of n pairs where $\cup i_j = I \subset [16tn(n^\epsilon + 1)]$. Then $(x, (\mu, DB', I)) \in \Lambda_2$ if either $x \in L$ or there exists a set DB of order $16tn(n^\epsilon + 1)$ such that there exists a random string r such that $\mu \leftarrow \mathsf{CHCom}(DB, r)$ and at least $1/4$ of the entries in DB given in index set I correspond to the entries in DB'.

3.1.2 Protocol Specification

We construct Π as the sequential composition (π_0, π_1, π_2), where the three subprotocols π_0, π_1 and π_2 are specified below.

$\pi_{\mathbf{0}} = (P_0, V_0)$:
P_0 is given public input x, auxiliary input y and randomness ω_{P_0}. V_0 is given as public input the same x and randomness ω_{V_0}.

(1) P_0 computes $SH_R \leftarrow \mathsf{SHSetup}(\omega_{P_0})$ and sends SH_R to V_0.

(2) (a) V_0 selects a string r of length $16tn^2(n^\epsilon + 1)$ uniformly at random and computes $DB \leftarrow \mathsf{DBGen}(16tn(n^\epsilon + 1), n, r)$.

(b) For $1 \leq j \leq 16tn(n^\epsilon + 1)$, V_0 using (independent) randomness $r_{1,j}$ computes $Com_{SH,j} \leftarrow \mathsf{SHCom}(DB, r_{1,j}, SH_R)$. V_0 sets $Com_{SH} = \{Com_{SH,j}\}_j$ and sends

Com_{SH} to P_0.

(c) V_0 selects randomness r_1 uniformly at random and computes $\mathsf{zap}_V \leftarrow \mathsf{ZV}(r_1)$ and sends zap_V to P_0.

(3) P_0 selects s' uniformly at random and computes random tape parsed as $(\omega_{P1}, \omega_{P2}) = f_{s'}^{P_1}(Com_{SH}, \mathsf{zap}_V, \omega_{P_0})$.

$\pi_{\mathbf{1}} = (P_1, V_1)$:

P_1 is given as input $View_{P_0}(\pi_0)$. V_1 is given as input $View_{V_0}(\pi_0)$. In what follows, P_1 uses ω_{P1} as his random tape.

(1) For the next $2i$ rounds, $1 \leq i \leq n^{\epsilon}$:

a. P_1 chooses uniformly and independently at random (using ω_{P1} as the random tape) a sequence ind of n indices $j_1, ..., j_n \in [16t(n^{\epsilon+1} + n)]$ and sends ind to V_1.

b. For $1 \leq k \leq n$:

(1) V_1 sends (DB_{j_k}, r_{1,j_k}) to P.

(2) P_1 computes $\mathsf{SHVer}(DB_{j_k}, r_{1,j_k}, SH_R, Com_{SH,j_k})$ and aborts if it outputs 0.

$\pi_{\mathbf{2}} = (P_2, V_2)$:

P_2 is given as input $View_{P_0}(\pi_0)$. P_2 uses ω_{P2} as his random tape. V_2 is given as input the view of the previous V_i (e.g. $View_{V_1}(\pi_1)$ if π_1 was the previous subprotocol executed, $View_{V_0}(\pi_0)$ otherwise.). Note that $View_{V_0}(\pi_0) \subseteq View_{V_1}(\pi_1)$.

(1) P_2 selects s and r_2 uniformly at random, computes $\eta \leftarrow \mathsf{CHCom}(s, r_2)$ and sends η to V_2. P_2 checks if:

- y is a witness for $x \in L$; if so, P_2 constructs the set $\gamma \leftarrow \mathsf{GamGen}(16tn(n^{\epsilon}+1), n)$, selects r_3 uniformly at random and computes $\mu \leftarrow \mathsf{CHCom}(\gamma, r_3)$. P_2 sends μ to V_2.

- otherwise, P_2 selects randomness r_3 uniformly at random and computes $\mu \leftarrow \mathsf{CHCom}(y, r_3)$. P_2 sends μ to V_2.

(2) (a) Denote by tr as the public transcript of previous subprotocols in the subprotocol composition together with the transcript of π_2 thus far. V_2 selects a $(t+1)$-wise independent hash function $h \in \mathcal{H}$ uniformly at random, computes $\sigma \leftarrow h(tr)$ and parses σ as the triple (R, R', r').

(b) V_2 computes $c \leftarrow \mathsf{SHCom}(R', r', SH_R)$ and sends (R, c) to P.

(3) (a) P_2 computes $r = f_s^{P_2}(R|c)$, $|r| = n \log |DB|$, cutting the length of r if necessary. P_2 sends r to V_2.

(b) Let $\alpha_1 = (x, (r, R, c, \eta))$ and $wit_1 = (s, r_2)$. P_2 computes $\mathsf{zap}_{P,1} \leftarrow ZP_{\Lambda_1}(\alpha_1, wit_1, \mathsf{zap}_V)$. P_2 sends $\mathsf{zap}_{P,1}$ to V_2.

(4) (a) V_2 computes $\mathsf{ZVVer}_{\mathsf{V},\Lambda_1}(\alpha_1, r_1, \mathsf{zap}_V, \mathsf{zap}_{P,1})$ and aborts if it equals 0.

(b) V_2 sets $\tau = R' \oplus r$ (of length $n \log |DB|$). V_2 sends (R', r') to P_2.

(c) Let $j_1, ..., j_n$ be the n consecutive $(\log |DB|)$-length strings of τ, which we consider as indices of DB. V_2 sends τ and $\{(DB_{j_i}, r_{1,j_i})\}_{1 \le i \le n}$ to P_2.

(5) (a) P_2 computes $\mathsf{SHVer}(R', r', SH_R, c)$ and aborts if it equals 0.

(b) For $1 \le i \le n$, P computes $\mathsf{SHVer}(DB_{j_i}, r_{1,j_i}, SH_R, Com_{SH,j_i})$ and aborts if it outputs 0 for any i.

(c) Let $\alpha_2 = (x, (\mu, \{(DB_{j_i}, j_i)\}_{j_i \in \tau}, \tau))$. If the auxiliary input, y, of P_2 is the witness for $x \in L$, let $wit_2 = y$. Otherwise, let $wit_2 = (\gamma', r_3)$. P_2 computes $\mathsf{zap}_{P,2} \leftarrow ZP_{\Lambda_2}(\alpha_2, wit_2, \mathsf{zap}_V)$. P_2 sends $\mathsf{zap}_{P,2}$ to V.

(6) V_2 computes $\mathsf{ZVVer}_{\mathsf{V}\Lambda_2}(\alpha_2, r_1, \mathsf{zap}_V, \mathsf{zap}_{P,2})$ and aborts if it outputs 0. V_2 outputs accept.

During protocol π_2, steps 1, 2, 3, 4ab and 5a comprise a nearly simultaneously resettable coin flipping scheme, the output of which is the (pseudo)random string τ in step 4b. We now sketch why the coin flipping scheme works. Since $\tau = R' \oplus r$ in step 4b, if a malicious P^* attempts to affect the distribution of τ, he must affect the distribution of either R' or r. However, R' is chosen uniformly at random using a $(t+1)$-wise independent hash function in step 2a, where the input is the protocol transcript. Since P^* can only reset t times, this

output will remain distributed uniformly at random. On the other hand r is the output of $f_s^{P_2}(R|c)$ in step 3a, where R and c are distributed uniformly at random for the same reason that R' is, and since the seed s was committed to in a perfectly binding fashion, P^* must rewind to change s in order to hope to change r. However, in changing s, since (R, R', c) are then changed uniformly at random due to the transcript change, P^* cannot hope to affect r. One final note is that P^* might simply not construct r as $f_s^{P_2}(R|c)$, but in this case, since we assume by definition of soundness that $x \notin L$ by the resettable soundness of the zap argument given in step 3b, P^* *must* format r correctly. Therefore, P^*'s commitment to s in step 1 is essentially a blind commitment to a (pseudo)random string later in the protocol in the case that $x \notin L$.

On the other side, cheating PPT V^* is unable to discover the PRF seed s due to the computational hiding property of P's commitment scheme. Therefore, no matter what the distribution of R and c, by definition of a PRF, since s is chosen uniformly at random, $r = f_s^{P_2}(R|c)$ will be distributed psuedorandomly. Since R' is committed as c in a computationally binding fashion in step 3a, once V^* obtains r, even if he rewinds polynomially many times, he must change c (or R) to change R', but r will then still be distributed pseudorandomly. Therefore, τ will be distributed pseudorandomly.

Theorem 3.1. *Assuming the existence of enhanced trapdoor permutations and collision-resistant hash functions, protocol* Π *is a* t*-bounded resettably sound rZK proof system for* L*.*

The proof that Π is rZK will follow from proofs that Π is t-resettably sound and complete (Lemma 3.2), that (π_0, π_2) is rWI (Theorem 3.3) and that there exists a specific, simpler protocol Π' that is also rZK (Theorem 5.1). We will then prove that since Π' is both rZK and sufficiently similar to Π, in a manner we define as *near compatible*, Π is also rZK (Lemma 4.1).

Lemma 3.2. Π *is* t*-resettably sound and* t*-resettably complete for* L*.*

Completeness follows from the completeness of the zap protocols. Resettable soundness follows from the rWI of zap proofs and the security of the coin-tossing protocol; since P^*

cannot discover 1/4 of the database DB except with negligible probability (due to the statistical hiding property of V's commitment scheme), P^* cannot find a correct witness for language Λ_2 except with negligible probability due to the distribution of the coin-tossing protocol output. See Section 6 for the proof of Lemma 3.2.

In what follows, we will need that the sequential composition (π_0, π_2) is rWI for languages Λ_2 and L.

Theorem 3.3. *Assuming the existence of enhanced trapdoor permutations and collision-resistant hash functions, protocol (π_0, π_2) is rWI for Λ_2 and for L.*

See Section 7 for the proof of Theorem 3.3. The intuition for the proof is that rWI holds due to the rWI of zap proofs, the security of the respective commitment schemes, and the security of the coin-flipping protocol.

4 From a rZK Proof System to a New rZK Proof System

We now outline, first at a high level and then in more detail, how we prove rZK of protocol Π by constructing another protocol where rZK is easier to prove. We note that this definition may likely be generalized, but we only detail properties that will apply in our case for simplicity. It is important to note that the "simpler" rZK protocol does not need to be t-resettably sound; since the purpose here is to prove rZK, resettable soundness is not required. Due to lack of space, we omit here the precise definition of *near-compatible* protocols.

The idea of our transformation stems from the idea of constructing a rZK proof system for L from a rWI proof system; see the constructions of [16, 62, 76]. What generally occurs is that first the setup of the rWI proof is executed; then a so-called extraction protocol is executed, where a cheating prover learns nothing, but a simulator learns some secret s. Finally, a rWI proof is completed for the language "$x \in L$ or the secret s has been learned"; in the specific case of [16], the "secret" was that the prover had committed to a string before the verifier had decommitted to that same string, while in our case, the prover commits to a largely correct guess of the database previously committed to by the verifier.

At a high level, we say that a protocol $\Pi = (\pi_0, \pi_1, \pi_2)$, which is the protocol that we wish to prove rZK, is near-compatible to a protocol $\Pi' = (\pi_0, \pi_1, \pi_2')$ if the following holds. Fix a language $L \in \mathcal{NP}$. Then (π_0, π_1, π_2') is rZK for L. (π_0, π_1, π_2) is an interactive proof for L, and (π_0, π_2) must be rWI so that it does not reveal to the verifier whether the transcript is generated by using the genuine witness of a real prover or by fake witness belonging to a simulator[7]. Finally, we wish that the extraction stage, π_1, is essential for the simulator to complete both (π_0, π_2') and (π_0, π_2) but extraneous for the honest prover.

Lemma 4.1. *(Informal) Fix a language L. Let (π_0, π_1, π_2) be near-compatible to (π_0, π_1, π_2'). Let (π_0, π_1, π_2') be rZK with a simulator that executes honestly for π_0 and π_2' and such that any witness extracted by the simulator is, except with negligible probability, a valid witness for (π_0, π_1, π_2) (with the same messages sent for π_0 and π_1). Then (π_0, π_1, π_2) is rZK.*

The intuition for the proof of Lemma 4.1 is that by definition of near-compatible protocols and by the lemma statement, if simulator Sim' for (π_0, π_1, π_2') is able to extract a witness to complete the protocol, then so is simulator Sim for (π_0, π_1, π_2) that acts identically to Sim' for π_1 and honestly for π_0 and π_2. This is because both simulators act identically for the rounds where extraction occurs. Further, (π_0, π_2) being rWI implies that V^* cannot distinguish whether the transcript is generated by a real prover using a witness for $x \in L$ or by a simulator using an extracted witness.

4.1 Near Compatibility in More Detail

We now give the precise definition of near-compatible protocols and the complete version of Lemma 4.1.

Definition 4.2. *Fix a language L. Let π_0, π_1, π_2, and π_2' be interactive protocols. Consider the interactive protocols (π_0, π_1, π_2), considered as the triple of PPT pairs $((P_0, V_0), (P_1, V_1), (P_2, V_2))$, and (π_0, π_1, π_2'), considered as the triple of PPT pairs*

[7]Some additional technical properties specified in the precise definition: it is enough for our purposes that the setup phase, π_0, consists of one round of messages sent by P followed by a round of messages sent by V. In order to prove the lemma, we will require security reductions that will need limited access to the prover's random tape; therefore, P's message for π_0 must be public coin.

$((P_0, V_0), (P_1, V_1), (P_2', V_2'))$. (π_0, π_1, π_2) *is called* near-compatible *to* (π_0, π_1, π_2') *if the following holds:*

(1) y *and language* L_1 *are determined from the transcript of* (π_0, π_2')*, and let* L_1' *be the language "*$x \in L$ or $y \in L_1$*".*

(2) (π_0, π_1, π_2') *is rZK for* L*.*

(3) (π_0, π_2') *is an interactive proof system for* L_1' *and* L*, where auxiliary input is given to* P_2' *during* π_2'*.*

(4) y' *and language* L_2 *is determined from the transcript of* (π_0, π_2)*, and let* L_2' *be the language "*$x \in L$ or $y' \in L_2$*".*

(5) (π_0, π_2) *is rWI for* L_2' *and for* L*, where auxiliary input is given to* P_2 *during* π_2*.*

(6) π_0 *consists of* P_0 *sending* V_0 *a single sequences of messages followed by* V_0 *sending* P_0 *a single sequence of messages, termed the* determining message*.*

(7) P_0 *is public coin; that is,* P_0*'s public output, without loss of generality, consists solely of the initial component of his random tape. Further, the private output of* P_0 *is a pseudorandom function* f_s^{PRF}*, where* s *is chosen from* P_0*'s random tape, applied to the transcript of* π_0*. The output is parsed as* (ω_1, ω_2)*.*

(8) P_1 *on uses as input the public transcript for* π_0 *and* ω_1 *for random tape, while* P_2 *and* P_2' *only require as input the public transcript for* π_0*,* ω_2 *for random tape, and the auxiliary input.*

We note that properties 4-8 of Definition 4.2 hold for the protocol Π, where properties 4, 6 and 7 are easily verified from the protocol specification and property 5 is due to Theorem 3.3. We proceed to the technical lemma that we use as the framework to prove Theorem 3.1.

Lemma 4.3. *Fix a language* L*. Let* π_0*,* π_1*,* π_2*, and* π_2' *be interactive protocols such that the protocols* (π_0, π_1, π_2)*, considered as the triple of PPT pairs* $((P_0, V_0), (P_1, V_1), (P_2, V_2))$*, is near-compatible to* (π_0, π_1, π_2')*, considered as the triple of PPT pairs* $((P_0, V_0), (P_1, V_1), (P_2', V_2'))$*, as defined in Definition 4.2 above. Suppose the following held for an rZK simulator* Sim' *for* (π_0, π_1, π_2') *and for a proposed simulator* Sim *for* (π_0, π_1, π_2)*:*

a. *There exists a PPT algorithm* $\mathcal{S}$ *such that the execution of* Sim' *with oracle access to* V'^* *is run as the composition of algorithms* $(P_0, \mathcal{S}, P_2')$ *(that is,* Sim' *runs straight-line for* π_0 *and* π_2'*).*

b. *The execution of* Sim *with oracle access to* V^* *is executed as the composition of algorithms* $(P_0, \mathcal{S}, P_2)$*.*

c. *Any witness for* $z \in L_1'$ *obtained by* Sim' *for* (π_0, π_1, π_2') *is, except with negligible probability, a valid witness for* $z' \in L_2'$ *where* L_2' *is determined in* (π_0, π_2) *and identical messages were sent for* π_0 *in both protocols.*[8]

Then the protocol (π_0, π_1, π_2) *is rZK for* L*; in particular, the simulator* Sim *that acts honestly for* π_0 *and* π_2 *and identically to* Sim' *for messages of* π_1 *satisfies the rZK property for* (π_0, π_1, π_2)*.*

See Section 8 for the proof of Lemma 4.3.

5 An Admissible, Near-Compatible rZK Proof System

Here we outline an admissible rZK proof system that has the same initialization phase and extraction phase as protocol Π but with a simplified end stage in order to make the proof of rZK easier. In particular, (π_0, π_1, π_2') is not constructed to be t-resettably sound, and therefore the verifier can eventually reveal the entire DB. For lack of space, we will only sketch π_2'. π_2' has the same inputs as π_2 above. π_2' also begins like π_2: first, prover commits his guess γ using a statistically binding commitment scheme and sends it to V. Then, however, V decommits the *entire* DB. P then executes a zap that either $x \in L$ or that γ corresponds to 1/4 of DB; we denote this new language by Λ_3.

Note that by construction, (π_0, π_1, π_2') satisfies the respective properties of near-compatibility. Further, (π_0, π_1, π_2') is admissible since the verifier, after its initial message, only sends decommitments.

[8] We note that $z' \in L_2'$ may not be completely defined during π_0. Instead, we say that a run of π_0 as part of the composition (π_0, π_1, π_2) determines a set of possible $z_i' \in L_2'$ to be completely determined in π_2. Then, for any witness w obtained by Sim for L_1' where the run π_0 is the same for (π_0, π_1, π_2') as it is for (π_0, π_1, π_2), this witness is a correct witness for any $z_i' \in L_2'$ except with negligible probability.

Since (π_0, π_1, π_2') and (π_0, π_1, π_2) are near-compatible, the only remaining subtlety is to note that the witness extraction property of Lemma 4.1 holds. But this is indeed the case because the simulator, will extract a set of entries from V that correspond to at least $1/2$ of V's DB except with negligible probability. Since, for π_2, the entries chosen for Λ_2 are selected uniformly at random from DB, if the simulator knows $1/2$ of DB, then the simulator will know $1/4$ of the entries selected for Λ_2 except with negligible probability.

5.1 π_2' Specification in More Detail

See Sections 2 and 3.1.1 for the notation used for this protocol.

We need a new language Λ_3: Fix a language L and a perfectly binding, computationally hiding non-interactive commitment scheme CHCom. Let μ be a set of $16tn(n^\epsilon + 1)$ computationally hiding, perfectly binding commitments of strings of length n and DB be a set of $16tn(n^\epsilon + 1)$ strings of length n. Then $(x, (\mu, DB)) \in \Lambda_3$ if either $x \in L$ or there exists a set DB' of order $16tn(n^\epsilon + 1)$ such that there exists a random string r such that $\mu \leftarrow \mathsf{CHCom}(DB', r)$ and at least $1/3$ of the entries in DB' equal the entries in the corresponding positions of DB.

We construct this protocol as the sequential composition of three subprotocols π_0, π_1 and π_2' as noted within the protocol specification. Because π_0 and π_1 have already been specified in Section 3.1, we give here only the specification for π_2'.

$\boldsymbol{\pi_2'} = (P_2', V_2')$:
P_2' is given as input $View_{P_0}(\pi_0)$. P_2' uses ω_{P2} as his random tape. V_2' is given as input $View_{V_1}(\pi_1)$. Note that $View_{V_0}(\pi_0) \subseteq View_{V_1}(\pi_1)$.

(1) P_2' checks if:

- y is a witness for $x \in L$; if so, P_2' constructs the set $\gamma \leftarrow \mathsf{GamGen}(16tn(n^\epsilon+1), n)$, selects r_3 uniformly at random and computes $\mu \leftarrow \mathsf{CHCom}(\gamma, r_3)$. P_2' sends μ to V_2'.
- otherwise, P_2' selects randomness r_3 uniformly at random and computes $\mu \leftarrow$

$\mathsf{CHCom}(y, r_3)$. P_2' sends μ to V_2'.

(2) V_2' sets $r_1' = \{r_{1,j_i}\}_i$ and sends (DB, r_1') to P.

(3) (a) For $1 \leq i \leq |DB|$ P_2' computes $\mathsf{SHVer}(DB_i, r_{1,j_i}, Com_{SH,j_i}, SH_R)$ and aborts if 0 is ever output.

(b) Let $\alpha = (x, (\mu, DB))$. If the auxiliary input, y, of P_2' is the witness for $x \in L$, let $wit_3 = y$. Otherwise, let $wit_3 = (\gamma', r_3)$. P_2' computes $\mathsf{zap}_{P,3} \leftarrow ZP_{\Lambda_3}(\alpha, wit_3, \mathsf{zap}_V)$. P_2' sends $\mathsf{zap}_{P,3}$ to V_2'.

(4) V_2' computes $\mathsf{ZVVer}_{\mathsf{V}\Lambda_3}(\alpha, r_1, \mathsf{zap}_V, \mathsf{zap}_{P,3})$ and aborts if he rejects. V_2' outputs accept.

Theorem 5.1. *Assuming the existence of enhanced trapdoor permutations and collision-resistant hash functions, protocol (π_0, π_1, π_2') is zero knowledge in the hybrid model (i.e, hZK) for L.*

Since the protocol (π_0, π_1, π_2') is hZK and already in the form needed to transform zero-knowledge proofs secure in the hybrid model to zero-knowledge proofs secure in the resettably model, Theorem 5.1 implies the following[9].

Corollary 5.2. *(π_0, π_1, π_2') is rZK for L.*

See Section 5.2 for a high level discussion of the proof of Theorem 5.1 and Section 10 for the full proof.

We would like to contrast the protocol (π_0, π_1, π_2') with that given in [16]. As noted in the high-level outline of Π in Section 3, the extraction stage of [16] and the subprotocol π_1 are very similar. Indeed, π_2' is a natural extension of the protocol in [16] because in both their protocol and ours, DB is revealed and the rWI proof incorporates the whole DB. In order to prove Theorem 5.1, we will need the fact that (π_0, π_2') is rWI for Λ_3 and for L.

Lemma 5.3. *Assuming the existence of enhanced trapdoor permutations and collision-resistant hash functions, then protocol (π_0, π_2') is rWI for Λ_3 and for L.*

[9]We note that the simulator does not change from Theorem 5.1 to Corollary 5.2. The reason is that the proof in [16] that takes a hZK protocol and proves that it is rZK does not change the simulator; rather, it proves that for every hybrid adversary there exists a corresponding adversary that however is still simulatable. In particular, if the simulator given here in the hybrid model only rewinds during π_1 and otherwise executes honestly, so does the simulator in the full rZK model.

See Section 9 for a proof of Lemma 5.3

5.2 High-Level Simulator Strategy in the Proof of Theorem 5.1

In [16], the high level strategy of the simulator was that it would try to "look ahead" to try to figure out the verifier's commitment ahead of time, but otherwise execute honestly for all other (non-extraction stage) rounds. This is also true for the simulator here: the simulator would like to discover as many DB decommitments as possible and otherwise executes honestly. The most important difference between the rZK simulator here and the simulator in [16] is that their protocol only requires 1 successful look-ahead to proceed, while our protocol requires polynomially many successful look-aheads to proceed.

To construct our simulator, we will use a nearly identical strategy as the simulator from [16] except that we will execute the individual (main-thread level) look-aheads $|DB|$ many times in parallel. Namely, the simulator Sim' in the extraction stage, π_1, attempts to discover half of DB; if this has not occurred at the end of the extraction stage, then the simulator simply aborts and fails to complete. One of the main inefficiencies of the [16] simulator is that it computes a distinct look-ahead subprotocol run (embedded in the subroutine $\mathsf{NextProverMsg}$, which then unfolds recursively, see details in [16]) at each of the n^ϵ round iterations of the extraction stage. The idea of their simulator is that if the simulator makes a distinct look-ahead subprotocol run at each round, which in turn consists of polynomially many look-ahead attempts, then except with negligible probability, the simulator will be able to extract one "secret". Since the look-ahead subprotocol success probability is independent from one round to the next, the strategy of our simulator is that instead of making one independent look-ahead subprotocol run at each round, we make $poly(n,t) = |DB|$ independent calls at each (main-thread) iteration of the extraction stage[10]. By a union bound, our simulator will also fail to extract only with negligible probability. A subtlety is that $|DB|$ successful look-aheads might not reveal as much of $|DB|$ as desired. However, because the prover messages in π_1 consist of n randomly chosen indices, V^* is unable to both complete

[10]We make these independent calls only at the main-thread level of the recursion; in this manner, simulator run-time does not expand exponentially.

the protocol with P/Sim' and sufficiently control the distribution of the prover messages that V^* chooses to proceed with.

See Section 10 for the proof of Theorem 5.1, including a full simulator specification.

6 Completeness and t-Resettable Soundness of Π

We here give the formal proofs for the completeness and the t-resettable soundness of the protocol $\Pi = (\pi_0, \pi_1, \pi_2)$.

Proof of Lemma 3.2. *t-Resettable Completeness:* Fix a language L and let $x \in L$. Let P^* select an incarnation $V^{(i)}$. For simplicity of notation we will refer to $V^{(i)}$ as V. If V accepts the proof, then V accepts both WI proofs in Π (namely, V outputs 1 for both $\mathsf{ZVVer}_{\mathsf{V},\Lambda_1}(\alpha_1, r_1, \mathsf{zap}_V, \mathsf{zap}_{P,1})$ in step 4a of π_2 and $\mathsf{ZVVer}_{\mathsf{V}\Lambda_2}(\alpha_2, r_1, \mathsf{zap}_V, \mathsf{zap}_{P,2})$ in step 6 of π_2). Since, by definition of resettable completeness, P^* adopts an honest strategy P, P^* will execute the entire proof according to the protocol specification, as will V. Therefore, P^* correctly formats r and uses the witness for its proper formatting (namely (s, r_2)) for the witness wit_1 for $\mathsf{zap}_{P,1}$ in step 3b of π_2 and uses the witness for $x \in L$ for $\mathsf{zap}_{P,2}$ in step 5c of π_2 (and also correctly formats all other output messages, else V will abort). Thus by the completeness of WI proofs, V will accept both proofs and will therefore accept the protocol.

t-Resettable Soundness: The intuition for resettable soundness is that since Π uses as subprotocols rWI zaps, P^* must find correct witnesses for those proofs. In order to properly construct $\mathsf{zap}_{P,2}$ in step 5c of π_2, since P^* does not know the witness for $x \in L$, P^* must construct a database that, except with negligible probability, matches V's DB in at least $1/4$ of its entries. However, P^* cannot discover $1/4$ of the database DB except with negligible probability because the protocol contains more than 4 times the number of entries that P^* could discover by resetting t times. Therefore P^* must try to change the distribution of the V^*'s decommitment to DB in step 4c of π_2 used for the final zap proof. However, the protocol to construct the indices used for the decommitment outputs a random string even

when P^* is malicious and unbounded, and therefore P^* will not be able to cheat except with negligible probability. A formal proof follows.

Suppose $x \notin L$ and fix a strategy P^*. Note that P^* is computationally unbounded. We first demonstrate that for any incarnation $V^{(i)}$ to accept the first zap, $\mathsf{ZVVer}_{\mathsf{V},\ \Lambda_1}(\alpha_1, r_1, \mathsf{zap}_V, \mathsf{zap}_{P,1})$ in step 4a of π_2, when $x \notin L$ implies that τ is distributed uniformly at random. We then demonstrate that if τ is distributed uniformly at random and $x \notin L$, the $V^{(i)}$ will accept the second zap, $\mathsf{ZVVer}_{\mathsf{V}\Lambda_2}(\alpha_2, r_1, \mathsf{zap}_V, \mathsf{zap}_{P,2})$ in step 6 of π_2, only with negligible probability in the security parameter n.

Since zaps are resettably sound interactive proofs and $x \notin L$, by resettable soundness, r must be constructed by P^* according to the protocol specification, which is $r = f_s^{P_2}(R|c)$, $|r| = n \log |DB|$. It is not important that r is distributed uniformly at random; what *is* important is that P cannot change $r = f_s^{P_2}(R|c)$ in step 3a of π_2 without creating a new (independently) random pair (R, c) output by any incarnation $V^{(i)}$ in step 2b of π_2, otherwise once P^* received the decommitment to R', P could then affect the distribution of τ. Indeed, since R', R and r' are constructed using the output of a $(t+1)$-wise independent hash function (chosen uniformly at random by each $V^{(i)}$ in step 2a of π_2) that incorporates P^*'s index queries and commitment to the seed s, P^* cannot change the distribution of R, R', r', even by moving from one incarnation of V to the next. Note that P^* could call a new incarnation V that used the same h as another incarnation; however, P^* will still not know precisely h (because P^* does not know the randomness used by an incarnation) and can only receive at most t outputs of h due to the t-bounded resetting property of P^*, so the uniform randomness of h's output holds. By the perfectly binding property of CHCom, P^* cannot change s and therefore cannot change the output $r = f_s^{P_2}(R)$ in step 3a of π_2 without receiving a new uniformly random pair (R, c).[11] Since c is statistically hiding for R', P^* cannot determine R' from c except with negligible probability and therefore, since $\tau = r \oplus R'$ and R' is distributed uniformly at random with r constructed independently from

[11] Note that the randomness used by V to construct c as the commitment of R' is also crucial to this fact.

R', τ is distributed uniformly at random.

Since $V^{(i)}$ chose its DB uniformly at random in step 2a of π_0, P^* cannot a priori predict independent entries of DB except with negligible probability, even when P^* already knows some of the entries of DB of any incarnation[12] of V. Therefore for P^* to reset $V^{(i)}$ t times implies that P^* knows at most $t(n^{c+1} + n)$ entries, or 1/16 of DB. Since τ is uniformly distributed, so are the corresponding decommitments of DB. The probability that P^*, who knows only 1/16 of DB, can correctly guess 1/4 of n entries chosen from DB uniformly at random is negligible in n by a Chernoff bound (to be precise, the probability is upper-bounded by $(\frac{e^3}{256})^{\frac{n}{16}}$). Therefore, if $x \notin L$, any strategy P^* has at most a negligible probability of $V^{(i)}$ accepting, for any incarnation $V^{(i)}$. □

7 Proof of Theorem 3.3

Remark 7.1. The proof that the interactive protocol (π_0, π_2) has completeness for L and Λ_2 is essentially the same as the proof of Lemma 3.2, and the proof that the protocol has soundness for Λ_2 also follows from Lemma 3.2 (in particular, if neither $x \in L$ nor the database guess is correct, then since in particular $x \notin L$, this reduces to the soundness implied by the t-resettable soundness of Lemma 3.2).

Proof of Theorem 3.3. We proceed with a series of hybrid experiments and prove that (π_0, π_2) is rWI by reductions to the computational hiding property of CHCom as well as the rWI property of zaps. We will also need that τ is distributed pseudorandomly, which we will reduce to the rWI of zaps as well as the indistinguishability of pseudorandom functions. We note that $\mathsf{zap}s$ are not inherently rWI but do have the property when the prover message is computed using a random tape that is a PRF output of prover's initial randomness as well as verifier's determining message (including zap_V, discussed below). A formal proof follows.

We fix a verifier strategy V^*. For a sequence $\bar{x} = x_1, ..., x_{poly(n)}$, we define a corresponding

[12]More specifically, since for some incarnations, DB might be the same, we mean that P^* cannot learn *additional* entries in those DBs a priori.

sequence of witnesses $\bar{y} = y_1, ..., y_{poly(n)}$ such that $\bar{y}_i \in W_L(\bar{x}_i)$ for $1 \leq i \leq poly(n)$. For an ensemble of databases $\bar{DB} = DB^1, ..., DB^{poly(n)}$, a corresponding ensemble of witnesses $\bar{\gamma} = \gamma^1, ..., \gamma^{poly(n)}$ is constructed such that, for any sequence of n randomly chosen positions, at least $1/4$ of each of the corresponding entries of γ^i is equal the corresponding entries in DB^i except with negligible probability (note that it is possible that $DB^i = DB^j$ for $i \neq j$). We note that while the existence of such a witness set is highly unlikely, all that matters is that a distinguisher cannot distinguish cases when a prover, with unbelievable luck, has such a witness set and uses it to complete the protocol (π_0, π_2). In what follows, we will assume that V^* tries to distinguish between when P uses a witness for $\bar{x}$ and when P uses a witness for DB to complete (π_0, π_2). Since the language Λ_2 is specified with a perfectly binding commitment to a guess by P for DB, there can only ever be one witness for DB in a particular execution of (π_0, π_2) for Λ_2. While V^* must also be able to distinguish the case where P uses different witnesses for $\bar{x}$, the proof for this case is strictly simpler than for the case we consider here (this is because no hybrid distribution for commitments are needed and otherwise the proof runs nearly identically to the one given here).

Let K be the total (maximum) number of session resets that V^* will require[13] (if such a later session doesn't occur, neither P nor V^* will output anything). Therefore, K is also a bound on the number of incarnations of P interact with. We denote by the *determining message* the sequence of public messages sent by V^* in step 1 of π_0. We denote by a *consistent protocol class* of P to be the set of incarnations of P interacting with V^* that all have the same index (i, j) and all receive the same determining message. We note that, except with negligible probability, V^* sending the same distinguishing message as an earlier interaction implies that V^* committed to the same DB before except with negligible probability, since V^* commits to DB with a computationally binding commitment scheme and V^* is a PPT algorithm.

We specify a sequence of hybrid games. In all the games listed below, per protocol specification, P will never use the witness from $\bar{x}$ for $\mathsf{zap}_{P,1}$ in step 3b of π_2 but rather the

[13]Recall that such a bound is explicit for any verifier strategy V^* per the definition given about for rWI.

witness that r is well-formed from (R, c, s, r_2). Note that after this series of hybrid games, we will require another series of hybrids where $\mathsf{zap}_{P,1}$ will come under further examination.

$\mathbf{H_0}$ In this hybrid experiment, all incarnations of P with V^* execute honestly, using the witnesses $\bar{y}$ for $\bar{x}$.

$\mathbf{H_1^i}$ For each of the first $i = 0, 1, ..., K$ consistent protocol classes of P, P uses a (consistent over protocol class) uniformly random tape throughout the protocol to execute π_2 instead of using a random tape from step 3 of π_0 onwards that is an output of the pseudorandom function $f_{s'}^{P,1}$, where s is taken from P's initial random tape. For each of the remaining consistent protocol classes, continue to execute the protocol per π_0, using the PRF $f_*^{P,2}$ to construct the random tape for π_2. Note that H_1^0 is the same experiment as H_0.

Suppose for some hybrids H_1^s and H_1^{s+1}, where verifier V^* interacts with prover incarnations $P^{(i,j)}$ with public input x_i, auxiliary input y_i, and random tape ω_j, there exists a distinguisher D_{WI} such that D_{WI} distinguishes V^*'s view for H_1^s from V^*'s view for H_1^{s+1} with non-negligible probability. Then we reduce to distinguishability of the PRF $f_*^{P,1}$. Let D_{PRF} be a distinguisher with oracle access to the oracle F_n, which is uniformly distributed over $f_*^{P,1}$ with appropriate input length, and the oracle R_n, which is uniformly distributed over the set of random functions with the appropriate input length. D_{PRF} is given as input $\bar{x}$, $\bar{y}$, the random tapes of the $P^{(i,j)}$, denoted $\bar{\omega}$, as well as the auxiliary input of V^*. D_{PRF} executes the $P^{(i,j)}$ and V^* internally and executes as follows: For each of the first $s-1$ consistent protocol classes of P that V^* calls to interact with, D_{PRF} executes the protocol between the invoked $P^{(i,j)}$ and V^* using $P^{(i,j)}$'s random tapes (one for π_0, one for π_2, per specification) throughout the interaction.

For the s and $s+1$ consistent protocol classes of P invoked by V^*, denote by ω_s and ω_{s+1} the random tapes of these incarnations. D_{PRF} executes the interaction between V^* and these consistent protocol classes as follows: Per specification of π_0, the respective incarnations of P and V^* interact per steps 1 and 2 of π_0. Finally, for step 3 of P_0, D_{PRF} sends $(Com_{SH}, \mathsf{zap}_V)$,

the random tapes of the respective incarnations and the PRF random seeds used by the respective incarnations to the oracles and receives two strings, ω_{P_s} and $\omega_{P_{s+1}}$. D_{PRF} executes the rest of the interaction of the s consistent protocol class of P using as random tape ω_{P_s} and the $s+1$ consistent protocol class of P using as random tape $\omega_{P_{s+1}}$. Note that one of these hybrids is using a random tape that is a PRF output and the other is using a random tape that is the output of a random function.

Finally, for the each of the last $s+2$ consistent protocol classes of P that V^* calls to interact with, D_{PRF} executes the protocol between the invoked $P^{(i,j)}$ and V^* according to the protocol specification of π_0, e.g. using for π_2 the random tape output by step 2 of π_0.

We note that the internal execution of (π_0, π_2) for hybrid experiments H_1^s and H_1^{s+1} are distributed identically to the external executions distinguished by D_{WI}. Therefore, D_{PRF} executes D_{WI} as a subroutine and distinguishes the oracle calls with non-negligible probability, a contradiction.

$\mathbf{H_2^i}$ Each of the following hybrid experiments are conducted identically to H_1^K except that for each of the first $i = 0, 1, ..., K$ consistent protocol classes of P, the following holds. In each of the lth consistent protocol classes, $1 \leq l \leq i$, P in step 1 of π_2 commits to a set γ^l, such that, except with negligible probability, any uniformly selected set of n indices of γ^l corresponds to those in DB^l, where DB^l is the DB committed by V^* in step 2b of π_0 of the l consistent protocol class. Recall that the commitment to DB^l is part of the determining message. All other consistent protocol classes commit to γ per GamGen in step 1 of π_2. Note that H_2^0 is identical to H_1^K.

Suppose for some hybrids H_2^s and H_2^{s+1}, where verifier V^* interacts with consistent protocol classes $P^{(i,j)}$ with public input x_i, auxiliary input y_i, and random tape ω_j, there exists a distinguisher D_{WI} such that D_{WI} distinguishes V^*'s view for H_2^s from V^*'s view for H_2^{s+1} with non-negligible probability. We wish to reduce to the hiding property of the commitment scheme. Consider the malicious receiver R^* and honest committer C for the corresponding 1-round perfectly binding commitment scheme executing CHCom and CHVer. R^* is given as

auxiliary input $\bar{x}$ and $\bar{y}$. R^* executes V^* internally and constructs the incarnations $P'^{(i,j)}$ for the corresponding $P^{(i,j)}$ that V^* interacts with with the same x_i and y_i. Each consistent protocol class is given the same two random tapes, one for π_0 and one for π_2. All incarnations with the same index have first tape identical to each other as well. Then R^* executes internally, with V^* and each of the incarnations of P for the first $s-1$ consistent protocol classes, where each of the incarnations commit to their respective DB guesses γ^l, $1 \leq l \leq s-1$.

Then, R^* sends γ^s and the 0 set computed per GamGen to C. C sends back com_1 and com_2. R^* executes the s and $s+1$ consistent protocol classes of P that interact with V^* using as message for step 1 of π_2 com_1 for the s consistent protocol class and com_2 for the $s+1$ consistent protocol class.

For the remaining consistent protocol classes, R^* uses 0 set computed per GamGen to execute step 1 of π_2. We note that by construction, the distributions constructed by R^* for (π_0, π_2) are distributed identically to the hybrid experiment distributions H_2^s and H_2^{s+1} because both have the same random tape distributions for the incarnations of P and all commitments, whether computed internally by R^* or externally by C, are computed as the "real" incarnations of P would have computed them. R^* executes a distinguisher D_{COM} for his views by running D_{WI} as a subroutine on the views of R^* identically distributed as H_2^s and H_3^{s+1} and outputs 1 when D_{WI} does. Therefore R^* with D_{COM} can distinguish commitments with non-negligible probability, a contradiction of the computational hiding property of the commitment scheme.

$\mathbf{H_3^i}$. Each of the following hybrid experiments are conducted identically to H_2^K except that for each of the first $i = 0, 1, ..., K$ consistent protocol classes of P, in the lth consistent protocol class, $1 \leq l \leq i$, uses as witness γ^l instead of the $\bar{y}$ witness for $\bar{x}$ for $\mathsf{zap}_{P,2}$ in step 5c of π_2. Note that H_3^0 is the same experiment as H_2^K.

In the following hybrid distribution argument, we assume that τ, sent by V^* in step 4b of π_2 is distributed pseudorandomly. We will prove this property after the main sequence of hybrid distributions.

Suppose for some hybrids H_3^s and H_3^{s+1}, where verifier V^* interacts with consistent protocol classes $P^{(i,j)}$ with public input x_i, auxiliary input y_i, and random tape ω_j, there exists a distinguisher D_{WI} such that D_{WI} distinguishes V^*'s view for H_3^s from V^*'s view for H_3^{s+1} with non-negligible probability. We wish to reduce to the rWI property of zap protocols. Consider a malicious V_{zap}^* and honest P_{zap} for a zap proof system.

We remark on how the P_{zap} incarnations are called by V_{zap}^*. We must be careful with specifying the incarnations of P_{zap} as the incarnations of P^* are specified both by their indices and their determining message; but the determining message for P_{zap}^*, namely zap_V, is different than the determining message for P, which includes the commitment to DB. However, V^* can only commit to at most K DBs for an incarnation of P (and, except with negligible probability, different commitments of DBs correspond to different DBs by the computational binding property of V^*'s commitment scheme SHCom). Therefore, for each incarnation $P^{(i,j)}$ of P, we specify K incarnations of P_{zap}, $P_{zap}^{(i,j_1)}$,..., $P_{zap}^{(i,j_K)}$, one for each consistent protocol class of P with index (i,j) that V^* may execute internally.

V^* is given as auxiliary input $\bar{x}$ and $\bar{y}$ as well as the witnesses γ^l for each of the l consistent protocol classes, $1 \leq l \leq K$. V_{zap}^* then executes V^* internally and instantiates the various incarnations $P^{(i,j)}$ in the same manner as specified by R^* in the above hybrid distribution proof. Then V_{zap}^* executes the interaction with V^* and the interactions of P internally except that, when V^* sends zap_V and a commitment of DB (e.g. step 1 of π_0, which is the determining message) to an incarnation of P, V_{zap}^* forwards that message to the corresponding incarnation of P_{zap}.

When the language Λ_2 is specified, V_{zap}^* forwards the language description to the corresponding incarnation of P_{zap}; in particular the committed DB guess by P, μ, is sent in step 1 of π_2, and τ and the corresponding decommitments to DB are sent in step 4 of π_2. Then:

(a) If the message that V_{zap}^* is forwarding from V^* is for one of the first $s-1$ consistent protocol classes of P, V_{zap}^* sends the corresponding γ^l and randomness used to compute μ to P_{zap}, who computes $\mathsf{zap}_{P,2}$ using it and sends $\mathsf{zap}_{P,2}$ to V_{zap}^*. V_{zap}^* forwards this message to V^* for step 5c of π_2 instead of P's message.

(b) For the s and $s+1$ consistent protocol classes of P, V^*_{zap} forwards y_s and γ^s (and random string used for μ) and y_{s+1} and γ^{s+1} (and random string used for μ), respectively. Then for one of s and $s+1$, P_{zap} uses the witness for $\bar{x}$ and for the other, P_{zap} uses the witness, γ, for DB. P_{zap} then sends these messages to V^*_{zap} who forwards them to their respective internal executions at step 5c of π_2 with V^* in place of P's respective messages for $\mathsf{zap}_{P,2}$.

(c) If the message that V^*_{zap} is forwarding is from the $s+2$ and later consistent protocol classes of P, V^* sends the corresponding y_l to P_{zap}, who computes $\mathsf{zap}_{P,2}$ and sends it to V^*_{zap}. V^*_{zap} sends this message to V^* for step 5c of π_2 instead of P's message.

Note that since τ is assumed to be distributed pseudorandomly, P_{zap} is able to compute a correct $\mathsf{zap}_{P,2}$ except with negligible probability (which is the same for "real" P). We note that by construction, the distributions constructed by V^*_{zap} for (π_0, π_2) are distributed identically to the hybrid experiment distributions H^s_3 and H^{s+1}_3 because both have the same random tape distributions for the incarnations of P and all zap messages, when computed by P_{zap}, are computed identically to how the "real" incarnations of P with the same randomness would have computed them.

V^*_{zap} executes a distinguisher D_{zap} over his views by running D_{WI} as a subroutine on the views of V^*_{zap} identically distributed as H^s_3 and H^{s+1}_3 and outputs 1 when D_{WI} does. Therefore $V*_{zap}$ with D_{zap} can distinguish zap transcripts with non-negligible probability, a contradiction of the rWI property of zaps.

$\mathbf{H^i_4}$. Each of the following hybrid experiments are conducted identically to H^K_3 except that for each of the first $i = 0, 1, ..., K$ consistent protocol classes of P, the incarnations use a random tape from step 3 of π_0 onwards that is an output of the pseudorandom function $f^{P,1}_{s'}$ on the determining message, where s' is taken from P's initial random tape. Note that H^0_4 is identical to H^K_3 and that H^K_4 is the transcript of the incarnations of P interacting with V^* using the witness for DB instead of $x \in L$. Indistinguishability of H^s_4 from H^{s+1}_4 follows by essentially the same reasoning as was given for the indistinguishability of H^s_1 from H^{s+1}_1.

Claim 7.2. *For any PPT strategy V^*, τ is distributed pseudorandomly except with negligible probability.*

The claim follows if we can prove that r sent in step 3a of π_2 is pseudorandom. Since $r = f_s^{P,2}(R|c)$ and c is a computationally binding commitment to R', $\tau = r \oplus R'$ must be distributed pseudorandomly and the claim follows.

We note that in proving that r is distributed pseudorandomly, we slightly modify the protocol in a manner that does not (we prove) change the distribution of (π_0, π_2). Namely, We fix a verifier strategy V^* and set for P public input $\bar{x}$ and a set of auxiliary inputs $\bar{y}$ that P uses to compute $\mathsf{zap}_{P,2}$. We further assume that each P has additional auxiliary input $\tilde{y}$ such that for $1 \leq i \leq poly(n)$, $\tilde{y}_i \in W_L(x_i)$ (and if that incarnation of P is not supposed to use a witness for $x \in L$ below for $\mathsf{zap}_{P,1}$, we assume $\tilde{y}_i = 0$). We need these because otherwise we cannot reduce to the case where r is output randomly (since the only way to do so is if P possesses a witness for $x \in L$). In sum, P uses $\bar{y}$ to compute $\mathsf{zap}_{P,2}$ but uses $\tilde{y}$ to compute $\mathsf{zap}_{P,1}$. Then we construct the following hybrid experiments:

$\mathbf{H_0'}$. In this hybrid experiment, V^* interacts with incarnations P using public input $\bar{x}$ and auxiliary input $\bar{y}$.

$\mathbf{H_1'^i}$. Each of the following hybrid experiments are conducted identically to H_0' except that for each of the first $i = 0, 1, ..., K$ consistent protocol classes of P, P uses a uniformly random tape throughout the protocol instead of using a random tape from step 3 of π_0 onwards that is an output of the pseudorandom function $f_{s'}^{P,1}$, where s is taken from P's initial random tape. Indistinguishability of $H_1'^s$ from $H_1'^{s+1}$ follows for the same reasoning as for H_1^s and H_1^{s+1} above.

$\mathbf{H_2'^i}$. Each of the following hybrid experiments are conducted identically to $H_1'^i$ except that for each of the first $i = 0, 1, ..., K$ consistent protocol classes of P, P formats the message

$\mathsf{zap}_{P,1}$ using as witness the corresponding entry of $\tilde{y}$.

Suppose for some hybrids $H_2^{'l}$ and $H_2^{'l+1}$, where verifier V^* interacts with consistent protocol classes $P^{(i,j)}$ with public input x_i, auxiliary input $\bar{y}_i$, and random tape ω_j, there exists a distinguisher D_{rand} such that D_{rand} distinguishes V^*'s view for $H_2^{'l}$ from V^*'s view for $H_2^{'l+1}$ with non-negligible probability. We wish to reduce to the rWI property of zap protocols. Consider a malicious V_{zap}^* and honest P_{zap} for a zap proof system. We must be careful with specifying the incarnations of P_{zap} as the incarnations of P^* are specified both by their indices and their determining message; but the determining message for P_{zap}^*, namely zap_V, is different than the determining message for P, which includes the commitment to DB. However, V^* can only commit to at most K DBs for an incarnation of P (and, except with negligible probability, different commitments of DBs correspond to different DBs by the computational binding property of V^*'s commitment scheme SHCom). Therefore, for each incarnation $P^{(i,j)}$ of P, we specify K incarnations of P_{zap}, $P_{zap}^{(i,j_1)}$,..., $P_{zap}^{(i,j_K)}$, one for each consistent protocol class of P with index (i,j).

V^* is given as auxiliary input $\bar{x}$, $\bar{y}$ and $\tilde{y}$. V_{zap}^* then executes V^* internally and instantiates the various incarnations $P^{(i,j)}$ using $\bar{x}$, $\bar{y}$ and $\tilde{y}$ and random tapes as discussed in the hybrid distribution of H_2^i above. Then V_{zap}^* executes the interaction with V^* and the interactions of P internally except that, when V^* sends zap_V to an incarnation of P, V_{zap}^* forwards that message to the corresponding incarnation of P_{zap} (calling the incarnation of P_{zap} with respect both to the determining messages, $\mathsf{zap_V}$ as well as for the commitment to DB, as discussed in the paragraph above), as well as for the commitment to the seed s is sent by P in step 1 of π_2, when R and c are sent by V^* in step 2b of π_2, and when r is sent by P in step 3a of π_2. These messages together specify the language Λ_1. Then:

(a) If the message that V_{zap}^* is forwarding from V^* is for one of the first $s-1$ consistent protocol classes of P, V_{zap}^* sends the corresponding witness from $\tilde{y}$ to P_{zap}, who computes $\mathsf{zap}_{P,1}$ using it and sends $\mathsf{zap}_{P,1}$ to V_{zap}^*. V_{zap}^* forwards this message to V^* for step 3b instead of P's message.

(b) For the l and $l+1$ consistent protocol classes of P, V_{zap}^* forwards both the $\tilde{y}$ and the

PRF witnesses (s, r_2) for each of two classes. Then for one of s and $s+1$, P_{zap} uses the witness for $\tilde{y}$ and for the other, P_{zap} uses the witness, (s, r). P_{zap} then sends these messages to V^*_{zap} who forwards them to their respective internal executions with V^* in place of P's respective messages for $\mathsf{zap}_{P,1}$.

(c) If the message that V^*_{zap} is forwarding is from the $l+2$ and later consistent protocol classes of P, V^* sends the corresponding witness for correct PRF formatting by forwarding s and the randomness r_2 used to commit s in step 1 of π_2 to P_{zap}, who computes $\mathsf{zap}_{P,1}$ using (s, r_2) as witness. V^*_{zap} sends this message to V^* for step 3b instead of P's message.

We note that by construction, the distributions constructed by V^*_{zap} for (π_0, π_2) are distributed identically to the hybrid experiment distributions $H_2^{'l}$ and $H_3^{'l+1}$ because both have the same random tape distributions for the incarnations of P and all zap messages, when computed by P_{zap}, are computed identically to how the "real" incarnations of P with the same randomness would have computed them.

V^*_{zap} executes a distinguisher D_{zap} over his views by running D_{rand} as a subroutine on the views of V^*_{zap} identically distributed as $H_2^{'l}$ and $H_2^{'l+1}$ and outputs 1 when D_{rand} does. Therefore $V*_{zap}$ with D_{zap} can distinguish zap transcripts with non-negligible probability, a contradiction of the rWI property of zaps.

$\mathbf{H_3^{'i}}$. In each of the first first $i = 0, 1, ..., K$ consistent protocol classes of P, P uses a random tape to output r in step 3a of π_2 and uses a witness for $x \in L$ to compute $\mathsf{zap}_{P,1}$. Note that $H_3^{'0}$ is the same as $H_2^{'K}$ and that $H_3^{'K}$ is when P always uses a random string for step 3a of π_2 and the witness for $x \in L$ for step 3b of π_2. Suppose for some hybrids $H_3^{'l}$ and $H_3^{'l+1}$, where verifier V^* interacts with consistent protocol classes $P^{(i,j)}$ with public input x_i, auxiliary input $\bar{y}_i$ and $\tilde{y}_i$, and random tape ω_j, there exists a distinguisher D_{rand} such that D_{rand} distinguishes V^*'s view for $H_3^{'l}$ from V^*'s view for $H_3^{'l+1}$ with non-negligible probability. Then, we reduce to the PRF indistinguishability of $f_*^{P,2}$ as follows. Let D_{PRF} be a distinguisher with oracle access to the F_n, which is uniformly distributed over $f_*^{P,2}$ with

appropriate input length, and R_n, which is uniformly distributed over the set of random functions with the appropriate input length. D_{PRF} is given as input $\bar{x}$, $\bar{y}$, $\tilde{y}$, the random tapes of the $P^{(i,j)}$), denoted $\bar{\omega}$, as well as the auxiliary input of V^*. D_{PRF} executes the $P^{(i,j)}$ and V^* internally and executes as follows:

For each of the first $l-1$ consistent protocol classes of P that V^* calls to interact with, D_{PRF} executes the protocol between the invoked $P^{(i,j)}$ and V^* except that D_{PRF} uses a (consistent across the classes) random string for the incarnations to compute r and uses a witness $x \in L$ to compute $\mathsf{zap}_{P,1}$ and sends it to V^* per the hybrid description.

For the l and $l+1$ consistent protocol classes, D_{PRF} forwards seed s and function input $R|c$ to the oracles F_n and R_n and receives $r^{(1)}$ and $r^{(2)}$. D_{PRF} uses these as the message r in the l and $l+1$ consistent protocol classes, respectively, and otherwise executes the protocol.

Finally, for the each of the last $s+2$ consistent protocol classes of P that V^* calls to interact with, D_{PRF} executes the protocol between the invoked $P^{(i,j)}$ and V^* according to the protocol specification of π_2, e.g. using the witness for $x \in L$ to compute $\mathsf{zap}_{P,1}$.

We note that the internal execution of (π_0, π_1) for hybrid experiments $H_3^{'}$ and $H_3^{'l+1}$ are distributed identically to the external executions distinguished by D_{rand}. Therefore, D_{PRF} executes D_{rand} as a subroutine and distinguishes the oracle calls with non-negligible probability, a contradiction.

$\mathbf{H_4^{'i}}$. Each of the following hybrid experiments are conducted identically to $H_3^{'K}$ except that for each of the first $i = 0, 1, ..., K$ consistent protocol classes of P, the incarnations use a random tape from step 3 of π_0 onwards that is an output of the pseudorandom function $f_{s'}^{P,1}$ on the determining message, where s' is taken from P's initial random tape. Note that $H_4^{'0}$ is identical to $H_3^{'K}$ and that $H_4^{'K}$ is the transcript of the incarnations of P interacting with V^* using its random tape (or a PRF of it) for step 3a of π_2 rather than constructing it according to the PRF. Indistinguishability holds by essentially the same reasoning for hybrids H_1^i above.

8 Proof of Lemma 4.3

Proof of Lemma 4.3. The intuition for the proof is that since (π_0, π_1, π_2') is rZK and since the simulators (Sim' for (π_0, π_1, π_2') and Sim for (π_0, π_1, π_2)) act honestly for π_0, π_2 and π_2' and identically for π_1, there exists a reduction from the rZK of $(\pi_0, \pi_1, \pi_2)$to the rZK of (π_0, π_1, π_2'). This is because π_2 is "independent" of π_1 and (π_0, π_2) and (π_0, π_2') are rWI for L_1' and L_2'. Therefore, if (π_0, π_1, π_2') is rZK then (π_0, π_1, π_2) is also rZK (as the simulator will give P_2 the appropriate auxiliary input, which V^* cannot distinguish.

We will look at three hybrid distributions: The first is where a real player executes the entire protocol (π_0, π_1, π_2); the second is where the player executes the first and third subprotocol but the simulator executes the second subprotocol; and the final hybrid distribution is where the the simulator executes the entire protocol. We will prove that the first and second distributions are computationally indistinguishably by reducing indistinguishability to the rZK property of (π_0, π_1, π_2'). Namely, we will produce a $V^{*\prime}$ and D' for the corresponding hybrid distributions of (π_0, π_1, π_2') that uses V^* and distinguisher D to distinguish the corresponding hybrid distributions. We will rely on the fact that π_0 and π_1 are identically executed by both provers and simulators for (π_0, π_1, π_2) and (π_0, π_1, π_2'), and that, since π_0 is public coin, $V^{*\prime}$ can simulate consistent messages for π_2 given a transcript for π_0.

We will prove that the second and third distributions are computationally indistinguishable by reducing to the rWI property of (π_0, π_2). A formal proof follows.

By denoting (P, V^*) (respectively $(P', V^{*\prime})$) as the composition of interactive protocols (π_0, π_1, π_2) (respectively (π_0, π_1, π_2')) we implicitly denote three pairs of protocols $\{(P_i, V_i^*)\}_{i=0,1,2}$ (respectively, $\{(P_i', V_i^{*\prime})\}_{i=0,1,2}$) that respectively execute the sequential composition. Note that since a simulator Sim (respectively Sim') for (π_0, π_1, π_2) (respectively (π_0, π_1, π_2')) only rewinds V^* (respectively $V^{*\prime}$) on the protocol π_1, Sim can be denoted by the triple $(\mathsf{Sim}_0, \mathsf{Sim}_1, \mathsf{Sim}_2)$ (respectively Sim' can be denoted by the triple $(\mathsf{Sim}'_0, \mathsf{Sim}'_1, \mathsf{Sim}'_2)$).

We consider the following three hybrid distributions: (P_0, P_1, P_2) (respectively (P_0, P_1, P_2')) is the distribution of the view of V^* (respectively $V^{*\prime}$) interacting with the real (incarnations of) player P for each of π_0, π_1, and π_2 (respectively player P' for each

of π_0', π_1', and π_2'); $(P_0, \mathsf{Sim}_1, P_2)$ (respectively $(P_0, \mathsf{Sim}_1', P_2')$) is the distribution of the view of V^* (respectively $V^{*\prime}$) interacting with the real player P (respectively P') for all rounds of π_0 and π_2 (respectively π_0 and π_2') and with the simulator Sim (respectively Sim') for all rounds of π_1; finally, $(\mathsf{Sim}_0, \mathsf{Sim}_1, \mathsf{Sim}_2)$ (respectively $(\mathsf{Sim}_0', \mathsf{Sim}_1', \mathsf{Sim}_2')$) is the distribution of V^* (respectively $V^{*\prime}$) interacting with the simulator for the whole protocol.

We note that, as specified in the lemma statement, Sim is actually the sequential composition of algorithms P_0, $\mathcal{S}$, and P_2 but we choose to write the distribution of V^* interacting with Sim as $(\mathsf{Sim}_0, \mathsf{Sim}_1, \mathsf{Sim}_2)$ in order not to confuse the reader.

(P_0, P_1, P_2) *is computationally indistinguishable from* $(P_0, \mathsf{Sim}_1, P_2)$*:* We reduce from a verifier V^* and a distinguisher D that can distinguish (P_0, P_1, P_2) from $(P_0, \mathsf{Sim}_1, P_2)$ with non-negligible probability to a verifier $V^{*\prime}$ for (π_0, π_1, π_2') and a distinguisher D' that can distinguish (P_0, P_1, P_2') from $(P_0, \mathsf{Sim}_1', P_2')$ with non-negligible probability as follows:

For (π_0, π_1, π_2'), we fix a prover P' by fixing $x_1, ..., x_{poly(n)} \in L$, witnesses $y_1, ..., y_{poly(n)}$ such that $y_i \in W_L(x_i)$, and random tapes $w_1', ..., w_{poly(n)}'$ that completely specify the various incarnations $P'^{(i,j)}$ of P'. We let the verifier $V^{*\prime}$ for (π_0, π_1, π_2') have as auxiliary input the same $x_1, ..., x_{poly(n)} \in L$ and witnesses $y_1, ..., y_{poly(n)}$ as well as any auxiliary inputs given to V^* from the pair (V^*, D) above.

(a) $V^{*\prime}$ executes π_0 and π_1 with P' by running, for any of P''s incoming messages, V^* internally with inputs V^*'s auxiliary inputs and previous transcript messages for π_0, π_1 and π_2 (as described below for π_2), to determine the messages to be sent. In such a manner, $V^{*\prime}$ executes π_0 and π_1 by passing messages between external P' and internal V^*.

(b) Whenever, on input from $V^{*\prime}$, V^* calls for a message to be sent to the (i, j) incarnation of P for π_2, $V^{*\prime}$ internally responds to V^*'s output message acting as $P^{(i,j)}$. $V^{*\prime}$ instantiates $P^{(i,j)}$ with input x_i, y_i, w_j, where randomness w_j is constructed such that the first component of w_j equals the public coin π_0 output of w_j' for $P'^{(i,j)}$ and the rest is chosen uniformly at random, but then forever fixed for the pair (i, j). Then,

per property 7 of the definition of near-compatible protocols, the pseudorandom pair (ω_1', ω_2') consisting of random tapes for players to be used for π_1 and π_2/π_2' can then be computed once the determining message has been sent by $V^{*\prime}$ in π_0. Note that the auxiliary input of a "real" $P^{(i,j)}$ is y_i, which $V^{*\prime}$ possesses. $V^{*\prime}$ also provides as input to $P^{(i,j)}$ the transcript for that session's public interactions for π_0 and for the view of $P^{(i,j)}$ for π_2 thus far in the session. Since by construction, $P'^{(i,j)}$ acts identically to $P^{(i,j)}$ for π_0 and by property 8 of Definition 4.2, $P^{(i,j)}$ only needs $View_{P_0}(\pi_0)$ as input for π_2 (together with y_i), this is an identically distributed instantiation of a "real" $P^{(i,j)}$.

(c) $V^{*\prime}$ never interacts with $P'^{(i,j)}$ for π_2' because V^*, executed internally, never calls for such a message.

If V^* executes in probabilistic polynomial-time, then so does $V^{*\prime}$. Further, we claim that the view of $V^{*\prime}$ interacting with the $P'^{(i,j)}$s includes the distribution consistent with the view of V^* interacting with the $P^{(i,j)}$s (i.e., the filtering of components of $V^{*\prime}$'s view are distributed computationally indistinguishably from the view of V^* interacting with the $P^{(i,j)}$s). To prove this claim, it must be demonstrated that the output of $V^{*\prime}$ acting as $P^{(i,j)}$ for π_2 using as transcript its interaction with $P'^{(i,j)}$ is indistinguishably distributed from how the 'real' $P^{(i,j)}$ would interact with V^* for π_2. From the definition of near-compatible protocols, since the P_0 component of $P^{(i,j)}$ is public coin and P_2's output is determined only by the transcript of P_0 with V^*, instantiating w_j as given in step 2 above together with x_i, y_i is a uniformly distributed instantiation of a $P^{(i,j)}$ executing π_2 given executions of π_0 and π_1, and the claim follows.

We construct a distinguisher D' for (P_0, P_1, P_2') and $(P_0, \mathsf{Sim}'_1, P_2')$ that executes D as a subprotocol on the distribution consistent with V^*'s view for (π_0, π_1, π_2). Since Sim_1 acts identically to Sim'_1 for π_1 (indeed, by hypothesis both are the PPT algorithm $\mathcal{S}$), the distribution that D' provides as input to D is distributed computationally indistinguishably from the distributions (P_0, P_1, P_2) and $(P_0, \mathsf{Sim}_1, P_2)$. This implies by our initial assumption for this case that D can distinguish (P_0, P_1, P_2) from $(P_0, \mathsf{Sim}_1, P_2)$ with probability

polynomially related to the non-negligible probability that D' can distinguish (P_0, P_1, P_2') from $(P_0, \mathsf{Sim}'_1, P_2')$. However, since (π_0, π_1, π_2') is rZK, no such pair $V^{*\prime}, D'$ can exist, yielding a contradiction.

$(P_0, \mathsf{Sim}_1, P_2)$ is computationally indistinguishable from $(\mathsf{Sim}_0, \mathsf{Sim}_1, \mathsf{Sim}_2)$:

Suppose that the two distributions are not computationally indistinguishable. We reduce to the rWI of (π_0, π_2) together with the fact that Sim executes π_0 and π_2 honestly. We note that Sim and real P execute π_2 with different witnesses; in particular, P_2 uses a witness for $x \in L$ while Sim_2 uses a witness for $y \in L_2$. Sim cannot, except with negligible probability use a witness for $x \in L$ because Sim' acts identically to Sim on the only messages that it does not execute honestly for, namely π_1, and (π_0, π_1, π_2') has the soundness property for L.

Suppose for V^* there existed sequence of $poly(n)$ public inputs $\bar{x}$ and a sequence of $poly(n)$ auxiliary inputs $\bar{y}$ for player P such that for $1 \leq i \leq poly(n)$, $\bar{y}_i \in W_L(\bar{x}_i)$ and such that for V^* interacting with P and Sim, there existed a distinguisher D that could distinguish $(P_0, \mathsf{Sim}_1, P_2)$ from $(\mathsf{Sim}_0, \mathsf{Sim}_1, \mathsf{Sim}_2)$. Then we consider a verifier V^*_{WI} for protocol (π_0, π_2) and external honest prover P_{WI} such that V^*_{WI} has as auxiliary input $\bar{x}$, $\bar{y}$ and also an additional sequence of auxiliary inputs $\tilde{y}$ such that $\tilde{y}$ contain appropriate witnesses for language L_1. We let the incarnations of P_{WI} have as input $\bar{x}_i$, and *either* $\bar{y}_i$ or $\tilde{y}_i$, as described below. V^*_{WI} will execute V^* internally and will execute P and Sim internally in the form of $\mathcal{S}$ in the following way:

(a) Executing V^* internally, for every incarnation (i, j) that V^* wishes to communicate with, V^*_{WI} queries the incarnation $P^{(i,j)}_{WI}$ who sends his initial message. V^*_{WI} forwards this message to V^*, who responds to V^*_{WI} with its message, which is the determining message.

(b) For π_1, V^*_{WI} picks new randomness and instantiates $\mathcal{S}$ with this new randomness as well as $\bar{x}$ and the public transcript for the current execution of π_0, together with the following: For the public coin message $P^{(i,j)}_{WI}$ sent in π_0, form the random string ω such that ω begins with the message sent by P_{WI} and otherwise is chosen uniformly

at random. Then using ω and f^{PRF} per specification of π_0, form ω_1 (choosing the PRF seed using ω per specification). Instantiate $\mathcal{S}$ with inputs ω and ω_1 in addition to previous inputs specified, which therefore consists of a complete view of $\mathcal{S}$ executing π_0 as it does so honestly as given in lemma statement. Execute π_1 between $\mathcal{S}$ and V^* in this fashion, forwarding messages from one to the other and storing V^*'s view of the interaction.

(c) For π_2, P_2 does not use the transcript of π_1 for its execution; therefore, V^*_{WI} carries out π_2 with $P^{(i,j)}_{WI}$ by continuing the internal execution of V^* and passing messages back and forth between $P^{(i,j)}_{WI}$ and V^*. Note that the randomness ω_2 used by P_{WI} for π_2 is computed internally from P_WI's execution of π_0. Then we have the following cases:

Case 1: P_{WI} only uses $\bar{y}$ as witnesses for the rWI proof (π_0, π_2).

Case 2: P_{WI} only uses $\tilde{y}$ as witnesses for the rWI proof (π_0, π_2).

We note that the two distributions for (π_0, π_1, π_2) constructed by V^*_{WI} corresponding to the two cases above are identically distributed to the distributions of $(P_0, \mathsf{Sim}_1, P_2)$ and $(\mathsf{Sim}_0, \mathsf{Sim}_1, \mathsf{Sim}_2)$. This is because, as noted, Sim executes P_2 with witness for L_2, executes P_0 honestly which doesn't rely on auxiliary inputs, and $\mathcal{S}$ only uses π_0 interactions as input as Sim never resets the verifier for protocol π_2. Therefore, V^*_{WI} uses a distinguisher D_{WI} that executes D as a subroutine on the distributions for (π_0, π_1, π_2) and outputs 1 when D does. This implies that D_{WI} can distinguisher rWI transcripts using different witnesses, which is a contradiction and completes the proof. □

9 Proof of Lemma 5.3

Proof of Lemma 5.3. We proceed with a series of hybrid experiments and prove that (π_0, π''_2) is rWI by reductions to the computational hiding property of CHCom as well as the rWI property of zaps. We will also need that τ is distributed pseudorandomly, which we will reduce to the rWI of zaps as well as the property of pseudorandom functions. We note that $\mathsf{zap}s$ are not inherently rWI but do have the property when the prover message is

computed using a random tape that is a PRF output of prover's initial randomness as well as verifier's determining message (discussed below). A formal proof follows.

We fix a verifier strategy V^*. For a sequence $\bar{x} = x_1, ..., x_{poly(n)}$, we define a corresponding sequence of witnesses $\bar{y} = y_1, ..., y_{poly(n)}$ such that $\bar{y}_i \in W_L(\bar{x}_i)$ for $1 \leq i \leq poly(n)$. For an ensemble of databases $\bar{DB} = DB^1, ..., DB^{poly(n)}$, a corresponding ensemble of witnesses $\bar{\gamma} = \gamma^1, ..., \gamma^{poly(n)}$ is constructed such that at least $1/3$ of γ^i is equals the corresponding entries in DB^i (note that it is possible that $DB^i = DB^j$ for $i \neq j$). We note that while the existence of such a witness set is highly unlikely, all that matters is that a distinguisher cannot distinguish cases when a prover, with unbelievable luck, has such a witness set and uses it to complete the protocol (π_0, π_2'). In what follows, we will assume that $V^{*\prime}$ tries to distinguish between when P' uses a witness for $\bar{x}$ and when P' uses a witness for DB to complete (π_0, π_2'). Since the language Λ_3 is specified with a perfect binding commitment to a guess for P' to DB, there can only ever be one witness for DB in a particular execution of (π_0, π_2') for Λ_3. While $V^{*\prime}$ must also be able to distinguish the case where P' uses different witnesses for $\bar{x}$, the proof for this case is strictly simpler than for the case we consider here (this is because no hybrid distribution for commitments are needed and otherwise the proof executes nearly identically to the one given here).

Let K be the total (maximum) number of session resets that $V^{*\prime}$ will require[14] (if such a later session doesn't occur, neither P nor V^* will output anything). Therefore, K is also a bound on the number of incarnations of P' interact with. We denote by the *determining message* the sequence of public messages sent by $V^{*\prime}$ in step 1 of π_0. We denote by a *consistent protocol class* of P' to be the set of incarnations of P' interacting with $V^{*\prime}$ that all have the same index (i, j) and all receive the same determining message. We note that, except with negligible probability, V^* sending the same distinguishing message as implies that V^* committed to the same DB before, since V^* commits to DB with a computationally binding commitment scheme and V^* is a PPT algorithm.

[14]Recall that such a bound is explicit for any verifier strategy $V^{*\prime}$ per the definition given about for rWI.

We specify a sequence of hybrid games.

$\mathbf{H_0}$ In this hybrid experiment, all incarnations of P' with $V^{*\prime}$ execute honestly, using the witnesses $\bar{y}$ for $\bar{x}$.

$\mathbf{H_1^i}$ For each of the first $i = 0, 1, ..., K$ consistent protocol classes of P', P' uses a (consistent over protocol class) uniformly random tape throughout the protocol to execute π_2' instead of using a random tape from step 3 of π_0 onwards that is an output of the pseudorandom function $f_{s'}^{P,1}$, where s is taken from P''s initial random tape. For each of the remaining consistent protocol classes, continue to execute the protocol per π_0, using the PRF $f_*^{P,2}$ to construct the random tape for π_2'. Note that H_1^0 is the same experiment as H_0.

Suppose for some hybrids H_1^s and H_1^{s+1}, where verifier $V^{*\prime}$ interacts with prover incarnations $P'^{(i,j)}$ with public input x_i, auxiliary input y_i, and random tape ω_j, there exists a distinguisher D_{WI} such that D_{WI} distinguishes $V^{*\prime}$'s view for H_1^s from $V^{*\prime}$'s view for H_1^{s+1} with non-negligible probability. Then we reduce to distinguishability of the PRF $f_*^{P,1}$. Let D_{PRF} be a distinguisher with oracle access to the oracle F_n, which is uniformly distributed over $f_*^{P,1}$ with appropriate input length, and the oracle R_n, which is uniformly distributed over the set of random functions with the appropriate input length. D_{PRF} is given as input $\bar{x}$, $\bar{y}$, the random tapes of the $P^{(i,j)}$, denoted $\bar{\omega}$, as well as the auxiliary input of $V^{*\prime}$. D_{PRF} executes the $P'^{(i,j)}$ and $V^{*\prime}$ internally and executes as follows: For each of the first $s-1$ consistent protocol classes of P' that $V^{*\prime}$ calls to interact with, D_{PRF} executes the protocol between the invoked $P^{(i,j)}$ and $V^{*\prime}$ using $P^{(i,j)}$'s random tapes (one for π_0, one for π_2', per specification) throughout the interaction.

For the s and $s+1$ consistent protocol classes of P' invoked by $V^{*\prime}$, denote by ω_s and ω_{s+1} the random tapes of these incarnations. D_{PRF} executes the interaction between $V^{*\prime}$ and these consistent protocol classes as follows: Per specification of π_0, the respective incarnations of P' and $V^{*\prime}$ interact per steps 1 and 2 of π_0. Finally, for step 3 of P_0, D_{PRF} sends $(Com_{SH}, \mathsf{zap}_V)$, the random tapes of the respective incarnations and the PRF random

seeds used by the respective incarnations to the oracles and receives two strings, ω_{P_s} and $\omega_{P_{s+1}}$ D_{PRF} executes the rest of the interaction of the s consistent protocol class of P' using as random tape ω_{P_s} and the $s+1$ consistent protocol class of P' using as random tape $\omega_{P_{s+1}}$. Note that one of these hybrids is using a random tape that is a PRF output and the other is using a random tape that is the output of a random function.

Finally, for the each of the last $s+2$ consistent protocol classes of P' that $V^{*\prime}$ calls to interact with, D_{PRF} executes the protocol between the invoked $P^{(i,j)}$ and $V^{*\prime}$ according to the protocol specification of π_0, e.g. using for π_2' the random tape output by step 2 of π_0.

We note that the internal execution of (π_0, π_2) for hybrid experiments H_1^s and H_1^{s+1} are distributed identically to the external executions distinguished by D_{WI}. Therefore, D_{PRF} executes D_{WI} as a subroutine and distinguishes the oracle calls with non-negligible probability, a contradiction.

$\mathbf{H_2^i}$ Each of the following hybrid experiments are conducted identically to H_1^K except that for each of the first $i = 0, 1, ..., K$ consistent protocol classes of P, the following holds. In the lth consistent protocol class, $1 \leq l \leq i$, P' in step 1 of π_2' commits to a set γ^l, 1/3 of γ^l corresponds to DB^l, where DB^l is the DB committed by $V^{*\prime}$ in step 2b of π_0 of the l consistent protocol class; note that the commitment to DB^l is part of the determining message. All other consistent protocol classes commit to γ per GamGen in step 1 of π_2'. Note that H_2^0 is identical to H_1^K.

Suppose for some hybrids H_2^s and H_2^{s+1}, where verifier $V^{*\prime}$ interacts with consistent protocol classes $P'^{(i,j)}$ with public input x_i, auxiliary input y_i, and random tape ω_j, there exists a distinguisher D_{WI} such that D_{WI} distinguishes $V^{*\prime}$'s view for H_2^s from $V^{*\prime}$'s view for H_2^{s+1} with non-negligible probability. We wish to reduce to the hiding property of the commitment scheme. Consider the malicious receiver R^* and honest committer C for the corresponding 1-round perfectly binding commitment scheme executing CHCom and CHVer. R^* is given as auxiliary input $\bar{x}$ and $\bar{y}$. R^* executes $V^{*\prime}$ internally and constructs the incarnations $P'^{(i,j)}$ for the corresponding $P'^{(i,j)}$ that $V^{*\prime}$ interacts with with the same x_i and

y_i. Each consistent protocol class is given the same two random tapes, one for π_0 and one for π_2'. All incarnations with the same index have first tape identical to each other as well. Then R^* executes internally, with $V^{*\prime}$ and each of the incarnations of P' for the first $s-1$ consistent protocol classes, where each of the incarnations commit to their respective DB guesses γ^l, $1 \leq l \leq s-1$.

Then, R^* sends γ^s and the 0 set computed per GamGen to C. C sends back com_1 and com_2. R^* executes the s and $s+1$ consistent protocol classes of P' that interact with $V^{*\prime}$ using as message for step 1 of π_2' com_1 for the s consistent protocol class and com_2 for the $s+1$ consistent protocol class.

For the remaining consistent protocol classes, R^* uses 0 set computed per GamGen to execute step 1 of π_2'. We note that by construction, the distributions constructed by R^* for (π_0, π_2') are distributed identically to the hybrid experiment distributions H_2^s and H_2^{s+1} because both have the same random tape distributions for the incarnations of P' and all commitments, whether computed internally by R^* or externally by C, are computed as the "real" incarnations of P' would have computed them. R^* executes a distinguisher D_{COM} for his views by executing D_{WI} as a subroutine on the views of R^* identically distributed as H_2^s and H_3^{s+1} and outputs 1 when D_{WI} does. Therefore R^* with D_{COM} can distinguish commitments with non-negligible probability, a contradiction of the computational hiding property of the commitment scheme.

$\mathbf{H_3^i}$. Each of the following hybrid experiments are conducted identically to H_2^K except that for each of the first $i = 0, 1, ..., K$ consistent protocol classes of P', in the lth consistent protocol class, $1 \leq l \leq i$, uses as witness γ^l instead of the $\bar{y}$ witness for $\bar{x}$ for $\mathsf{zap}_{P,3}$ in step 3b of π_2'. Note that H_3^0 is the same experiment as H_2^K.

Suppose for some hybrids H_3^s and H_3^{s+1}, where verifier $V^{*\prime}$ interacts with consistent protocol classes $P^{(i,j)}$ with public input x_i, auxiliary input y_i, and random tape ω_j, there exists a distinguisher D_{WI} such that D_{WI} distinguishes $V^{*\prime}$'s view for H_3^s from $V^{*\prime}$'s view for H_3^{s+1} with non-negligible probability. We wish to reduce to the rWI property of zap

protocols. Consider a malicious ${V^{*\prime}}_{zap}$ and honest P_{zap} for a zap proof system.

We remark on how the P_{zap} incarnations are called by ${V^{*\prime}}_{zap}$. We must be careful with specifying the incarnations of P_{zap} as the incarnations of P^* are specified both by their indices and their determining message; but the determining message for P^*_{zap}, namely zap_V, is different than the determining message for P', which includes the commitment to DB. However, $V^{*\prime}$ can only commit to at most K DBs for an incarnation of P' (and, except with negligible probability, different commitments of DBs correspond to different DBs by the computational binding property of $V^{*\prime}$'s commitment scheme SHCom). Therefore, for each incarnation $P^{(i,j)}$ of P', we specify K incarnations of P_{zap}, $P^{(i,j_1)}_{zap}$,..., $P^{(i,j_K)}_{zap}$, one for each consistent protocol class of P' with index (i,j) that V^* may execute internally.

$V^{*\prime}$ is given as auxiliary input $\bar{x}$ and $\bar{y}$ as well as the witnesses γ^l for each of the l consistent protocol classes, $1 \leq l \leq K$. ${V^{*\prime}}_{zap}$ then executes $V^{*\prime}$ internally and instantiates the various incarnations $P^{(i,j)}$ in the same manner as specified by R^* in the above hybrid distribution proof. Then ${V^{*\prime}}_{zap}$ executes the interaction with $V^{*\prime}$ and the interactions of P' internally except that, when $V^{*\prime}$ sends zap_V and a commitment of DB (e.g. step 1 of π_0, which is the determining message) to an incarnation of P', ${V^{*\prime}}_{zap}$ forwards that message to the corresponding incarnation of P_{zap}.

When the language Λ_3 is specified, ${V^{*\prime}}_{zap}$ forwards the language description to the corresponding incarnation of P_{zap}; in particular the committed DB guess by P', μ, is sent in step 1 of π_2, and the corresponding decommitments to DB are sent in step 2 of π_2. Then:

(a) If the message that ${V^{*\prime}}_{zap}$ is forwarding is from the first $s-1$ consistent protocol classes of P', ${V^{*\prime}}_{zap}$ sends the corresponding γ^l and randomness used to compute μ to P_{zap}, who computes $\mathsf{zap}_{P,3}$ using it and sends $\mathsf{zap}_{P,3}$ to V^*_{zap}. ${V^{*\prime}}_{zap}$ forwards this message to $V^{*\prime}$ for step 3b of π'_2 instead of P''s message.

(b) For the s and $s+1$ consistent protocol classes of P', ${V^{*\prime}}_{zap}$ forwards y_s and γ^s (and random string used for μ) and y_{s+1} and γ^{s+1} (and random string used for μ), respectively. Then for one of s and $s+1$, P_{zap} uses the witness for $\bar{x}$ and for the other, P_{zap} uses the witness, γ, for DB. P_{zap} then sends these messages to ${V^{*\prime}}_{zap}$ who forwards

them to their respective internal executions at step 3b of π_2' with $V^{*\prime}$ in place of P's respective messages for $\mathsf{zap}_{P,3}$.

(c) If the message that ${V^{*\prime}}_{zap}$ is forwarding is from the $s+2$ and later consistent protocol classes of P', $V^{*\prime}$ sends the corresponding y_l to P_{zap}, who computes $\mathsf{zap}_{P,3}$ and sends it to V^*_{zap}. ${V^{*\prime}}_{zap}$ sends this message to $V^{*\prime}$ for step 3b of π_2' instead of P''s message.

We note that by construction, the distributions constructed by ${V^{*\prime}}_{zap}$ for (π_0, π_2') are distributed identically to the hybrid experiment distributions H_3^s and H_3^{s+1} because both have the same random tape distributions for the incarnations of P' and all zap messages, when computed by P_{zap}, are computed identically to how the "real" incarnations of P' with the same randomness would have computed them.

${V^{*\prime}}_{zap}$ runs a distinguisher D_{zap} over his views by running D_{WI} as a subroutine on the views of ${V^{*\prime}}_{zap}$ identically distributed as H_3^s and H_3^{s+1} and outputs 1 when D_{WI} does. Therefore $V*_{zap}$ with D_{zap} can distinguish zap transcripts with non-negligible probability, a contradiction of the rWI property of zaps.

$\mathbf{H_4^i}$. Each of the following hybrid experiments are conducted identically to H_3^K except that for each of the first $i = 0, 1, ..., K$ consistent protocol classes of P', the incarnations use a random tape from step 3 of π_0 onwards that is an output of the pseudorandom function $f_{s'}^{P,1}$ on the determining message, where s' is taken from P''s initial random tape. Note that H_4^0 is identical to H_3^K and that H_4^K is the transcript of the incarnations of P' interacting with $V^{*\prime}$ using the witness for DB instead of $x \in L$. Indistinguishability of H_4^s from H_4^{s+1} follows by essentially the same reasoning as was given for the indistinguishability of H_1^s from H_1^{s+1}.

Remark 9.1. Lemma 5.3 has most of the reasoning needed to show that (π_0, π_1, π_2') is rWI for Λ_3.

The only two facts to point out is that incarnations of P_1 can be constructed for external players P_0 using the same strategy as used to construct internal incarnations of P_2', namely by using the fact that P_0 is public coin, extending the random string and using the PRF

$f^{P,1}$ together with the extended random string. Further, none of the hybrids above would change as P_1 simply sends random strings to V^*, so nothing would change and the fact that the distributions are identical, when π_1 is included, wouldn't be affected. Therefore, we have the following corollary:

Corollary 9.2. *(π_0, π_1, π_2') is rWI for Λ_3.*

10 Simulator Specification and Proof of Theorem 5.1

As the protocol (π_0, π_1, π_2') is admissible, we can construct our simulator in the hybrid model (see Definition 2.14). We make a few simplifications without loss of generality. We assume that V^* does not make invalid decommitments. We note that by definition, the simulator is given the maximum number of sessions $\delta(n) = K$. Without loss of generality, we will assume that V^* opens exactly K sessions in each run.

Finally, we note that the simulator Sim' will execute the composition $(P_0, \mathcal{S}, P_2')$ with V^* where the goal of $\mathcal{S}$ is to output to P_2' an auxiliary input that correspond to at least 1/3 of V^*'s DB.

10.1 Notation and Variables

By an *iteration* of π_1, we mean a single back-and-forth message execution of steps 1a and 1b (in that order). In what follows, we will require the following basic notation. Say that a message is an *iteration-j (verifier) message* if it is a valid decommitment in iteration j of some session i. Sessions are addressed via their *session identifier*. A section identifier s includes the first three messages in the session (namely, the transcript of π_0). This includes the prover's initialization message, and the verifier's commitment to DB. A session in an executed prefix is *completed*, if, within this execution prefix, the verifier has decimated properly in all its n^ϵ iterations. A session is *solved* if the simulator obtains 1/3 of the verifier's DB. We denote by (i) a string of n entries of DB. $DB_{(i)}$ is a sequence of n entries of DB corresponding to (i). $(DB_{(i)}, (i))$ is the sequence of the n pairs of entries of $DB_{(i)}$ with its corresponding

index in (i). We will frequently refer to (i) as an index even though it is in fact a collection of indices.

We also will require the following variables. K is the number of possible distinct sessions that V^* can instantiate (in the hybrid model). In particular, we set $K = max\{\delta(n), n\}$. m is the number of parallel look-ahead subprotocols that the simulator will execute. We set $m = |DB|$. l is a local variable bounding the number of recursive calls of Simulate; initially, $l = L = 2\lceil \log_{\frac{k}{2^{14}}} K \rceil$. h is a local variable holding the current (prefix of) the execution history of V^*; initially, h holds the common input x. We require a sequence of global variables $S_1, ..., S_m$, each holding a set of triplets (s, j, DB_i, i). Initially, all S_a are empty. Each S_a consists of "identified database elements". If $(s, j, DB_{(i)}, (i)) \in S_a$, then s is a session identifier, j is a numeral of an iteration within session s, and $(DB_{(i)}, (i))$ is the sequence of n pairs corresponding to the index (i) of n elements in V^*'s DB. We set $S = \cup_{j=1}^{m} S_j$. We require a sequence of global variables $SS_1, ..., SS_m$ holding a set of solved sessions. Initially, each SS_j is empty. We set $SS = \cap_{j=1}^{m} SS_j$. a is the counter of parallel look-ahead runs. Initially, $a = 0$. We will call a the *multi-extraction value*.

To more concretely compare the simulator of [16] with our own, almost all of this notation is identical to that used in the [16] simulator specification, but in [16], $\mathcal{S}$ tries to solve once for a single session and store the commitment information into S_1 and then stores the solved session IDs into SS_1. Here, $\mathcal{S}$ tries to solve the same session for each S_i, and therefore makes m independent attempts to solve and input the DB information into each of $S_1, ..., S_m$ (independently). Only if the session has been solved in each of the m attempts will the session ID be input into SS.

10.2 Simulator $\mathcal{S}$ Specification

Simulator $\mathcal{S}$:

(1) Let $h \leftarrow \mathsf{Simulate}(L, 0, h)$.

(2) If the last message in h is a halt message of the verifier then output h. Otherwise the last message in h is fail, and output fail.

Procedure $\mathsf{Simulate}(l, a, h)$: Here l is the level of recursion, h is the prefix of a transcript at a point where V^* is expected to provide the next message, and a is the multi-extraction value. Iterate the following instructions:

(1) Extend the transcript by one verifier message:

  (1) Set $\mathsf{v}-\mathsf{msg} \leftarrow V^*(h)$ and $h \leftarrow (h, v-msg)$.

  (2) If $\mathsf{v}-\mathsf{msg} = \mathsf{halt}$ (i.e. verifier terminates), then return h.

  (3) If $a \neq 0$ and if $\mathsf{v}-\mathsf{msg}$ is a decommitment to $DB_{(i)}$ in iteration j of session s, then set $S_a \leftarrow S_a \cup \{(s, j, DB_{(i)}, (i))\}$.

(2) Extend the transcript by one prover message:

  (1) Set $\mathsf{p}-\mathsf{msg} \leftarrow \mathsf{NextProverMsg}(l, a, h)$ and $h \leftarrow (h, \mathsf{p}-\mathsf{msg})$.

  (2) If $\mathsf{p}-\mathsf{msg} = \mathsf{fail}$, then return h. Otherwise, continue the simulation by going back to Step 1.

Procedure $\mathsf{NextProverMsg}(l, a, h)$: Extract the last message in h, which is a verifier message denoted $\mathsf{v}-\mathsf{msg}$, and proceed as follows.

(1) If V^* wants to start a new session or the message is for π_0, run π_0 with V^* honestly until arrival at π_1, then go to step 3.

(2) If protocol is in π_2, run π_2 with V^* honestly, with auxiliary input y as defined in Solve step 1(2).

(3) *Iteration-j message in session s* of π_1:

- If $l = L$:

  (1) If $s \in SS$:

    (a) If $j = n^{\epsilon}$, return yes.

    (b) If $j < k$ then return a random (i) of DB.

  (2) If $s \notin SS$ and if $j = n^{\epsilon}$, return fail.

  (3) Else for each a such that $s \notin SS_a$:

    (a) Set $S'_a \leftarrow \mathsf{Solve}(L-1, a, h, s, j)$. If S'_a contains $(s, j+1, DB_{(i)}, (i))$ for some (i) then add s to SS_a and return uniformly random (i').

- If $l < L$:

  (1) If $s \in SS_a$:

    (a) If $j = n^\epsilon$, then return yes.

    (b) If $j < n^\epsilon$ then return a random index of DB.

  (2) If $s \notin SS_a$.

    (a) If $j = n^\epsilon$, then return fail.

    (b) Else, if $(s, j+1, DB_{(i)}, (i)) \in S_a$ then add s to SS_a and return (i).

    (c) Else (i.e. if $j < k$ and $(s, j+1, DB_{(i)}, (i)) \notin S_a$):

      1. If $l > 0$, then set $S'_a \leftarrow \mathsf{Solve}(l-1, a, h, s, j)$. If S'_a contains $(s, j+1, DB_{(i)}, (i))$ for some (i) then add s to SS_a and return uniformly random (i'). Otherwise, return a random index (i) of DB.

      2. If $l = 0$, then return a random index (i) of DB.

Procedure $\mathsf{Solve}(l, a, h, s, j)$: Set $b = 1$. Next:

(1) Make up to 128K attempts to solve session s at iteration j. That is, as long as $b \leq 128K$, do:

  (1) Run $\mathsf{Simulate}(\mathsf{l}, \mathsf{a}, \mathsf{h})$.

  (2) If $s \in SS$, then return all $(s, j, DB_{(i)}, (i)) \in S$ and construct auxiliary input y as the set of $16tn(n^\epsilon + 1)$ strings of length n that correspond in the jth entry with the various $(DB_{(i)}, (i))$ and in all other entries consist of 0^n. Otherwise, set $b \leftarrow b + 1$ and go back to Step 1 (i.e. continue the attempt).

(2) Return $\emptyset$.

10.3 Proof of Theorem 5.1

In what follows, we will follow closely the proof for the simulator in [16]. We make some initial remarks. First, we note that Solve never updates h, even thought Simulate may update h; in particular, all invocations of Simulate start with the same value of h. Second, the simulator will only halt when V^* halts within the main procedure; simulation does not stop even when V^* calls for a halt within recursive calls because the output may skew the simulator output.

Lemma 10.1. *$\mathcal{S}$ runs in (worst-case) polynomial time.*

Proof. We say that an invocation of Simulate (respectively an invocation of NextProverMessage or of Solve) is at level l if it is called with first parameter are l. Let $T_{NPM}(l, a, n)$ (respectively, $T_{SOLVE}(l, a, n)$) denote the worst case running time of NextProverMsg (respectively Solve) at level l, multi-extraction value a and with security parameter n. We assume, for simplicity, that V^* runs in worst-case polynomial time in n. Recall that $L = 2\lceil \log_{\frac{k}{2^{14}}} K \rceil$. Then we have the following claims

(1) The running time of the main procedure is at most $poly(n) \cdot T_{NPM}(L, 0, n)$;

(2) $T_{NPM}(L, 0, n) < m(n) \cdot T_{SOLVE}(l-1, a, n) + poly(n)$

(3) For $l < L$ and $1 \leq a \leq m(n)$, $T_{NPM}(l, a, n) < T_{SOLVE}(l-1, a, n) + poly(n)$;

(4) $T_{SOLVE}(l, a, n) \leq poly(n) \cdot T_{NPM}(l, a, n)$;

(5) $T_{SOLVE}(0, a, n) = poly(n)$.

From these claims, we obtain that $T_{NPM}(L, 0) \leq n^{O(L)}$ and the running time of the main procedure is $poly(n)$ as long as $K = k^{O(1)}$.

To justify the claims: The first follows as $\mathcal{S}$ has run time dominated by $O(n^\epsilon)$ calls to NextProverMsg$(L, 0, n)$. The second and third claims follow from the fact that NextProverMsg makes at most $m(n)$ calls to Solve at level L and one call to Solve at level $l < L$ for any value of a, with additional overhead due to various polynomial time verifications/ storage activities. The fourth claim follows from the fact that Solve makes $128K$ calls to Simulate, which in turn executes a polynomial ($O(n^\epsilon)$) number of calls to NextProverMsg at the same recursive levels and K is polynomial in n. The final claim follows from the fact that NextProverMsg at the bottom recursive level does not execute any look aheads. □

In order to prove that the output of $\mathcal{S}$ is correctly distributed, we have to show that it satisfies two lemmas:

Lemma 10.2. *Conditioned on the event that $\mathcal{S}$ does not output* fail*, the distribution of V^*'s view of its interaction with P is computationally indistinguishable from the distribution of the output of* Sim′*; in particular V^*'s view of its interaction with $\mathcal{S}$ is indistinguishable from its view of its interaction with P_1.*

Lemma 10.3. *$\mathcal{S}$ outputs* fail *with negligible probability.*

In order to prove Lemma 10.2, the following technical lemma will be needed.

Lemma 10.4. *Let X be an ordered set of $q(n)$ elements. Let p be a polynomial and construct the sets S_1,...,$S_{p(n)}$, where $|S_k| = t(n)$ for some polynomial t, by constructing each $S_k = \{(i_1, ..., i_n) : i_j \in [q(n)],\ i_j$ chosen indep. and uniformly at random$\}$. Then, for large enough p, the probability that there exists an element $x \in S_1 \times ... \times S_{p(n)}$ such that every index in x is contained in some $D \subset [q(n)]$ where $|D| = q(n)/2$ is negligible in n. In other words, except with negligible probability (over the random choices to construct each of the S_i's), every $x \in S_1 \times ... \times S_{p(n)}$ describes a set of strictly more than $q(n)/2$ district indices of X.*

Remark 10.5. Lemma 10.4 proves that even if V were to pick the indices that $\mathcal{S}$ succeeds on to fill S_1,..., $S_{|DB|}$, V *still* must reveal at least half of the DB entries (when $m = |DB|$); in particular, $\mathcal{S}$ will have more than $|DB|/2$ different correct entries of DB in S to construct the auxiliary input y in Solve step 1(2).

Proof of Lemma 10.4. Fix $D \subset [q(n)]$ such that $|D| = q(n)/2$. Consider the set S_k. The probability that there is a tuple $(i_1, ..., i_n)$ chosen uniformly and independently at random such that $i_1, ..., i_n \in D$ is 2^{-n} since $|D| = q(n)/2$. By a union bound, the probability that S_k contains a single such tuple is $t(n)/2^n$. By another union bound, the probability that *each* S_k, $1 \leq k \leq p(n)$ contains some $(i_{k,1}, ..., i_{k,n})$ such that $i_{k,1}, ..., i_{k,n} \in D_l$ for *some* $D_l \subset [q(n)]$, $|D_l| = q(n)/2$ is

$$\binom{q(n)}{q(n)/2} \left(\frac{t(n)}{2^n}\right)^{p(n)} < 2^{q(n)} \left(\frac{t(n)}{2^n}\right)^{p(n)}. \tag{3.1}$$

When $p(n) = q(n)$, this becomes $\left(\frac{2t(n)}{2^n}\right)^{q(n)}$, which is negligible for large enough n and completes the proof. □

Having proved the needed technical lemma, we now proceed to the proof of Lemma 10.2.

Proof of Lemma 10.2. Since the Solve algorithms do not affect the distribution, it remains to prove that V^* cannot distinguish between when a party executing (π_0, π_1, π_2') uses a witness

for $x \in L$ versus the witness that the γ corresponds to 1/3 of DB. By Lemmas 10.3 and 10.4, if $\mathcal{S}$ has not been halted by V^* before the beginning of π_2', $\mathcal{S}$ is assumed to have a witness for DB that it outputs to P_2 to use as the auxiliary input y. The remainder of the Lemma follows from the Corollary 9.2 that (π_0, π_1, π_2') is rWI for Λ_3, as discussed in Remark 9.1 and the proof of Lemma 5.3. □

The remainder of our arguments here will focus on the proof of Lemma 10.3; in particular, we demonstrate that $\mathcal{S}$ fails with probability at most $poly(n) \cdot 2^{-O(n)}$.

Proof of Lemma 10.3. Since the simulator makes an independent sequence of look-aheads for each of the m different multi-extraction values at each round of interaction with V^* in the main thread (i.e. $l = L$, $a = 0$) (see Solve), we prove that the simulator almost never fails because it almost never fails when it is only operated with one multi-extraction value.

We consider the algorithm $\mathcal{S}'$ that acts as follows: Fix a value a', $1 \leq a' \leq m$ and let $\mathcal{S}' = \mathcal{S}_a'$ run identically to $\mathcal{S}$ for π_0, π_1 except for $\mathcal{S}'$ only allows NextProverMsg to consider a' and no other and outputs a zero string as the auxiliary input y. In particular, NextProverMsg for $\mathcal{S}'$ will only call Solve$(L-1, a', h, s, j)$ and never Solve$(L-1, a'', h, s, j)$ for any $a'' \neq a'$. Further, $S = S_{a'}$ and $SS = SS_{a'}$. We will prove that $\mathcal{S}'$ outputs fail (which it only considers for $SS_{a'}$ in NextProverMsg) with negligible probability. Since at each round of interaction with V^* in the main thread, the randomness used for Solve$(L-1, a', h, s, j)$ and Solve$(L-1, a'', h, s, j)$ and their respective recursive calls are independent, this implies that $\mathcal{S}$ outputs fail with negligible probability by a union bound.

We prove that $\mathcal{S}'$ outputs fail with probability at most $2^{-O(n)}$. The remainder of this proof follows almost exactly that in [16], whose language we follow very closely for ease of comparison.

In particular, we look to bound the following three types of "bad" events: First, the probability of V^* ever decommitting in iteration j of session s may be small. Second, V^* may open and complete too many new sessions before it decommits in iteration j of session s. Third, the transcript h may contain existing sessions that are closed to completion but not yet solved.

For the first type of bad event, we argue that a session where many of its iterations have small probability of completing will almost never complete. For the second type, we argue that if V^* opens and completes too many new sessions before decommitting at iteration j of session s, then it must be that during other iterations of this session, only few new sessions are opened. For the third type, we show that it never occurs, but this will require that k, the number of iterations, be non-constant.

The proof will proceed as follows. First, we will fix notation. Then, we will show that each type of "bad" event is not a problem as discussed above. Finally, we give a lemma (10.9) that under certain conditions each invocation of Solve succeeds with overwhelming probability. Finally, we complete the proof of Lemma 10.3 using Lemma 10.9. We will first need the following notation.

Simulator states, executions, and attempts. A state σ of the simulator at any point during its run describes all current memory and control information about the current point in the run. An execution describes a specific run of the simulator at some state. It is determined by input x and by the random choices made by the simulator during the execution. In particular, we denote by "execution r" the execution with random choices $r \in \{0,1\}^*$. Note than r contains many distinct strings of n indices, (i), that $\mathcal{S}$ will send to V^*. Formally, an execution r is a sequence of states, where each state follows from the previous one by the code of the simulator and the behavior of V^*. We denote by "execution-segments" as executions that start at some state and end at some other state (i.e. this may be some segment of the larger execution).

Successful attempts. An attempt within an invocation of $\mathsf{Solve}(\mathsf{l}, \mathsf{a}', \mathsf{h}, \mathsf{s}, \mathsf{j})$ is successful if during this attempt an entry $(s, j, DB_{(i)}, (i))$ is added $S_{a'}$. An invocation of Solve is successful if one of its attempts is successful.

Definition of q_σ. Let σ be some state of the simulator at the point where an invocation $\mathsf{Solve}(l, a', h, s, j)$ is called, and consider the following bad event: An invocation of $\mathsf{Simulate}(l, a', h)$ within an attempt of this invocation of Solve returns a transcript in which

V^* halts, and a triple $(s, j, DB_{(i)}, (i))$ was not added to $S_{a'}$ during this attempt. Let q_σ denote the probability of this event. Here, the probability is taken over the random choices of the simulator, starting from state σ.[15]

Definition of μ_σ. Consider an execution r of Simulate starting at state σ. Let $h_{\sigma,r}$ denote the value returned by Simulate in this execution. (Recall that this value holds a transcript, or history of a run of V^*.) Let $\mu_{\sigma,r}$ denote the number of sessions in $h_{\sigma,r}$ that are opened after Simulate is called and are completed before $h_{\sigma,r}$ ends. Let μ_σ denote the expected value of $\mu_{\sigma,r}$ when r is chosen at random. Roughly, μ_σ is the number of expected sessions there are opened and completed only in the main thread of the simulation. Note that since K is the bound on the number of sessions in an execution of V^*, we have that $\mu_\sigma \leq K$ for all σ.

Safe sessions, states, and executions. Consider a state σ of the simulator at the point where either an invocation Simulate(l, a', h) is called or an invocation Solve(l, a', h, s, j) is called. A session s' is called an old session for σ if the verifiers commitment message (i.e. the verifier's commitment to DB) in session s' is already sent in h. An old session s' is called $m\text{-}safe$ for σ if session s' is already solved by the simulator (i.e. $s' \in SS_a$), or V' has not yet decommitted in at least $m+1$ iterations of session s'. State σ is called m-safe if all old sessions for it (and, in particular, session s) are $m\text{-}safe$. An execution of Solve starting from state σ is m-safe if σ is m-safe. An execution of Simulate is m-safe throughout if *all* the states during this execution, in which Simulate is recursively called, are m-safe. An attempt of Solve is m-safe throughout if the corresponding execution of Simulate is $m\text{-}safe$ throughout.

Next, we prove three claims that bound the probability of bad events in executions of Solve and Simulate. The claims here correspond to the three types of bad events that may occur, as discussed above. The first lemma is used to bound the number of attempts that may fail due to old sessions.

Lemma 10.6. *Let $l \geq 0$ and consider an $(l+1)\text{-}safe$ execution of* Solve(l, a', h, s, j). *Then at most $K-1$ attempts in this execution are not $l\text{-}safe$ throughout.*

[15] This definition assumes that invocations of Solve start at the same state, which is not precise. We return at the very end of the proof to demonstrate that such slight imprecision does not affect the proof.

Proof of Lemma 10.6. Assume that an attempt in the given execution of Solve is not l-$safe$ throughout. This means that some old session s' is not l-$safe$ throughout at some state during this attempt. For that to happen, V^* must have properly decommitted in some iteration $k-l$ of session s'. However, in this case the entry $(s', k-l, DB_{(i)}, (i))$ is added to $S_{a'}$ in this attempt. Since the current execution of Solve is $(l+1)$-$safe$, we have the the prover's index for iteration $k-l$ in sessions s' is not yet sent in h. Consequently, in future attempts session s' will remain l-$safe$. (Either V^* will not decommit in iteration $k-l$ or session s' will be solved and added to $SS_{a'}$.) There are at most $K-1$ old sessions. Therefore at most $K-1$ attempts are prevented from being l-$safe$ throughout. □

For the next two lemmas, consider an execution of Simulate at level l. The second lemma (Lemma 10.7) says roughly that any new session that reaches the point where $2k/3$ of its iterations are completed must have many iterations where the corresponding invocations of Solve, at level $l-1$, have low probability of failure due to the fact that V^* halts. The third lemma (Lemma 10.8) says that any new session that reaches the point where $2k/3$ of its iterations are completed must have many iterations where the corresponding values of μ are sufficiently small.

More precisely, consider an execution r of Simulate at level l, starting from state σ. Recall that $h_{\sigma,r}$ denotes the value returned by Simulate in this execution. Let $\sigma_{i,j,r}$ denote the state of the simulator at the point where Solve is invoked (at level $l-1$) for the jth iteration at the ith new session in $h_{\sigma,r}$. Recall that $\mu_{\sigma,r}$ denotes the number of sessions in $h_{\sigma,r}$ that are opened after the execution of Simulate started and completed before this execution returns, and that μ_σ denotes the expected value of $\mu_{\sigma,r}$ where r is chosen at random. Define the following bad events:

Qbad executions. An execution r of Simulate, starting from state σ, is i-Qbad if $q_{\sigma_{i,j,r}} > \frac{31}{32}$ for at least $k/12$ iterations j out of the first $2k/3$ iterations in the ith new session in $h_{\sigma,r}$ to complete $2k/3$ iterations. An execution is Qbad if it is i-Qbad for some i.

Mbad executions. An execution r of Simulate at level l, starting from state σ, is i-Mbad if $\mu_{\sigma_{i,j,r}} > \frac{1}{64}(\frac{k}{2^{14}})^{l-1}$ for at least $k/2$ iterations j out of the first $2k/3$ iterations in the ith new

session to complete $2k/3$ iterations in $h_{\sigma,r}$. An execution is Mbad if it is i-Mbad for some i.

Observe that in an execution r of Simulate at level l that is neither Qbad nor Mbad, any session that completes $2k/3$ iterations has at least one iteration j where both $q_{\sigma_{i,j,r}} \leq \frac{31}{32}$ and $\mu_{\sigma_{i,j,r}} \leq \frac{1}{64}(\frac{k}{2^{14}})^{l-1}$. We will use this in proving the induction step of Lemma 10.9 below.

Long executions. An execution r of Simulate, starting from state σ, is l-long if $\mu_{\sigma,r} > (\frac{k}{2^{14}})^l$. An execution that is not *l-long* is said to be l-short.

Lemma 10.7. *Let σ be a state of the simulator at the point where an execution of* Simulate *is called. Then an execution of* Simulate *starting from state σ is* Qbad *with probability at most $K \cdot 2^{-O(k)}$.*

Proof of Lemma 10.7. Let β_i denote the probability that an execution is *i-Qbad*. We show that $\beta_i \leq 2^{-O(k)}$ for every i. Consider an *i-Qbad* execution r and let r_j denote the segment of r that is used by the simulator from state $\sigma_{i,j-1,r}$ until state $\sigma_{i,j,r}$ is reached. Note that $r_1, .., r_k$ are disjoint and contained in r. Since the execution is *i-Qbad*, we have that $q_{\sigma_{i,j,r}} > \frac{31}{32}$ for at least $k/12$ iterations j out of the first $2k/3$ iterations in the ith session. Thus, we have:

$$\begin{aligned}
\beta_i &\leq \sum_{B\subset\{1,\ldots,\frac{2k}{3}\},|B|<\frac{k}{12}} \Pr_r(\text{r is } i\text{-}Qbad \mid q_{\sigma_{i,j,r}} > \frac{31}{32} \text{ for } j \in B) \\
&\leq \sum_{B\subset\{1,\ldots,\frac{2k}{3}\},|B|<\frac{k}{12}} \Pr_r(\wedge_{j\in B} \text{ iteration } j \text{ completed} \mid q_{\sigma_{i,j,r}} > \frac{31}{32} \text{ for } j \in B) \\
&\leq \sum_{B\subset\{1,\ldots,\frac{2k}{3}\},|B|<\frac{k}{12}} \Pi_{j\in B} \Pr_{r_j}(\text{iteration } j \text{ completed} \mid q_{\sigma_{i,j,r}} > \frac{31}{32}) \\
&\leq \binom{2k/3}{k/12} \cdot (1/32)^{k/12} \leq 2^{-O(k)}.
\end{aligned} \tag{3.2}$$

Note that the third inequality follows from the fact that the simulator uses independent random choices for each message sent to V^*. It follows than an attempt is Qbad with probability $\sum_{i=1}^{K} \beta_i \leq K \cdot 2^{-O(k)}$. □

Lemma 10.8. *Consider a state σ at the point where an invocation of* Simulate *at level l is called, and assume that $\mu_\sigma \leq \frac{1}{64}(k/2^{14})^l$. Then, a random l-short execution of* Simulate *starting from state σ is Mbad with probability at most $K2^{-O(k)}$.*

Proof of Lemma 10.8. Let $h^{i,j}_{\sigma,r}$ denote the segment of $h_{\sigma,r}$ that starts when V^* decommits in iteration $j-1$ in the ith session opened in execution r, and ends when V^* either halts of decommits in iteration j of this session. Let $m^{i,j}_{\sigma,r}$ denote the number of sessions that are opened and completed within the segment $h^{i,j}_{\sigma,r}$. Observe that for any execution r and session i we have that $\sum_{j=1}^{k} m^{i,j}_{\sigma,r} \leq \mu_{\sigma,r}$. This is because for all r, the segments $\{h^{i,j}_{\sigma,r}\}_{j=1,\ldots,k}$ are disjoint and contained in $h_{\sigma,r}$. It holds that in each execution r there are at most $k/4$ iterations j with $m^{i,j}_{\sigma,r} > \frac{4}{k}\mu_{\sigma,r}$ Since in l-short executions we have $\mu_{\sigma,r} \leq (k/2^{14})^l$, it follows that in such executions there are at least $3k/4$ iterations j with

$$m^{i,j}_{\sigma,r} \leq \frac{4}{k}\mu_{\sigma,r} \leq \frac{4}{k}(\frac{k}{2^{14}})^l = \frac{1}{2^{12}}(\frac{k}{2^{14}})^{l-1}.$$

It follows that in Mbad executions there are at least $k/4$ iterations j among the first $2k/4$ iterations, such that $m^{i,j}_{\sigma,r} \leq \frac{1}{2^{12}}(\frac{k}{2^{14}})^{l-1}$ and $\mu_{\sigma_{i,j,r}} \leq \frac{1}{64}(k/2^{14})^{l-1}$. Call such iterations unlikely. Notice that an unlikely iteration occurs with probability at most 2^{-6} because $m^{i,j}_{\sigma,r}$ describes a value drawn at random from a distribution with expectancy at least $\mu_{\sigma_{i,j,r}}$. It follows from a Markov inequality that $m^{i,j}_{\sigma,r}$ is less than 2^6 times $\mu_{\sigma_{i,j,r}}$ only with probability 2^{-6}. Moreover, unlikely iterations occur independently of each other, since they use disjoint segments of the random input r. More precisely, recall that r_j denotes the segment of r that is used by the simulator from state $\sigma_{i,j-1,r}$ until state $\sigma_{i,j,r}$ is reached. Let γ_i denote the

probability that an l-short execution is i-$Mbad$. We then have that

$$\begin{aligned}\gamma_i &\leq \sum_{B\subset\{1,\ldots,\frac{2k}{3}\},|B|<\frac{k}{4}} \Pr_r(\text{iterations } j\in B \text{ are unlikely}) \\ &\leq \sum_{B\subset\{1,\ldots,\frac{2k}{3}\},|B|<\frac{k}{4}} \Pi_{j\in B}\Pr_{r_j}(\text{iteration } j \text{ is unlikely}) \\ &\leq \binom{2k/3}{k/4}\cdot(2^{-6})^{k/6} \leq 2^{-O(k)}.\end{aligned}$$

It follows that an l-short execution is Mbad with probability at most $\sum_{i=1}^{K}\gamma_i \leq K\cdot 2^{-O(k)}$. □

We finally have the following main technical lemma.

Lemma 10.9. *Let σ be a state of the simulator at the point where some innovation of* Solve *at level l is called, and assume that:*

- $q_\sigma \leq \frac{31}{32}$
- $\mu_{sigma} < \frac{1}{64}(\frac{k}{2^{14}})^l$
- *State σ is $(l+1)$-safe.*

Then there exists a constant c such that a random execution of Solve *starting from state σ fails with probability 2^{-cK}. The probability is taken over the random choices of the simulator starting from state σ.*

Proof of Lemma 10.9. We proceed by induction on l:

Base case: $l = 0$. An execution of Solve its not successful only if none of its $128K$ attempts is successful. We bound the probability that this event happens. We first assert the following fact:

Base case fact: The invocation of $\mathsf{Simulate}(0, a', h)$ within a random attempt that is 0-safe throughout returns to to failure of NextProverMsg with probability at most $1/64$.

Proof: Executions of Simulate that are 0-safe throughout never return due to failure of NextProverMsg on an old session. The probability that an execution of Simulate returns due to failure of NextProverMsg on a new session is at most the probability that V^* starts and completes a new session during this execution. However, since $\mu_{sigma} < \frac{1}{64} \cdot (k/2^{14})^0 = \frac{1}{64}$, it follows from a Markov inequality that V^* starts and completeness even a single new session during this execution with probability at most 1/64. □

We are not ready to prove the base case. Say than an attempt is unfortunate if one of the two events occur:

(a) The invocation of Simulate returns due to the fact that V^* halts, and a tuple $(s, j, DB_{(i)}, (i))$ was not added to $S_{a'}$ during this attempt.

(b) The invocation of Simulate returns due to failure of NextProverMsg on a new session.

By the premise of the lemma, the probability of event 1 is at most $q_\sigma < \frac{31}{32}$. By our Base-case fact, the probability of event 2 is at most 1/64. Altogether the expected number of unfortunate attempts us at most $\frac{63}{64} \cdot 128K = 126K$. Since the attempts are independent of each other, it follows from a Chernoff inequality that the probability that more than $126.5K$ attempts are unfortunate is at most 2^{-cK} for some $c > 0$. In other words, except with probability 2^{-cK}, there are at least $1.5K$ attempts that are either successful or where Simulate returns due to failure of NextProverMsg on an old session. However, it follows from Lemma 10.6 that at most K attempts are not 0-safe throughout, thus in at most K attempts Simulate return due to failure of NextProverMsg on an old session. It follows that, except with probability 2^{-cK}, at least $K/2$ attempts succeed. (Note that we only care about a single attempt to succeed here).

Induction step: As in the base case, we bound the probability that none of the $128K$ attempts in an execution of Solve, now at level l, is successful. We first demonstrate the following fact.

Induction-step fact: Let σ be the state where an invocation of Simulate at level l is called. Assume that Lemma 10.9 holds for all $l' < l$ and that $\mu_\sigma < \frac{1}{64}(\frac{k}{2^{14}})^l$. Then a random execution of Simulate, starting from state σ, that is l-short and l-safe throughout, returns due to failure of NextProverMsg with probability at most $K2^{-poly(k)}$.

Proof: Recall than an execution of Simulate that is l-safe throughout never returns due to failure of NextProverMsg on an old session. We bound the probability that an l-safe throughout execution of Simulate returns due to failure of NextProverMsg on a new session. In fact, we show that except with probability $1/64 + K2^{-poly(k)}$, all new sessions in this execution remain $k/3$-safe, which is to say that they do note go beyond their $2k/3$th iteration without being solved.

Say that an execution of Simulate at level l is bad if it is other Qbad or Mbad. Recall that a random execution of Simulate at level l is Qbad with probability at most $K2^{-poly(k)}$ by Lemma 10.7, and is Mbad given that it is l-short with probability at most $K2^{-poly(k)}$ by Lemma 10.8. It follows that a random execution of Simulate is bad with probability at most $K2^{-poly(k)}$. We show that an execution of Simulate that is no bad and l-safe throughout fails during Solve on a new session with probability at most $K2^{-O(K)}$. In particular, we demonstrate the following inductive claim. Order the new sessions in an execution of Simulate according to the order in which they complete $2k/3$ iterations. That is, let s_i denote the ith session to complete $2k/3$ iterations. Let Z_i denote the event that, at the point where the session s_i has completed $2k/3$ iterations, there are new sessions that are not $k/3$-safe. We show by induction on i that there exists a constant $c > 0$ such that $\Pr(Z_i) < i \cdot 2^{-cK}$.

We proceed to prove the inductive step. Let $i \geq 1$ (note that the base step is included here), and let r be a random execution of an attempt that is l-safe throughout, not Qbad and not Mbad. Consider the invocations of Solve at level $l-1$ that are associated with the first $2k/3$ iterations of the ith session. Sat than an invocation is good if the three conditions of Lemma 10.9 are satisfied with respect to this invocation. We want to show that at least one of these invocations of Solve is good. Recall that, since the execution is neither Qbad

nor Mbad, there is at least one such innovation of Solve for which the first two conditions of the lemma are met. It remains to show that this invocation is also l-safe. Since the attempt is l-safe throughout, all the old sessions are l-safe. From the hypothesis of the induction on i we have that, except with probability $(i-1)2^{-cK}$, all the new session are $k/3$ safe, hence also l-safe.

We conclude that, except with probability $(i-1)2^{-cK}$, at least one out of the first $2k/3$ invocations of Solve associated with the ith session is good. Applying the hypothesis of the induction on l, we get that this invocation of Solve fails with probability 2^{-cK}. Therefore, the ith session is not solved by the time $2k/3$ of its iterations are completed with probability at most $2^{-cK} + (i-1)2^{-cK} = i2^{-cK}$. This completes the inductive step on i and the proof of the induction step fact. □

We now complete the induction on l. Recall the definition of unfortunate attempts. By the premise of the lemma, the probability of event 1 is at most $q_\sigma < \frac{31}{32}$. By our Induction step fact, the probability that Simulate returns due to failure of NextProverMsg in an l-short execution is at most $K2^{-poly(k)} = 2^{-poly(n)}$. By the premise of Lemma 10.9, $\mu_\sigma < \frac{1}{64}(\frac{1}{2^{14}})^l$. It follows from a Markov inequality that an execution of Simulate within an attempt of Solve at level l is l-long with probability at most $1/64$. Therefore, the probability of event 2 is at most $1/64 + 2^{-poly(n)}$. Thus the expected number of unfortunate attempts is at most $(\frac{63}{64} + 2^{-poly(n)})128K = 126K + 2^{-poly(n)}$. Since the attempts are independent of each other, it follows from a Chernoff inequality that the probability that more than $126/5K$ attempts are unfortunate is a most 2^{-cK} for some $c > 0$. Therefore, except with probability 2^{-cK}, there are at least $1.5K$ attempts that are either successful or where Simulate returns due to failure of NextProverMsg on an old session. However, it follows from Lemma 10.6 that at most K attempts are not l-safe throughout, thus in at most K attempts Simulate returns due to failure of NextProverMsg on an old session. It follows that exempt with probability 2^{-cK}, at least $K/2$ attempts succeed. □

We give the final lemma needed to complete the proof of Lemma 10.3.

Lemma 10.10. *The main procedure of $\mathcal{S}'$ fails with probability at most $2^{-cK} = 2^{-poly(n)}$.*

Proof of Lemma 10.10. The main procedure consists of a single invocations of Simulate at level L, and fails if this invocation returns due to the failure of NextProverMsg. We prove the lemma by applying the induction-step fact of the previous lemma to this invocation (which is to say, we note that the induction-step fact also applied to the innovation of Simulate within the main procedure). Specifically, all the requirements of the induction-step fact are met:

- Lemma 10.9 holds for all $l < L$.
- Let σ_0 be the state of the simulator at the point where Simulate is invoked within the main procedure. Thus, for $K > 64$ we have $\mu_{\sigma_0} < \frac{K^2}{64} = \frac{1}{64}(\frac{k}{2^14})^L$. Furthermore, any executions of Simulate within the main procedure is L-short.
- There are no old sessions for state σ_0. Therefore, any execution of Simulate within the main procedure is L-safe throughout.

We conclude that a random execution of Simulate within the main procedure of $\mathcal{S}'$ returns due to failure of NextProverMsg with probability at most $2^{-poly(n)}$. □

We make one final note. In our definitions and arguments regarding q_σ and μ_σ, we have ignored the fact that different attempts within an invocation of Solve start with somewhat different states of the simulator. In particular the sets $S_{a'}$ and $SS_{a'}$ may differ from attempt to attempt (and also in the continuation of a simulation once Solve returns). Consequently, when invoked by Simulate in different attempts, procedure NextProverMsg may try to solve different sessions in steps 3(2)(b) and 3(2)(c) (for $l < L$ of) NextProverMsg. However, this is transparent to V^* since $\mathcal{S}'$ will send independent random strings to V^* in each attempt, so the above analysis is precise.

CHAPTER 4

5PM: Secure Pattern Matching

1 Introduction

Pattern matching is fundamental to computer science[1]. It is used in many areas, including text processing, database search [2], networking and security applications [70] and recently in the context of bioinformatics and DNA analysis [72, 82, 11]. It is a problem that has been extensively studied, resulting in several efficient (although insecure) techniques to solve its many variations, e.g, [81, 1, 64, 58]. The most common interpretation of the pattern matching problem is the following: given a finite alphabet Σ, a text $T \in \Sigma^n$ and a pattern $p \in \Sigma^m$, the *exact pattern matching decision problem* requires one to decide whether or not a pattern appears in the text. The *exact pattern matching search problem* requires finding all indices i of T (if any) where p occurs as a substring starting at position i. If we denote by T_i the ith character of T, the output should be the set of matching positions $MP := \{i \mid p \text{ matches } T \text{ beginning at } T_i\}$. The following generalizations of the exact matching problem are often encountered, where the output in all cases is the set MP:

- *Pattern matching with single character wildcards*[2]*:* There is a special character "$*$" $\notin \Sigma$ that matches any single character of the alphabet, where $p \in \{\Sigma \cup \{*\}\}^m$ and $T \in \Sigma^n$. Using such a "wildcard" character allows one pattern to be specified that could match several sequences of characters. For example the pattern "$TA*$" , would match any of the following character sequence in a text[3]: TAA, TAC, TAG, and TAT.

[1]We would like to thank Matt Cheung and Sky Faber for helpful discussions and comments.
[2]Such wildcards are also called "do not cares" and "mismatches" in the literature.
[3]Here and throughout, we use the DNA alphabet ($\Sigma = \{A, C, G, T\}$) for examples.

- *Substring pattern matching:* Fix some $l \leq m$; a match for p is found whenever there exists in T an m-length string that differs in l characters from p (i.e., has Hamming distance l from p). For example, the pattern "TAC" has $m = 3$. If $l = 1$, then any of the following words would match: $*AC$, $T * C$, or $TA*$; note that this is an example of non-binary substring matching.

A secure version of pattern matching has many applications. For example, secure pattern matching can help secure databases containing medical information, such as DNA records, while still allowing one to perform pattern matching operations on such data. The need for privacy-preserving DNA matching has been highlighted in recent papers [8, 59, 80]. In addition to the case of DNA matching, where substring matching may be particularly useful, Hamming distance-based approximate matching has also been demonstrated in the case of secure facial recognition [72]. We note that both of these settings require computation over non-binary alphabets.

1.1 Our Contributions

This paper presents a new protocol for arbitrary alphabets, **5**ecure **P**attern **M**atching (or 5PM), that addresses, in addition to exact matching, more expressive search queries including single character wildcards and substring pattern matching, in addition to providing the ability to hide pattern length.

5PM has communication complexity sub-linear in circuit size (as opposed to general MPC, which has communication complexity linear in circuit size) to securely compute non-binary substring matching in the malicious model and is also the first non-general MPC protocol to explicitly compute non-binary Hamming distance in the malicious model. In addition, our extension of Hamming distance computation to substring matching has minimal overhead; our protocol makes a single computation pass per text element, even for multiple Hamming distance values, and therefore is able to securely compute non-binary substring matching efficiently (see Table 4.2 for details).

5PM performs exact, single character wildcard, and substring pattern matching in the

honest-but-curious and malicious (static corruption) models. Our malicious model protocol requires $O((m+n)k^2)$ bandwidth complexity. Further, our protocol can be specified to require 2 (one-way) rounds of communication in the semi-honest model and 8 (one-way) rounds of communication in the malicious model.

We construct our protocols by reducing the problems of Hamming distance and pattern matching, including single character wildcards and substring matching, to a sequence of linear operations. We then rely on the observation that such linear operations, such as the inner products and matrix multiplication, can be efficiently computed in the malicious model using additively homomorphic encryption schemes.

The security requirements (informally) dictate that the party holding the text learns nothing except the upper bound on the length of the pattern, while the party holding the pattern only learns either a binary (yes/no) answer for the decision problem or the matching positions (if any), and nothing else.

1.2 Comparison to Previous Work

Exact Matching. In the exact pattern matching setting, the algorithm of Freedman, Ishai, Pinkas and Reingold [32] achieves polylogarithmic overhead in m and n and polynomial overhead in the security parameters in the honest-but-curious setting. Using efficient arguments [61, 66] with the modern probabilistically checkable proofs (PCP) of proximity [10], one can extend (at least asymptotically) their results to the malicious (static corruption) model. However, the protocol in [32] works only for exact matching and does not address more general problems including single character wildcards and substring matching, which is the main focus of our work. Other protocols that address secure exact matching (and not wildcard or substring matching) are [80, 33, 48, 35, 69, 59]; of these, only [35] obtains (full) security in the malicious setting. We note that [69] is more efficient than [32] but only in the random oracle model; here, we are interested in standard security models.

Paper	NB Hamming Distance	Exact Matching	Wildcard Matching	NB Substring Matching	Security
[32]	Yes*	Yes	Yes*	No	HBC/M
[84]	Yes*	Yes	Yes	Yes*	HBC/M
[57]	Yes	No**	No**	No*	HBC
5PM	Yes	Yes	Yes	Yes	HBC/M

Table 4.1: Comparison of previous protocol functionality, NB= Non-Binary HBC = Honest but Curious, M= Malicious, *=Using unary encoding, **=Can be extended

Paper	Encryptions	Exponentiations	Multiplications	Bandwidth	Rds
[57]	$O(n+m)$	$O(nm)$	$O(nm)$	$O((nm)k)$	$O(1)$
5PM	$O(n+m)$	$O(n+m)$	$O(nm)$	$O((n+m)k)$	2

Table 4.2: Detailed comparison with [57] for non-binary substring matching in HBC model with Text length=n, Pattern length=m, Security Parameter=k, Rds= Rounds.

Paper	Encryptions	Exponentiations	Multiplications	Bandwidth	Rds
[49]	$O(mn)$	$O(mn)$	$O(mn)$	$O(mnk^2)$	$O(1)$
[84]	$O(n+m)$	$O(n \log m)$	$O(nm)$	$O((n+m)k^2)$	$O(1)$
5PM	$O(n+m)$	$O(nm)$	$O(nm)$	$O((n+m)k^2)$	8

Table 4.3: Detailed comparison with [84] and [49] for single character wildcards and substring matching in malicious model with Text length=n, Pattern length=m, Security Parameter=k, Rds= Rounds.

Single Character Wildcards and Substring Matching. Recently, Vergnaud [84] built on the work of Hazay and Toft [49] to construct an efficient secure pattern matching scheme for wildcard matching and substring matching (requiring t runs over the preliminary matching result to search for t different Hamming distance values, which is also required by 5PM) in the malicious adversary model. More specifically, [84, 49] take advantage of the fact that $(p_i - t_i)^2$ equals 0 if binary values p_i and t_i are equal and 1 if they are not equal; therefore, binary Hamming distance can essentially be computed by counting the number of 1s in the computation. However, when p_i and t_i are non-binary, it is unknown how to execute oblivious polynomial computations that output 0 when p_i and t_i equal and 1 (or some other fixed value) when they do not equal.

However, non-binary elements can be computed by unary encoding, that is an element $\alpha \in \Sigma$ can be encoded as an element $\alpha' \in \{0,1\}^{|\Sigma|}$ with all 0s except for a single 1 in the place

representing α (lexicographically). There are two subtleties of such an appraoch- the first is that if $\alpha \neq \beta$, then α' and β' will have Hamming distance *two*. Second, in the malicious case, zero knowledge proofs are needed to demonstrate that α' is well-formed. This is easily accomplished using techniques in this paper[4].

[84] requires $O(m+n)$ encryptions, $O(n \log m)$ exponentiations, $O(nm)$ multiplications (of encrypted elements), and $O(n+m)$ bandwidth, all in a constant number of rounds. By contrast, 5PM has the same overhead except for $O(nm)$ exponentiations (see Table 4.3). However, our work is of interest for several reasons. The first is that we have implemented our protocol and believe it to be more efficient (additional work is needed on this front). The second is that our techniques are of independent interest and may be extended to additional functionalities.

Non-binary Hamming Distance. Jarrous and Pinkas [57] gave the first construction of a secure protocol for computing non-binary Hamming distances. In order to count the non-binary mismatches, they leverage 1-out-of-2 oblivious transfers. 5PM can also compute non-binary Hamming distance even when the text and pattern have the same length (and where the output is not blinded to only reveal whether or not a pattern match occurred). We note that [57] can be used to implement exact and substring matching with additional tools to blind Hamming distance output (for instance, see [84]). [57], to compare 2 strings of length n, requires $O(n)$ 1-out-of-2 OTs, $O(n)$ multiplications of encryptions and $O(nk)$ bandwidth, while 5PM requires $O(n)$ exponentiations (which require less computation than OTs), $O(n^2)$ multiplications, and $O(nk)$ bandwidth. The advantage of 5PM over [57] is twofold: the first is that it is proven secure in the malicious model while [57] is not. The second advantage is that 5PM, in both the honest-but-curious and malicious models, amortizes well in the substring matching setting, while [57] does not amortize because it cannot reuse OT outputs to compute substring matching (see table 4.2).

[4]Prove using a zero-knowledge argument that the sum of the encryption of α' is 1 and every individual encrypted element in the vector is either a 1 or a 0, see Section 6.1.

Other Techniques. In the most general case, secure exact, approximate and single character wildcards pattern matching is an instance of general secure two-party computation techniques (for instance, [86, 41, 56, 25]). All of these schemes have bandwidth and computational complexity at best linear in the circuit size. For instance, a naive implementation of Yao [86] requires bandwidth $O(mn)$ in the security parameter. In contrast, we aim for a protocol where circuit size is $O(mn)$ yet we achieve communication complexity of $O(m+n)$.

Finally, we observe that with the construction of fully homomorphic encryption (FHE) schemes [36], the following "folklore" construction can be executed for any pattern matching algorithm: Client encrypts its pattern using an FHE scheme and sends it to Server. Server applies the appropriate pattern matching circuit to the encrypted pattern (where the circuits output is a *yes/no* indicating whether a match exists or not), and sends the FHE circuit output to Client. Client decrypts to obtain the answer. Such a scheme requires $O(m)$ bandwidth, but since FHE schemes are not yet practical, we view the 5PM protocol outlined here as an efficient and practical solution to secure pattern matching with single character wildcards and substring matching.

2 Preliminaries

The rationale behind our secure 5PM protocol is based on a modification of an insecure pattern matching algorithm (IPM) [50] that can perform exact matching, exact matching with single character wildcards and substring matching within the same algorithm. In Section 3.1, we show how our modified algorithm can be reduced to basic linear operations whose secure and efficient evaluation allows us to obtain our 5PM protocol.

2.1 Insecure Pattern Matching (IPM) Algorithm

To illustrate how our modified algorithm works, we begin by describing how it performs exact matching; we then show how it handles single character wildcards and substring matching.

2.1.1 Exact Matching

IPM involves the following steps:

(a) *Inputs*: An alphabet Σ, a text $T \in \Sigma^n$ and a pattern $p \in \Sigma^m$.

(b) *Initialization*: For each character in Σ, the algorithm constructs a vector, here termed a **C**haracter Delay Vector (CDV), of length equal to the pattern length, m. These vectors are initialized with zeros. For example, if the pattern is: "$TACT$" over $\Sigma = \{A, C, G, T\}$, then the CDVs will be initialized to: $CDV(A) = [0, 0, 0, 0]$, $CDV(C) = [0, 0, 0, 0]$, $CDV(G) = [0, 0, 0, 0]$ and $CDV(T) = [0, 0, 0, 0]$.

(c) *Pattern preprocessing*: For each pattern character p_i ($i \in \{1, ..., m\}$), a delay value, $d^r_{p_i}$, is computed to be the number of characters from p_i to the end of the pattern, i.e., $d^r_{p_i} = m - i$ for the rth occurrence of p_i in p. The $d^r_{p_i}$th position of $CDV(p_i)$ is set to 1. For example the CDVs of "$TACT$" would be:

$CDV(A) = [0, 0, 1, 0]$ because $d^1_A = 4 - 2 = 2$

$CDV(C) = [0, 1, 0, 0]$ because $d^1_C = 4 - 3 = 1$

$CDV(G) = [0, 0, 0, 0]$ because $G \notin p$

$CDV(T) = [1, 0, 0, 1]$ because $d^1_T = 4 - 4 = 0$ and $d^2_T = 4 - 1 = 3$

(d) *Matching pass and comparison with pattern length*: A vector of length n called the **Activation Vector (AV)** is constructed and its elements are initialized with zeros. For each input text character T_j, $CDV(T_j)$ is added element-wise to the AV from position j to position $\min(n, j + m - 1)$. To determine if there was a pattern match in the text, after these operations the algorithm checks (when $j \geq m$) if $AV_j = m$. If so, then the match started at position $j - m + 1$. The value $j - m + 1$ is added to the set of matching positions (MP). *Note that $n - AV_j$ is the non-binary Hamming distance of the pattern and the text staring at position $j - m + 1$.*

The intuition behind the algorithm is that when an input text character matches a character in the pattern, the algorithm *optimistically* assumes that the following characters will correspond to the rest of the pattern characters. It then adds a 1 at the position in the activation vector several steps ahead, where it would expect the pattern to end (if the character appears in multiple positions in the pattern, it adds a 1 to all the corresponding positions where the pattern might end). If all subsequent characters are indeed characters in the pattern, then at the position where a pattern would end the number of added 1s will sum up to the pattern length; otherwise the sum will be strictly less than the pattern length. This algorithm does not incur false positives and always indicates when (and where) a pattern occurs if it exists, as shown in [50].

Insecure Pattern Matching (IPM) Example The pattern to be matched is "$TACT$" and the text is "$GATTACT...$". The first step is to construct the CDV using the delays for the characters of the pattern "$TACT$". Delays for "T" will be 3 and 0, for "A" will be 2, and for "C" will be 1. These delays are then converted to CDVs as shown in Figure 4.1. The activation vector will be initialized to all zeros. The characters of the text are then considered one at a time. For each input text character ($T[j]$) at position j in text, the following steps have to be taken: (1) retrieve $CDV[T[j]]$; (2) add elements of $CDV[T[j]]$ to elements of activation vector from position j to $j+m-1$ and (3) check if $AV[j]$ is equal to the pattern length ($|TACT| = m = 4$).

2.1.2 Single Character Wildcards, Pattern Hiding and Substring Matching

Single character wildcards can be handled in IPM by representing a single character wildcard with a special character,"$*$" which is not in the text alphabet. When "$*$" is encountered in the pattern preprocessing phase it is ignored, i.e., no 1s are added to any CDV. Additionally, at the last step when elements of the AV are searched in the comparison phase, the threshold value being compared against will be $m-l$ instead of m, where l is the number of occurrences of "$*$" in the pattern. The intuition behind single character wildcards is that by reducing the

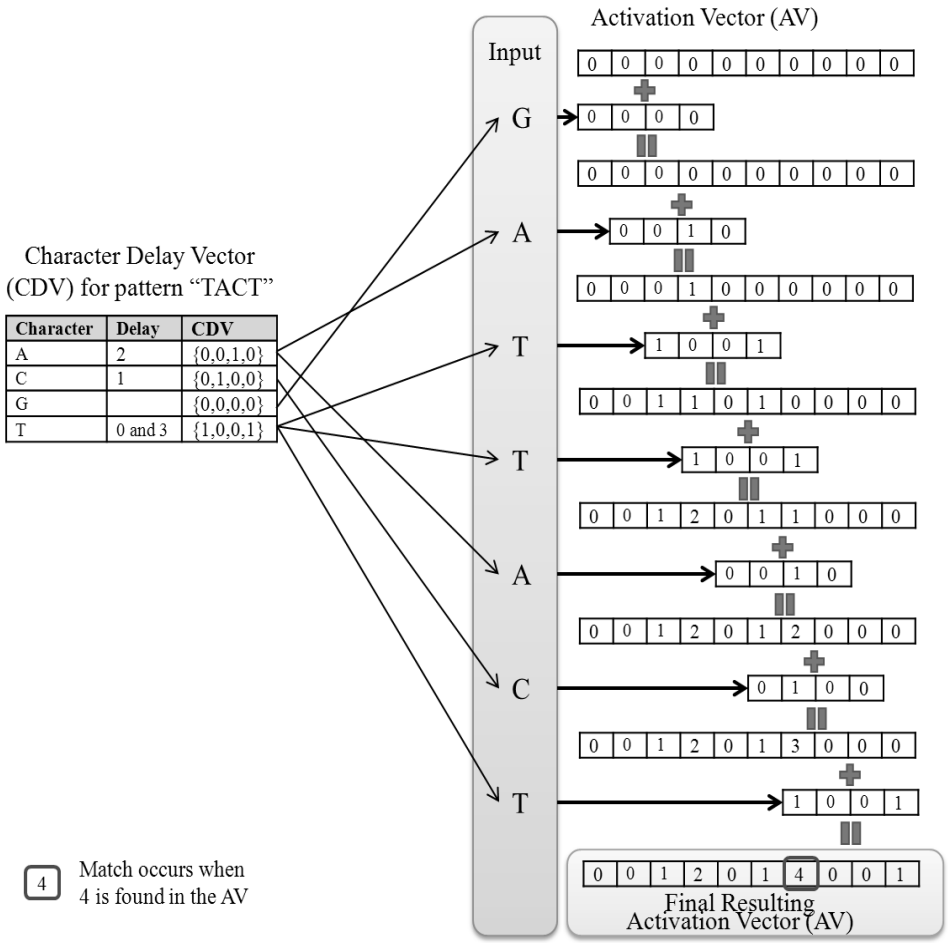

Figure 4.1: Example of IPM's operation

threshold for each wildcard, the algorithm implicitly skips matching that position in the text, allowing that position of the pattern to correspond to any character. This operation does not incur any false positives for the same reason that the exact matching IPM algorithm does not: there, for each pattern p, there is only one encoding into CDVs and only one sequence of adding CDVs as one moves along the text that could add up to m. The same reasoning holds when "$*$" is present in p (except that the sequence adds to $m - l$).

We note that, using single character wildcards, one can always hide pattern length by setting p' as the concatenation of p and a string of $n - m$ wildcards, $*^{n-m}$, and using p' to execute pattern matching for p.

Substring matching, or matching text substrings of Hamming distance $m - l$ from the pattern, is handled similarly to single character wildcards; the threshold value being compared against in the AV is decreased to $m - l$. For further details, we refer the reader to [50].

2.2 Preliminary Cryptographic Tools

This section outlines preliminary cryptographic tools required for our protocols. For $x, y \in \mathbb{Z}_q^n$, we define the inner product of x and y over $\mathbb{Z}_q$, denoted $\langle x, y \rangle$, as $\sum x_i y_i \bmod q$.

Additively Homomorphic Encryption: We make use of additively homomorphic semantically secure encryption schemes. For concreteness, we concentrate in the rest of the paper on the additively homomorphic ElGamal encryption scheme whose security depends on the Decisional Diffie-Hellman (DDH) computational hardness assumption. An additively homomorphic ElGamal encryption scheme [21] is instantiated by choosing a group of appropriate prime order q, $\mathbb{G}_q$, with generator g, and setting the secret-key to be $x \in \mathbb{Z}_q$ and the public-key to be $(g, h = g^x)$. To encrypt a message m one chooses a uniformly random $r \in \mathbb{Z}_q$ and computes $(g^r, g^m h^r)$. To decrypt a pair (α, β), one computes $\log_g \frac{\beta}{\alpha^x}$. It is important to note for additive ElGamal that the decryptor has to both decrypt and also compute a discrete logarithm to discover the message. However, our scheme only requires a determination of whether an encrypted value is of a 0 or not, which can accomplished without computing

logarithms.

Threshold Encryption: The malicious model version of 5PM requires an additively homomorphic, semantically secure, threshold encryption scheme [28]. While we use threshold ElGamal, in practice, any scheme is acceptable if it satisfies the required properties and supports the needed zero-knowledge arguments. Threshold ElGamal in the two party case can be informally defined as follows [14]: party P_1 has share x_1 and party P_2 has share x_2. The parties jointly set the secret-key to be $x = x_1 + x_2$ (this can be performed without revealing x_1 and x_2, see subprotocol π_{encr} in Section 3.3). Without loss of generality, P_1 partially decrypts (α, β) by sending $(\alpha, \frac{\beta}{\alpha^{x_1}})$ to P_2, who fully decrypts (α, β) by computing $\frac{\beta}{\alpha^{x_1}\alpha^{x_2}} = \frac{\beta}{\alpha^{x_1+x_2}}$. We denote the partial decryption algorithm for party P_i as D_{P_i}.

Commitment Schemes: For the malicious model protocol, we will make use of perfectly hiding, computationally binding commitment schemes (for further discussion, see [37]). The Pedersen commitment scheme [75] is a well-known example of such a commitment scheme; for a multiplicative group of prime order q, $\mathbb{G}_q$ and for fixed generators $g, h \in \mathbb{G}_q$, commitment to message s using randomness r is $g^s h^r = comm(g, h, r, s)$.

Zero-Knowledge Arguments of Knowledge: In order to construct a protocol that guarantees that each party behaves properly even in the malicious setting, we utilize efficient interactive zero-knowledge arguments of knowledge (ZK-AoKs). For further details, see Section 4.

2.3 Computing Linear Operations Using Additively Homomorphic Encryption Schemes

Our secure pattern matching protocol relies on the following observations about linear operations and additively homomorphic encryption schemes. In what follows, let E be the encryption algorithm for an additively homomorphic encryption scheme for key pair (pk, sk). Suppose the plaintext group G can be expressed as $\mathbb{Z}_n$ for some $n \in \mathbb{N}$; in particular, G is a ring. Let $M_{a,b}(G)$ denote the set of matrices of size $a \times b$ with entries in G.

2.3.1 Matrix Multiplication

Consider two matrices, A and B, where $A \in M_{k,l}(G)$ and $B \in M_{l,m}(G)$. Suppose that P_1 possesses pk, $E_{pk}(A)$, the entry-wise encryption of A , and also the unencrypted matrix B. Then P_1 can compute $E_{pk}(A \cdot B)$, the encryption of the multiplication of A and B under the same pk. Such an operation is possible because one can obtain an encryption of the inner product over G of an unencrypted vector $(x_1, ..., x_m)$ with an encrypted vector $(E(y_1), ..., E(y_m))$ by computing $\Pi E(y_i)^{x_i} = E(\sum x_i y_i))$.

2.3.2 Matrix Operators

Consider a matrix $A \in M_{k,l}(G)$. One can construct a $k \times (k+l-1)$ matrix A' by initializing A' as a matrix with all 0s and then, for each row $1 \leq i \leq k$, setting $(A'(i,i), ..., A'(i,i+l-1)) = (A(i,1), ..., A(i,l))$. We denote such a function by $A' \leftarrow Stretch(A)$, and note that since this function is a linear operator, it can be computed using matrix multiplication. We observe that for any encryption scheme E, $E(Stretch(A)) = Stretch(E(A))$, when E is applied to each entry in A.

Consider a matrix $A \in M_{k,l}(G)$. We denote by $Cut(A, j)$ as the matrix $A' \in M_{k,l-2j+2}$ such that for $1 \leq a \leq k$, $1 \leq b \leq l - 2j + 2$, $A'(a, b) = A(a, b + j - 1)$. In particular, such a function outputs the middle $l - 2j + 2$ columns of $M_{k,l}$. We note that Cut is a simple projection operator and is also computable by matrix multiplication. We observe that for any encryption scheme E, $E(Cut(A, j)) = Cut(E(A), j)$

Finally, consider a matrix $A \in M_{k,l}(G)$. We denote by $ColSum(A)$ the function that takes as input A and outputs a $1 \times l$ vector whose ith entry is the sum of all entries in the i column of A. In particular, $ColSum(A) = [1....1] \cdot A$. We observe that for any additively homomorphic encryption scheme E, $ColSum(E(A)) = E(ColSum(A))$.

Since we will be composing these functions, a shorthand for their composition will be convenient. For matrices $A \in M_{k,l}(G)$ and $B \in M_{l,m}(G)$, we denote the composition function $ColSum(Cut(Stretch(A \cdot B), j))$ by $PM_{5PM}(A, B, j)$.

2.3.3 Searching an Encrypted Vector, π_{VFind}

Suppose party P_1 possesses (pk, sk) for an additively homomorphic encryption algorithm E, and a single value $m \in G$ and P_2 possesses a vector of l distinct encryptions $E_{pk}(vec)$, where $vec = (x_1, \ldots, x_l) \in G^l$. Then P_1 can determine if $E(vec)$ contains an encryption of m while learning nothing else about vec, while P_2 cannot learn m, through the following protocol π_{VFind}:

(1) P_1 computes $E(-m)$ from $-m$. P_1 sends $E(-m)$ to P_2.

(2) P_2 computes $E(vec')$ by multiplying (via the group operation of the ciphertext space) $E(-m)$ to each encrypted entry in $E(vec)$. Note that an entry in $E(vec')$ will be an encryption of 0 if and only if one of the encryptions of $E(vec)$ was an encryption of m. P_2 computes $E(vec^r)$ from $E(vec')$ by exponentiating each encrypted entry of $E(vec')$ by an (independent) random exponent. P_2 sends $E(vec^r)$ to P_1.

(3) P_1 decrypts $E(vec^r)$ to obtain vec^r; if a 0 exists at position i, the ith position of $E(vec)$ is $E(m)$.

Note that if P_2 wishes to hide the position of $E(m)$ from P_1, P_2 could randomly permute the positions of $E(vec^r)$ and send the permuted vector to P_1.

2.3.4 Efficiently Determining Equality of Two Matrices, π_{VecEQ}

Suppose parties P_1 and P_2 have agreed upon an additively homomorphic threshold encryption scheme E_{th}. Further, suppose P_1 and P_2, possess encrypted matrices $E_{th}(A) \in M_{k,l}(G')$ and $E_{th}(B) \in M_{k,l}(G')$, respectively, where the message space G' is the group $\mathbb{Z}_q$, for a prime q. Let D_{P_i} denote the partial decryption algorithm of party P_i. P_1 and P_2 wish to determine if their encrypted matrices are equal without exchanging their decryptions. They can do so by hashing their encrypted matrices to a single group element and exchanging the outcome of the hashes. More specifically, an affine hash function $\mathbb{Z}_q^{kl} \rightarrow \mathbb{Z}_q$ can be specified by letting P_1 and P_2 jointly compute a uniformly random pair $(a, b) \in \mathbb{Z}_q^{kl} \times \mathbb{Z}_q$ using standard commitment techniques and setting the hash to $hf(x) = \langle x, a \rangle + b$, where $\langle \cdot, \cdot \rangle$ is the inner product over $\mathbb{Z}_q$ (here, we consider the matrices as kl-length strings). Note that such a hash

function can be computed on encrypted strings because the encryption scheme is additively homomorphic. Denote by *comm* a (perfectly hiding, computationally binding) commitment scheme; in practice we use Pedersen commitments [75]. We denote the following subprotocol by π_{VecEQ}:

(1) P_1 selects $(a_1, b_1) \in \mathbb{Z}_q^{kl} \times \mathbb{Z}_q$ uniformly at random and computes $E_{th}(b_1)$. P_1 computes and sends
$comm(a_1), comm(E_{th}(b_1)), comm(E_{th}(A))$ to P_2.

(2) P_2 selects $(a_2, b_2) \in \mathbb{Z}_q^{kl} \times \mathbb{Z}_q$ uniformly at random and computes $E_{th}(b_2)$. P_2 sends $a_2, E_{th}(b_2), E_{th}(B)$ to P_1.

(3) P_1 sets $a = a_1 + a_2$, $E_{th}(b) = E_{th}(b_1 + b_2)$ and computes $z_1 = E_{th}(\langle a, A\rangle + b)$, $z_2 = E_{th}(\langle a, B\rangle + b)$. P_1 decommits to a_1, $E_{th}(b_1)$ and $E_{th}(A)$ to P_2 and sends $D_{P_1}(z_1)$, $D_{P_1}(z_2)$ to P_2.

(4) P_2 aborts if it does not accept the decommitments, else P_2 sets $a = a_1 + a_2$, $E_{th}(b) = E_{th}(b_1 + b_2)$ and computes $z_1 = E_{th}(\langle a, A\rangle + b)$, $z_2 = E_{th}(\langle a, B\rangle + b)$. P_2 sends $D_{P_2}(z_1)$, $D_{P_2}(z_2)$, $D_{P_2}(D_{P_1}(z_1))$, and $D_{P_2}(D_{P_1}(z_2))$ to P_1.

(5) P_1 aborts if $D_{P_2}(D_{P_1}(z_1)) \neq D_{P_2}(D_{P_1}(z_2))$, otherwise P_1 sends $D_{P_1}(D_{P_2}(z_1))$ and $D_{P_1}(D_{P_2}(z_2))$ to P_2.

(6) P_2 aborts if $D_{P_1}(D_{P_2}(z_1)) \neq D_{P_1}(D_{P_2}(z_2))$.

The bandwidth complexity of π_{VecEQ} is dominated by the size of $E_{th}(A)$ (and $E_{th}(B)$). Only with probability $1/q$ will the decryptions equal each other when $A \neq B$ because the hash function is chosen uniformly at random. In the malicious case, arguments of consistency for correct partial decryptions will also be needed.

3 5PM Protocol

This section utilizes the above observations and cryptographic tools to construct the secure pattern matching protocol (5PM). We develop π_{5PM}^{H} for the honest-but-curious adversary model and π_{5PM}^{M} for the malicious (static corruption) adversary model.

Notation	Description	Section
π^H_{5PM}	Pattern matching algorithm secure in HBC adversary model	3.2
π^M_{5PM}	Pattern matching algorithm secure in malicious adversary model	3.3
Key	Key generation algorithm for homomorphic encryption scheme	2.2
E	Homomorphic encryption algorithm	2.2
D	Decryption algorithm for encryption scheme E	2.2
D_{P_i}	Partial decryption algorithm for party P_i using E for threshold encryption	2.2
$Stretch(A)$	Function from n by m matrices to n by $n+m-1$ matrices that "stretches" rows of A	2.3.2
$Cut(A,n)$	Function that outputs the first n columns of matrix A	2.3.2
$ColSum(A)$	Function that outputs the sums of the columns of matrix A	2.3.2
$PM_{5PM}(A,B,n)$	Composition function $ColSum(Cut(Stretch(A \cdot B),n))$	2.3.2
π_{VFind}	Two-party protocol that determines if encryption of P_1's value exists in P_2's encrypted vector	2.3.3
Gen_{CDV}	Algorithm with input of a pattern $p \in (\Sigma \cup \{*\})^m$ outputs $\lvert\Sigma\rvert \times m$ matrix M_{CDV}	3.1
Gen_T	Algorithm that on input of a text $T \in \Sigma^n$ outputs the $n \times \lvert\Sigma\rvert$ matrix M_T	3.1
π_{VecEQ}	Two-party protocol that determines equality of two encrypted vectors (π^M_{5PM})	2.3.4
A_{rel}	Arguments of consistency require for malicious protocol (π^M_{5PM})	3.3.2

Table 4.4: Notation used for 5PM protocols

3.1 Converting IPM to Linear Operations.

For a fixed alphabet Σ, a text $T \in \Sigma^n$, and pattern $p \in (\Sigma \cup \{*\})^m$, IPM can be represented in terms of linear operations described in Section 2.3 as follows:

(a) The text T can be transformed into an $n \times |\Sigma|$ matrix, M_T. The transformation is performed by applying a unary encoding of alphabet characters to T, i.e., $M_T(i, T_i) = 1, \forall i \in \{1, ..., n\}$; all other entries in M_T are 0. We denote the algorithm that computes M_T from T as $M_T \leftarrow Gen_{M_T}(T)$.

(b) The CDVs of alphabet characters can be grouped into a $|\Sigma| \times m$ matrix, M_{CDV}. This step is equivalent to constructing CDVs for alphabet characters (steps b and c in Section 2.1.1).We denote the algorithm that compute M_{CDV} from p as $M_{CDV} \leftarrow Gen_{M_{CDV}}(p)$.

(c) Multiply M_T by M_{CDV} to obtain an $n \times m$ matrix $M_{T(CDV)}$ that represents T row-wise

in terms of CDVs, where the ith row is $CDV(T_i)$. In reality, since M_T and M_{CDV} are 0/1 matrices, multiplication is more computationally expensive than necessary, and vectors can simply be selected (as shown in IPM description in Section 2.1).

(d) Compute $\overline{M}_{T(CDV)} = Stretch(M_{T(CDV)})$. This transformation, jointly with the previous step, constructs a matrix of CDVs where the ith row contains only $CDV(T_i)$, which starts in the ith position in the ith row (sets up step d in Section 2.1.1).

(e) Compute $AV = ColSum(Cut(\overline{M}_{T(CDV)}, m))$ to obtain the final activation vector AV of length $n - m + 1$. Entries in AV are checked to see if any are equal to the threshold value m, or $m - l$ for single character wildcards or substring matching (completes step d in Section 2.1.1).

A key observation is that if only one of M_T and M_{CDV} are encrypted, an encrypted activation vector, $E(AV)$ can be obtained by both parties as shown in Sections 2.3.1 and 2.3.2.

3.2 Honest-but-curious (HBC) 5PM Protocol

We begin by describing the intuition behind required modifications to secure IPM in the HBC adversary model. We then describe details of the HBC protocol, π^H_{5PM}.

3.2.1 Protocol Intuition

For an additively homomorphic encryption scheme E, if Client sends Server $E(M_{CDV})$, by the reasoning of Sections 2.3 and 3.1, since the pattern matching operation can be reduced to a sequence of linear operations (namely matrix multiplication and the functions $Stretch$, Cut, and $ColSum$), Server can compute $E(AV)$, an encrypted activation vector, using only M_T and $E(M_{CDV})$. Since Client sends only $E(M_{CDV})$ and $E(m - l)$, Server learns nothing about Client's pattern due to semantic security of the encryption scheme.

Next, Client, for pattern matching threshold m (or $m - l$ in the single character wildcards/substring matching case) executes π_{VFind} specified in Section 2.3.3, where Client uses $E(AV)$, to discover whether (and where) a pattern exists. By the security of π_{VFind}, Server

does not learn m and Client learns nothing about $E(AV)$ other than whether or not (and where, if the pattern matching locations are not hidden by Server) an encryption of m exists in $E(AV)$. In practice, Client sends $E(m)$ in the same (first) round as $E(M_{CDV})$, and Server's response to π_{VFind} occurs in the second round, concluding execution of the secure pattern matching protocol.

3.2.2 π^H_{5PM} Protocol Specification

Client		**Server**
Input: $p \in (\Sigma \cup \{*\})^m$		**Input:** $T \in \Sigma^n$
Initialization:		
1) $(pk, sk) \leftarrow Key(1^k)$		
2) $M_{CDV} \leftarrow Gen_{CDV}(p)$		
3) $E(M_{CDV}) \leftarrow M_{CDV}$		
$E(-m+l) \leftarrow -m+l$	$\xrightarrow{4) E(M_{CDV}); E(-m+l); pk_C}$	**Activation Vector Formation:**
		5) $M_T \leftarrow Gen_T(T)$; $E(AV_S) \leftarrow PM_{5PM}(M_T, E(M_{CDV}), m)$
		6) $E(AV_S^r) \leftarrow \pi_{VFind}(E(AV_S, E(-m+l)))$
		7) Optional: Permute $E(AV_S^r)$
Decrypting and Determining	$\xleftarrow{6) E(AV_S^r)}$	
Matches:		
8) $MP = \{i \mid D(E(AV_S^r[i])) = 0\}$		
Output:		**Output:**
$MP = \{j \mid T_j ... T_{j+m-1} = p\}$		Nothing

Table 4.5: Overview of 5PM protocol for HBC adversary model, π^H_{5PM}. See Table 4.4 for notation.

Recall that, over a specified alphabet Σ, Server holds text $T \in \Sigma^n$ and Client holds a pattern $p \in (\Sigma \cup \{*\})^m$. The output of Server is an encrypted activation vector $E(AV)$ of length n. We refer the reader to Sections 3.1 and 2.3.2 for the notation used here. The protocol operation is as follows:

(1) Client computes $(sk, pk) \leftarrow Key(1^k)$ using the key generation algorithm of an additively homomorphic encryption scheme, E.

(2) Client computes $M_{CDV} \leftarrow Gen_{CDV}(p)$. In the case where Client wishes to hide the length of p, Client computes M_{CDV} for the pattern p' equal to the concatenation of p with $*^{n-m}$.

(3) Client encrypts M_{CDV} entry-wise using public-key pk to obtain $E(M_{CDV})$.

(4) Client sends $E(M_{CDV})$ and pk to Server. In addition, Client sends $E(-m)$ (or $E(-m+l)$ in the single character wildcards or substring matching cases).

(5) Server computes $M_T \leftarrow Gen_T(T)$. Server computes $E(AV) = E(PM_{5PM}(M_T, M_{CDV}, m))$, which is computed as specified in Section 2.3.1 and Section 2.3.2.

(6) Server executes round 2 of π_{VFind} (see Section 2.3.3) using $E(-m)$ and $E(AV)$. Server sends output of the subprotocol, denoted $E(AV_S^r)$, to Client.

(7) Optional: Per π_{VFind}, Server randomly permutes $E(AV_S^r)$ to hide possible pattern match locations.

(8) Client executes round 3 of π_{VFind} using $E(AV_S^r)$ to determine results of the pattern matching.

We note that π_{5PM}^H can perform substring matching for multiple substring lengths (such as for a Hamming distance bound) simultaneously by sending multiple $E(m-l)$ values at step 6 in the above specification. Then, for each value of l, Server constructs a distinct $E(AV)$ and sends Client a distinct corresponding $E(AV_S^r)$ indicating matching locations for that l value. In particular, π_{5PM}^H does not require multiple independent protocol executions to compute substring matching for a range of substring length values. In addition, π_{5PM}^H can simply compute the Hamming distance of the pattern with each consecutive m positions of the text by simply not executing π_{VFind} and sending the output of the protocol at step 5, and Client can decrypt to obtain all of the Hamming distance values between the pattern and the text.

Theorem 3.1. *Given an additively homomorphic semantically secure encryption scheme over a prime-order cyclic group (Key, E, D), π_{5PM}^H is secure in the HBC model.*

See Section 7 for a detailed security proof.

3.3 Malicious Model 5PM Protocol

In this section, we explain how to modify π^H_{5PM} to obtain a protocol, π^M_{5PM}, which is secure in the malicious (static corruption) model. We describe an instantiation of π^M_{5PM} based on additively homomorphic threshold ElGamal encryption (see Section 2.2) for concreteness; generalization to other encryption schemes follows provided they have efficient Σ protocols for the statements required here. First, we explain intuition behind π^M_{5PM}. Second, we give interactive zero-knowledge consistency arguments that will be required. Finally, we divide π^M_{5PM} into 6 subprotocols and describe their construction and how they are combined into the final protocol π^M_{5PM}. In the interest of clarity and space, we leave the exact protocol specification and security proof to Sections 6 and 7, respectively, of this paper. Note that this protocol, as noted in Section 2.1.2, can be modified to both hide pattern length (by using, for pattern p, the pattern p' equal to to p concatenated with $*^{n-m}$) and also to match against multiple substring values without multiple executions of the entire protocol (i.e., by sending multiple $E(m-l)$ values and computing a new activation vector for each value).

3.3.1 Protocol Intuition

The 8 round protocol for the malicious model, π^M_{5PM}, consists of the following six subprotocols:

(1) π_{encr}: initializes an additively homomorphic threshold encryption scheme.

(2) $\pi_{S,AV}$: allows Server to construct an encrypted activation vector for Client's encrypted pattern and Server's text.

(3) $\pi_{C,AV}$: allows Client to construct an encrypted activation vector for Client's pattern and Server's encrypted text.

(4) π_{vec}: allows Client and Server to verify that their activation vectors are equal without revealing them.

(5) π_{rand}: allows Server to send an encryption of its randomized activation vector to Client.

(6) π_{ans}: demonstrates to Client where the pattern matches the text (if at all).

The intuition behind constructing π^M_{5PM} is as follows: in π^H_{5PM}, only Server performs the computation to obtain the activation vector, AV. In the malicious setting, Client has to verify that Server correctly computed AV. Since Server performs $O(nm)$ multiplications when computing AV in π^H_{5PM}, requiring a zero-knowledge argument for each multiplication therefore would require bandwidth of at least $O(nm)$. Such overhead is unacceptable if bandwidth $O(n+m)$ is desired.

We utilize a more bandwidth-efficient approach to ensure that a malicious Server has computed the correct AV: in π^M_{5PM}, both Client and Server perform secure pattern matching independently using the function PM_{5PM} where one of M_{CDV} and M_T are encrypted, and then compare their results. Each party computes an AV in parallel (see subprotocols $\pi_{C,AV}$ and $\pi_{S,AV}$, respectively, in Section 3.3.3) using an additively homomorphic threshold encryption scheme (instantiated using subprotocol π_{encr} in Section 3.3.3). To ensure that no cheating has occurred, Client and Server then check that each other's AV was computed correctly. Therefore, proving that Server has behaved honestly is reduced to proving that Client and Server have obtained the same result from matching p against T. To efficiently perform comparison of encrypted AVs, Client and Server check that their encrypted AVs are equal using subprotocol π_{VecEQ} described in Section 2.3.4 (in addition to some zero-knowledge arguments to demonstrate well-formedness). Only if hashed AV values match will Server provide Client with its decrypted (and blinded) AV (using the subprotocols π_{rand} and π_{ans} in Section 3.3.3). The comparison subprotocol is denoted by π_{vec} in Section 3.3.3.

Throughout, both Client and Server will have to use various arguments of consistency outlined in Section 3.3.2 to prove that they have not deviated from the protocol.

There is one additional technical difficulty that we have to overcome: in order to prove security we must provide simulators that simulate transcripts when interacting with adversarial parties (see Section 7 for security definitions and simulator constructions). When constructing the Simulator for Client's view, Simulator receives the actual answer that it must provide to Client from the ideal functionality only at the last moment (if Client does

not abort). Thus, the Simulator must provide a final answer which is not consistent with the previous interactions, while the real Server must be unable to do so. To achieve this, we demonstrate that the Simulator can extract the knowledge of the exponent of some h^* specified by Client during the first subprotocol (π_{encr}); then, the final subprotocol (π_{ans}) utilizes a zero-knowledge argument of knowledge that demonstrates that either the final randomized AV is correct *o*r that Server knows the discrete logarithm of h^*. Since a real Server cannot extract the discrete logarithm of h^* but the Simulator can by construction, this allows the Simulator to reveal the correct randomized AV even when it is inconsistent with the previous outputs of the conversation. We stress that we do not use NP-reductions and rather build highly efficient protocols to fit our needs.

3.3.2 Zero-Knowledge Arguments of Knowledge (ZK-AoKs) of Consistency

We first describe five required interactive arguments which we rely on to prove statements required for the π_{5PM}^{M} protocol. They are designed for use with the specified threshold ElGamal encryption scheme (Section 2.2). We apply a standard construction outlined in Section 4 of this paper to transform three-move arguments of knowledge and construct five-move ZK arguments of knowledge π_{DL}, π_{isBit}, π_{eqDL} and π_{fin}, respectively. All ZK-AoKs are executed between a prover P and a verifier V in five moves; we note that either Client or Server may execute the arguments of consistency as P while the other party will then execute as V. π_{DL} is the only ZK-AoK used on its own in π_{5PM}^{M}; it proves knowledge of a discrete logarithm of a public $h = g^x$. π_{isBIT} is a ZK-AoK that proves that an encryption is either of a 0 or of a 1, π_{eqDL} is a ZK-AoK that proves that two discrete logarithms are equal, and π_{fin} is a ZK-AoK that proves that *either* two discrete logarithms are equal *or* that P knows the discrete logarithm of a public $h = g^x$. The five required interactive arguments are:

(1) $\mathbf{A_{M01}}$*, an AoK of Consistency for Matrix formation 0/1:* P, for an $l \times u$ matrix of encryptions, $E(M)$, proves to V that each column of $E(M)$ contains encryptions of 0 and at most one 1.

(2) $\mathbf{A_{M1}}$, *an AoK of Consistency for Matrix formation* 0/1-1*: P*, for an $l \times u$ matrix of encryptions, $E(M)$, proves to V that each row of $E(M)$ contains encryptions of 0 and *exactly* one 1.

(3) $\mathbf{A_{PD}}$, *an AoK of Consistency for Partial Decryption: P*, for a vector of l encryptions, (x_i, y_i) and a vector of their l partial decryptions (x'_i, y'_i), proves to V that the partial decryptions are correctly constructed.

(4) $\mathbf{A_{Rand}}$, *an AoK of Consistency for Randomization: P*, for a vector of l encryptions (x_i, y_i) and a vector of their exponentiations, $(x_i^{r_i}, y_i^{r_i})$, proves to V that P knows r_i for each i.

(5) $\mathbf{A_{FD}}$, *an AoK of Consistency for Final Decryption: P*, for a vector of l encryptions (x_i, y_i), their partial decryptions (x'_i, y'_i), and some g^w, proves to V that either P has computed all the partial decryptions correctly *or* that possesses the discrete logarithm w of g^w.

3.3.3 π_{5PM}^{M} Protocol Outline

We provide the details of π_{5PM}^{M} by describing individual subprotocols that constitute it, π_{encr}, $\pi_{S,AV}$, $\pi_{C,AV}$, π_{vec}, π_{rand} and π_{ans}. These subprotocols utilize the interactive arguments described in Section 3.3.2 to prove various statements of consistency. We denote by $comm(s)$ as the (perfectly hiding, computationally binding) commitment of s, which using Pedersen commitments [75] is $g^s h^r = comm(g, h, r, s)$. For the exact protocol specification of π_{5PM}^{M}, including precisely how the subprotocols are interleaved so that π_{5PM}^{M} requires only 8 rounds, see the Section 6.2; we will however mention here during which global rounds (1 through 8) these subprotocols occur.

We remark that in our construction of ZK arguments of knowledge from Σ protocols, whenever a ZK subprotocol is required, the first two rounds of the five round protocol can be completed in parallel at the very beginning of the overall protocol π_{5PM}^{M}. Such "preprocessing" does not affect security. Further, knowledge extraction used in the security proofs is not affected by this preprocessing.

$\pi_{\mathbf{encr}}$ is a two party protocol executed between Client and Server that initializes an additively homomorphic threshold encryption scheme (e.g., ElGamal) and also sets up an independent "trapdoor" s^* alluded to in Section 3.3.1 and required for the simulator in the security proof. In the ElGamal case, for simplicity, we assume that Client and Server have already agreed on appropriate prime q such that $\log q = O(k)$, $\mathbb{G}_q$ and $g \in \mathbb{G}_q$. This subprotocol begins at the first global round and ends at global round 6. Client chooses its secret-key s_C and trapdoor s^*, and sets $h_1 \leftarrow g^{s_C}$, $h^* \leftarrow g^{s^*}$. Client sends h_1, h^* to Server. Client executes two parallel instantiations of π_{DL} proving knowledge of the discrete logs of h_1 and h^* (i.e., s_C and s^*). Then, Server chooses its secret-key s_S, sets $h_2 \leftarrow g^{s_S}$, and sends h_2 to Client and executes π_{DL} proving knowledge of the discrete logarithm of h_2 (i.e., s_S). Both parties set the public-key to be $h = h_1 h_2 = g^{s_C + s_S}$.

$\pi_{\mathbf{C,AV}}$ is a two party protocol executed between Client and Server which outputs to Client an encrypted activation vector $E(AV_C)$ corresponding to matching Client's p against Server's T. This subprotocol starts at global round 2 and ends at global round 6. First, Server constructs $M_T \leftarrow Gen_{M_T}(T)$ as specified in Section 3.1. Then, Server encrypts M_T and sends $E(M_T)$ to Client. Server also executes, for $E(M_T)$, A_{M1} to prove that $E(M_T)$ is formatted correctly (namely, that each row of $E(M_T)$ has one encryption of a 1 per row and encryptions of 0 everywhere else- therefore each row of $E(M_T)$ corresponds to the encoding of exactly one element of the alphabet Σ). Client then obtains $E(AV_C)$ by computing $E(PM_{5PM}(M_T, M_{CDV}, m))$ (see Section 2.3.2) and then multiplying each encryption by $E(-p_t)$ (where p_t is the pattern matching threshold), observing the function PM_{5PM} can be computed using encrypted $E(M_T)$.

$\pi_{\mathbf{S,AV}}$ is a two party protocol executed between Client and Server which outputs to Server an encrypted activation vector corresponding to matching Client's p against Server's T. This subprotocol starts at global round 3 and ends at global round 5, with ZK preprocessing occurring during global rounds 1 and 2. Client encrypts M_{CDV} and p_t and

sends $E(M_{CDV})$ and $E(p_t)$ to Server. Client also executes A_{M01} to prove that $E(M_{CDV})$ is formatted correctly (namely, $E(M_{CDV})$ consists of at most one encryption of 1 per column and consists of encryptions of 0 everywhere else, therefore ensuring that there is at most one character delay value per distance). Server computes $E(AV_S)$ by computing $E(PM_{5PM}(M_T, M_{CDV}, m))$ and then multiplying each encryption by $E(-p_t)$ (this slightly differs from Server's actions during π^H_{5PM} since the consistency proof of π_{vec} must also include subtraction of the pattern matching threshold p_t).

$\boldsymbol{\pi_{\mathbf{vec}}}$ is a two party protocol executed between Client and Server that outputs to each party whether their respective encrypted activation vectors are equal (without revealing their values). This subprotocol begins at global round 3 and ends at global round 8, with ZK preprocessing occurring during global rounds 1, 2 and 3. Client computes $E(AV'_C)$ by multiplying each element of AV_C with an encryption of 0; Server computes $E(AV'_S)$ from $E(AV_S)$ similarly. Client and Server execute π_{VecEQ} (see Section 2.3.4) where Client has input $E(AV'_C)$ and Server has input $E(AV'_S)$. In addition, whenever a party sends the other a partial decryption, they execute A_{PD} to prove that the execution is well formed. Note that the probability that π_{VecEQ} will complete without abort for unequal vectors AV_S and AV_C is negligible ($\frac{1}{q}$).

$\boldsymbol{\pi_{\mathbf{rand}}}$ is a two party protocol executed between Client and Server that outputs to Client an encrypted vector $E(AV^r_S)$ that contains randomizations of the values in non-matching (non-zero) positions in $E(AV'_S)$. This subprotocol starts at global round 6 and ends at global round 8, with ZK preprocessing occurring during global rounds 2 and 3. Server computes $E(AV^r_S)$ from $E(AV'_S)$ by exponentiating each encryption in $E(AV'_S)$ by a random value. Server sends $E(AV^r_S)$ to Client and executes A_{rand} to prove that $E(AV^r_S)$ was obtained correctly from $E(AV'_S)$.

$\boldsymbol{\pi_{\mathbf{ans}}}$ is a two party protocol executed between Client and Server that outputs to Client the randomization, AV^r_S, of Server's activation vector AV_S. Note that AV^r_S will have a 0

wherever there is a match; every non-matching entry will contain a random element. Client is assumed to already know $E(AV_S^r)$. This subprotocol starts at global round 6 and ends at global round 8, with ZK preprocessing occurring during global rounds 2 and 3. We present a slightly modified version of the actual subprotocol used because this protocol in practice must be rearranged slightly to keep π_{5PM}^M at 8 rounds (see Section 6.2 for details). Server sends $D_S(E(AV_S^r))$ to Client and executes A_{FD} to prove that either $D_S(E(AV_S^r))$ was obtained correctly or that Server knows s^* (for h^* sent by Client in the first round of π_{encr}). Client aborts if it does not accept A_{FD} and otherwise obtains AV_S^r by computing $D_C(D_S(E(AV_S^r)))$.

Protocol Efficiency and Security: Overall bandwidth of π_{5PM}^M is dominated by the $O(m|\Sigma|)$ encrypted values that Client sends to Server in $\pi_{S,AV}$ and $O(n|\Sigma|)$ encrypted values that Server sends to Client in $\pi_{C,AV}$ and π_{ans}. Since alphabet size, $|\Sigma|$, is constant, we obtain the desired bandwidth, including the ZK protocols, of $O((m+n)k^2)$ for security parameter k and total number of encryptions of $O(m+n)$. In particular, when Client hides pattern size, the corresponding pattern will have length n and therefore the bandwidth complexity is $O(nk^2)$. Computational complexity for Client is dominated by the subprotocol $\pi_{C,AV}$ where Client performs $O(mn)$ exponentiations of encrypted elements, and computational complexity for Server is dominated by subprotocols $\pi_{S,AV}$, where Server performs $O(mn)$ multiplications of encrypted elements, and π_{vec} and π_{ans}, where $O(nk)$ exponentiations are needed for the ZK protocols.

Theorem 3.2. *Assuming that the Decisional Diffie-Hellman (DDH) problem is hard, π_{5PM}^M is secure in the malicious (static corruption) model.*

See Section 7 for a detailed security proof.

4 Converting Σ protocols to Zero-Knowledge Arguments of Knowledge (ZK-AoKs)

We describe here the construction used to convert a Σ protocol into an efficient zero-knowledge argument of knowledge. We first provide the necessary definitions. We then give a construction of an efficient extractable equivocable commitment scheme. We finally use this scheme to construct a zero-knowledge argument of knowledge from a Σ protocol for the same relation. We note that an algorithm is expected PPT if it is a probabilistic algorithm that runs in expected polynomial time.

4.1 Definitions

Let R be a binary relation where for all $(x, w) \in R$, $|w| \in poly(|x|)$. w is called the *witness* for x. Consider an interactive argument consisting of a pair of PPT algorithms (P, V) (thought of as probabilistic next message functions). x is known to both P and V while w is only known to P. Informally, P proves to V that there is a w such that $(x, w) \in R$. We consider interactive protocols that have the following specification:

(a) P sends message a, $|a| \in poly(|x|)$.

(b) V selects message $e \in \{0,1\}^{poly(|x|)}$ uniformly at random and sends e to P. We denote e as the *challenge*.

(c) P sends a reply $z \in \{0,1\}^{poly(|x|)}$.

Note that this interaction is public coin for V. Based on the tuple (also called a *conversation*) (a, e, z), V either *accepts* or *rejects*. For any x, we call a conversation (a, e, z) that V accepts an *accepting conversation*.

Definition 4.1. *A 3-move interactive protocol $\Pi = (P, V)$ of the above form is said to be a Σ protocol for a relation R if it satisfies the following properties:*

(a) Completeness: *On common input x, if the honest prover P has as private input w such that $(x, w) \in R$, then honest V always accepts.*

(b) Special Soundness: *For any common input x and any pair of accepting conversations*

(a, e, z) *and* (a, e', z') *for* x *where* $e \neq e'$*, there exists a* w *that can be computed in polynomial time such that* $(x, w) \in R$*.*

(c) Special Honest-Verifier Zero-Knowledge (SHVZK): *There exists a PPT* M *that on input* x *and a properly formatted* e *outputs an accepting conversation of the form* (a, e, z) *with the same probability distribution (over* e*) as conversations between honest* P *and honest* V*.*

Definition 4.2. *An interactive protocol for a relation* R *consisting of a pair of PPT algorithms* (P, V) *is an* argument of knowledge *with knowledge error* κ *if the following properties are satisfied:*

(a) Completeness: *On common input* x*, if the honest prover* P *has as private input* w *such that* $(x, w) \in R$*, then honest* V *always accepts.*

(b) Knowledge Soundness: *There exists a (expected) PPT* E *called the* knowledge extractor *which, given input* x *and oracle (black-box) access to* P*, attempts to compute* w *such that* $(x, w) \in R$*. For any prover* P^**, let* $\epsilon(x)$ *be the probability that* V *accepts on input* x*. Then there exists a constant* c *such that whenever* $\epsilon(x) > \kappa(x)$*,* E *will output a correct* w *in expected time at most* $\frac{|x|^c}{\epsilon(x)-\kappa(x)}$ *where an individual oracle call to* P^* *is considered as one step.*

κ can be thought of as the probability that V can be convinced there exists a w such that $(x, w) \in R$ even if such a pair does not exist.

Lemma 4.3 (Damgard [24])**.** *Let* Π *be a* Σ *protocol for relation* R *where the challenge* e *is drawn uniformly at random from* $\{0, 1\}^t$*. Then* Π *is a proof of knowledge with knowledge error* 2^{-t}*.*

Remark 4.4. Lemma 4.3 holds because Definition 4.1 includes the special soundness property. Σ protocols that only have standard soundness will not always satisfy the lemma.

Definition 4.5. *For any binary relation* R*, an interactive protocol consisting of a pair of PPT algorithms* (P, V) *is a* zero-knowledge argument *if it satisfies the following properties:*

(a) Completeness: *On common input x, if the honest prover P has as private input w satisfying $(x, w) \in R$, then honest V always accepts.*

(b) Soundness: *For all x such that there does not exist a w with $(x, w) \in R$, V will only accept with negligible probability.*

(c) Zero-Knowledge: *For all PPT V^*, there is a PPT simulator M with oracle access to V^* such that, given input x and V^*'s auxiliary input, V^*'s view of its interaction with real P is computationally indistinguishable from V^*'s view of its interaction with M.*

We demonstrate that any Σ protocol for a binary relation R can be converted into a ZK argument of knowledge for R. We first construct an extractable equivocable commitment scheme and use this scheme together with the Σ protocol specification for the ZK-AoK construction.

4.2 Extractable Equivocable Commitment Schemes

To construct a ZK-AoK from a Σ protocol, an efficient *extractable equivocable commitment scheme* will be required. Such a scheme is an interactive protocol between a PPT committer C and a PPT receiver R consisting of three functions: $EComSet$ instantiates the commitment scheme, com computes the commitment, and $ECom Ver$ verifiers that decommitment is valid. More specifically, R, for a security parameter k, computes $(pk, t) \leftarrow EComSet(1^k)$ and sends pk to C. C computes $c \leftarrow com(s, r, pk)$ for message s and randomness r and sends c to R as its commitment to m. To decommit, C sends (r, m) to R. R computes $\{0, 1\} \leftarrow ECom Ver(m, r, c, pk)$, *accepts* if 1 is output and *rejects* otherwise.

Definition 4.6. *A* computationally binding equivocable commitment scheme *is a pair of PPT algorithms (R, C) that interact as above and satisfy the following properties.*

(a) Statistically Hiding: *For pk correctly constructed and any messages s and s', the distributions of $com(s, r, pk)$ and $com(s', r', pk)$ are statistically indistinguishable over the choice of random input (e.g,r and r').*

(b) Computationally Binding: *For any PPT algorithm C running in expected time polynomial in k, the probability that C on input pk can compute a tuple (s, r, s', r') such*

that $com(s, r, pk) = commit(s', r', pk)$ *with* $s \neq s'$ *is negligible in* k.

(c) Equivocable: *There is a PPT algorithm* S *that, on inputs* t, pk, *any commitment* c *and any accepting decommitment* (s, r) *to* c, *can construct for any valid* s' *an* r' *such that* $c = com(s', r', pk)$.

An equivocable commitment scheme is extractable *if there is a PPT algorithm* E *that, upon oracle access to* R, *is able to obtain a trapdoor* t *in expected polynomial time.*

We now give a construction of an equivocable commitment scheme, *EP*, based on Pedersen commitments [75]. We assume that the receiver R and committer C have already agreed on a prime order group $\mathbb{G}_q$ and generator $g \in \mathbb{G}_q$. The committer C has a message s. For $b \in \{0, 1\}$, we denote $\bar{b} = 1 - b$.

EP - Commitment:

EP-1: R chooses for $1 \leq i \leq k$ and $0 \leq j \leq 1$, $x_{1,0}, x_{1,1}, \ldots, x_{k,0}, x_{k,1} \in \mathbb{Z}_q$ uniformly and independently at random and sets $h_{i,j} \leftarrow g^{x_{i,j}}$. R sends $(h_{1,0}, h_{1,1}, ..., h_{k,0}, h_{k,1})$ to C.

EP-2: C chooses $e \in \{0, 1\}^k$. C chooses $r_1, ...r_k \in \mathbb{Z}_q$ uniformly and independently at random. C sets $c = h_{1,e_1}^{r_1} \cdot ... \cdot h_{k,e_k}^{r_k} \cdot g^s \leftarrow com_{\mathbb{G}_q, g, h_{1,e_1}, ..., h_{k,e_k}}(s, r_1, ..r_k)$. C sends e and c to R.

EP-3: R sends $(x_{1,\bar{e}_1}, ..., x_{k,\bar{e}_k})$ to C.

C checks that for $1 \leq i \leq k$, $h_{i,\bar{e}_i} = g^{x_{i,\bar{e}_i}}$. If not, C aborts.

EP - Decommitment

EP-4: C sends $s, r_1, ..., r_k$.

EP-5: R verifies that $c = h_{1,e_1}^{r_1} \cdot ... \cdot h_{k,e_k}^{r_k} \cdot g^s$ and aborts if equality does not hold.

The above *EP* protocol has bandwidth complexity $O(k^2)$ and computational complexity $O(k^2 \log^2 k)$.

Just like Pedersen commitments, this commitment scheme is statistically hiding and computationally binding.

We show that knowledge of *a*ny discrete logarithm $x_{i,j}$ from the public-key together

with a valid decommitment would allow S to open commitments to any value. Indeed, let the public-key be $(h_{1,e_1}, ..., h_{k,e_k})$, where $h_{i,e_i} = g^{x_i}$. Without loss of generality, suppose S obtained x_1 and also obtained $c, s, r_1, ..., r_k$ such that $c = com(s, r_1, ..., r_k, pk)$. Let s' be any message. S sets $r'_i = r_i$ for $2 \leq i \leq k$ and sets $r'_1 = \frac{s+x_1 r_1 - s'}{x_1} \mod q$. Then $com(s', r'_1, r_2, ..., r_k) = c = com(s, r_1, ..., r_k)$.

To demonstrate that this scheme is extractable, for any R there we construct a simulator M_R. M_R runs EP honestly with R through **EP-3**, then rewinds to **EP-2** and sends a new $e' \in \{0,1\}^k$ to R. Since $e' \neq e$, M_R obtains some discrete logarithm $x_{i,j}$ of the public-key $(h_{1,e_1}, ..., h_{k,e_k})$ and therefore can decommit to any s'. We require so many possible trapdoors because the probability that R can both complete the EP protocol (namely the step **EP-3**) and not know or abort when asked for the discrete logarithm of any of the $h_{i,j}$s is roughly the same probability that M_R will fail to extract a trapdoor since knowledge of the discrete logarithms occurs at **EP-3**; therefore we require many trapdoors (i.e. discrete logarithms) to make the probability that R sends an invalid response at **EP-3** negligible in the security parameter.

4.3 Construction of a ZK-AoK from Σ Protocols

We give a construction for how to transform a three-move Σ argument of knowledge Σ_{rel} for a binary relation R_{rel} into a five-move ZK argument of knowledge π_{rel} for R_{rel} using the extractable equivocable commitment scheme EP described in Section 4.2. Recall from Section 4.1 that we denote the transcript for a Σ protocol as (a, e, z), where e is chosen uniformly at random by V. We assume that V and P have already agreed on a multiplicative group $\mathbb{G}_q$ with prime order q and generator $g \in \mathbb{G}_q$. P $(x, w) \in R_{rel}$; V possesses x. We denote the following construction by Σ-ZK-AoK, which consists of the following steps:

rel-1: P executes **EP-1** acting as receiver.

rel-2: V selects e according to the second message of Σ_{rel}. V executes **EP-2** using e as the value being committed.

rel-3: P computes a according to the first message specification of Σ_{rel}. P executes **EP-3**

and also sends a to V.

rel-4: V executes **EP-4**, opening e.

rel-5: P executes **EP-5** and aborts if P does not accept. P computes z according to the third message specification of Σ_{rel} in response to a, e and x. P sends z to V.

rel-6: V verifies (a, e, z) according to Σ_{rel}.

π_{rel} has bandwidth complexity $O(k^2)$ and computational complexity $O(k^2 \log^2 k)$ in addition to that of Σ_{rel}.

Lemma 4.7. *If Σ_{rel} is a Σ-protocol, then π_{rel} is a ZK argument of knowledge.*

Proof: Completeness. Completeness follows from the completeness of Σ_{rel}.

Soundness. To demonstrate soundness, assume that there exists an x such that there is no w with $(x, w) \in R_{rel}$ and yet V accepted π_{rel} with non-negligible probability. Let (a, e, z) be the transcript for Σ_{rel} contained within π_{rel}. Then it follows that Σ_{rel} has a verifier V that accepts a transcript with non-negligible probability for the same x. This implies that there are at least two distinct challenges e and e' such that P can produce accepting transcripts (a, e, z) and (a, e', z') for Σ_{rel} within π_{rel} (in fact, there must be a non-negligible number of such challenges). However, by special soundness of Σ_{rel}, a w can be computed in polynomial time from these transcripts such that $(x, w) \in R_{rel}$. But such a w does not exist, which leads to a contradiction.

To demonstrate that π_{rel} is zero-knowledge, for any verifier V^*, we describe the simulator M_V. M_V acts as an honest prover for steps **rel-1** through **rel-4**. At step **rel-4**, M_V receives V^*'s challenge e. M_V then, by the SHVZK property of Σ_{rel}, computes (a, z) such that (a, e, z) is an accepting transcript for x; note that in particular, the fact that Σ protocols are *special* honest verifier zero knowledge is important, as it implies the ability to construct correct transcripts for arbitrary (pre-selected) distributions of verifier messages. M_V rewinds to step **rel-3** where it sends a (as well as executes **EP-3**) and executes the rest of π_{rel} honestly. In particular, M_V sends z at step **rel-5**. V^*'s view of its interaction with P is indistinguishable from its view of its interaction with M_V because V^* cannot affect the distribution of its challenges based on P's messages since V^* commits to its challenge (in a perfectly binding

fashion) before it receives the first message of Σ_{rel}. Since the distribution of e is not affected by initial messages, M_V's transcript of Σ_{rel} within π_{rel} is computationally indistinguishable from P's output for Σ_{rel} by the special honest-verifier zero-knowledge (SHVZK) property of Σ_{rel}. Computational indistinguishability of the whole transcript follows.

To show the existence of a knowledge extractor, E_P, for each P, let $E_{rel,P}$ be the knowledge extractor for Σ_{rel} and let S_P be the trapdoor extractor for the commitment scheme EP. E_P then runs ZK-AoK using S as a subprotocol to extract the trapdoor for EP. E_P then rewinds to **rel-4**, after P has already instantiated the commitment scheme and sent is initial message a for Σ_{rel}, and changes its challenge for Σ_{rel} according to the specification of $E_{rel,P}$. Note that E_P will have to decommit to multiple challenges for Σ_{rel} at step **rel-4** in order to execute $E_{rel,P}$ as a subprotocol. However, since E_P possesses the trapdoor for EP and EP is equivocable, E_P can decommit (e.g,construct messages for **EP-4**) to whatever challenge $E_{rel,P}$ specifies. Since S can extract the trapdoor in polynomial time and E_{rel_P} can extract the witness for Σ_{rel} in (expected) polynomial time, E_P can extract the witness for π_{rel} in (expected) polynomial time.

Remark 4.8. We note that above, the zero-knowledge simulator M_V was able to interact with V without actually knowing the witness w for x. This is because of the simulation soundness of Σ_{rel}; namely, since a simulator can produce accepting transcripts only seeing the Verifier's challenge there (and without seeing w), M_V can produce proper transcripts for π_{rel} without ever knowing w. Such a property is called *simulation soundness* and will be useful for a security reduction needed for π^M_{5PM} (see Section 7.3.3).

5 Required Σ protocols

We outline in this section specific Σ protocols needed for the malicious model version of $5PM$, π^M_{5PM}. These three-move protocols are executed between a PPT prover (P) and a PPT verifier (V) and are used to construct zero-knowledge arguments of knowledge using the transformation in Section 4.3. For each Σ protocol we first describe the relation

demonstrated by P then the three protocol messages exchanged between P and V. We note that with the exception of Σ_{fin}, the security of each of the following Σ protocols is proven in the places in which they are cited; the security of Σ_{fin} follows since it is a standard example of an OR Σ protocol of two Σ protocols already shown here (for more, see [24]).

1- Σ_{DL}, Proving Knowledge of Discrete Logs [79]: For g and $h = g^x$, P demonstrates knowledge of witness x. The relation R_{DL} is $((\mathbb{G}_q, q, g, h), x) \in R_{DL}$ if $h = g^x$. The Σ protocol steps are:

- Σ_{DL-1}: P chooses $r \in \mathbb{Z}_q$ and sets $a \leftarrow g^r$. P sends a to V.
- Σ_{DL-2}: V chooses a challenge $c \in \mathbb{Z}_q$ and sends c to P
- Σ_{DL-3}: P sets $z \leftarrow r + cx$ and sends z to V.
- Σ_{DL-4}: V checks that $g^z = ah^c$ and aborts if not.

2- Σ_{eqDL}, Proving Equality of Discrete Logs [21]: For $g, h, x, y \in \mathbb{G}_q$, P demonstrates knowledge of a witness w such that $x = g^w$ and $y = h^w$. The relation R_{eqDL} is $((\mathbb{G}_q, q, g_1, g_2, h_1, h_2), w) \in R_{eqDL}$ if $h_1 = g_1^w$ *and* $h_2 = g_2^w$. The Σ protocol steps are:

- Σ_{eqDL-1}: P chooses $r \in \mathbb{Z}_q$ and sets $(a, b) \leftarrow (g^r, h^r)$. P sends (a, b) to V.
- Σ_{eqDL-2}: V chooses $c \in \mathbb{Z}_q$ and sends c to P.
- Σ_{eqDL-3}: P sets $z \leftarrow r + wc$ and sends z to V
- Σ_{eqDL-4}: V checks that $g^z = ax^c$ and that $h^z = by^c$ and aborts if not.

3- Σ_{isBit}, Proving Encryption of 0 or 1 [20]: P demonstrates that it possesses an ElGamal encryption of $m \in \{0, 1\}$ with generators g, public-key h and randomness r (recall that encryption of m is of the form $(g^r, g^m h^r) = (x, y)$). P proves that either $\log_g x = \log_h y$ or $\log_g x = \log_h y/g$. The relation R_{isBit} is $((\mathbb{G}_q, q, g, h, \alpha, \beta), (b, r)) \in R_{isBit}$ if $(\alpha, \beta) = (g^r, g^b h^r)$ *and* $b \in \{0, 1\}$. The Σ protocol steps are:

- $\Sigma_{isBit-1}$:
  - If $m = 1$: P chooses $r_1, d_1, w_2 \in \mathbb{Z}_q$ and sets $a_1 \leftarrow h^{r_1}(gh^r)^{-d_1}$, $a_2 \leftarrow h^{w_2}$.
  - If $m = 0$: P chooses $w_1, r_2, d_2 \in \mathbb{Z}_q$ and sets $a_1 \leftarrow h^{w_1}$, $a_2 \leftarrow h^{r_2}(h^r g^{-1})^{-d_2}$.

P sends (a_1, a_2) to V.

- $\Sigma_{isBit-2}$: V chooses $c \in \mathbb{Z}_q$ and sends c to P.
- $\Sigma_{isBit-3}$:
  - If $m = 1$: P sets $d_2 \leftarrow c - d_1$, $r_2 \leftarrow w_2 + rd_2$.
  - If $m = 0$: P sets $d_1 \leftarrow c - d_2$, $r_1 \leftarrow w_1 + rd_1$.

  P sends (d_1, d_2, r_1, r_2) to V.
- $\Sigma_{isBit-4}$: V verifies that $c = d_1 + d_2$, $h^{r_1} = a_1(g^m h^r)^{d_1}$ and $h^{r_2} = a_2(g^{m-1}h^r)^{d_2}$ and aborts if not.

4- Σ_{fin}, Proving Equality of Discrete Logs Or Knowledge of Discrete Log: P demonstrates for $g, h, g_1, h_1, g_2, h_2 \in \mathbb{G}_q$ that *either* it knows the value $\log_{g_1} h_1 = \log_{g_2} h_2$ *or* he knows x such that $h = g^x$ given g. The relation R_{fin} is $((\mathbb{G}_q, q, g_1, g_2, h_1, h_2, g, h), (\alpha, x) \in R_{fin}$ if either (DLE): $h_1 = g_1^\alpha$ and $h_2 = g_2^\alpha$ *or* (DL:) $h = g^x$. The Σ protocol steps are:

- Σ_{fin-1}:
  - If DLE: P chooses $r_1, d_1, w_2 \in \mathbb{Z}_q$ and sets $a_1 \leftarrow g^{r_1}h^{-d_1}$, $a_2 \leftarrow g_1^{w_2}$ and $a_3 \leftarrow g_2^{w_2}$.
  - If DL: P chooses $w_1, r_2, d_2 \in \mathbb{Z}_q$ and sets $a_1 \leftarrow g^{w_1}$, $a_2 \leftarrow g_1^{r_2}h_1^{-d_2}$ and $a_3 \leftarrow g_2^{r_2}h_2^{-d_2}$.

  P sends $(g_1, h_1, g_2, h_2, g, h, a_1, a_2, a_3)$ to V.
- Σ_{fin-2}: V choses $c \in \mathbb{Z}_q$ and sends c to P.
- Σ_{fin-3}:
  - If DLE: P sets $d_2 \leftarrow c - d_1$, $r_2 \leftarrow w_2 + \alpha d_2$.
  - If DL: P sets $d_1 \leftarrow c - d_2$, $r_1 \leftarrow w_1 + xd_1$.

  P sends (d_1, d_2, r_1, r_2) to V.
- Σ_{fin-4}: V verifies that $d_1 + d_2 = c$, $g^{r_1} = a_1 h^{d_1}$, $g_1^{r_2} = a_2 h_1^{d_2}$ and $g_2^{r_2} = a_3 h_2^{d_2}$ and aborts if not.

6 Detailed π^M_{5PM} Specification

We provide here the detailed protocol specification of the malicious model version of $5PM$, π^M_{5PM}. First, we must specify the various zero-knowledge arguments of consistency that are required.

6.1 Arguments of Knowledge of Consistency

We first describe five required interactive arguments which we rely on to prove statements required in the π^M_{5PM} protocol. They are designed for use with the specified threshold ElGamal encryption scheme (Section 2.2). We apply the Σ-ZK-AoK construction outlined in Section 4 to transform the three-move arguments of knowledge outlined in Section 5 to construct the five-move ZK arguments of knowledge π_{DL}, π_{isBit}, π_{eqDL} and π_{fin}, respectively. All arguments are executed between a prover P and a verifier V. π_{DL} is the only ZK-AoK used on its own in π^M_{5PM}; it proves knowledge of a discrete logarithm of a public $h = g^x$. π_{isBIT} is a ZK-AoK that proves that an encryption is either of a 0 or of a 1, π_{eqDL} is a ZK-AoK that proves that two discrete logarithms are equal, and π_{fin} is a ZK-AoK that proves that *either* two discrete logarithms are equal *or* that P knows the discrete logarithm of a public $h = g^x$. The five required interactive arguments are:

$\mathbf{A_{M01}}$, *an AoK of Consistency for Matrix formation 0/1:* In this interactive argument, P sends an $l \times u$ matrix of encryptions, $E(M)$. P demonstrates to V that each column in $E(M)$ contains at most one encryption of a 1 and the rest encryptions of a 0. We assume that P has sent $E(M)$ to V. We denote by A_{M01} the five-move interactive argument where P proves to V using $(l+1)u$ parallel instantiations of π_{isBit} that each entry of $E(M)$ is an encryption of either a 0 or a 1 and that each column-wise product of $E(M)$ is an encryption of either a 0 or a 1. If V accepts the argument A_{M01}, then it accepts that each column is made up of entries that are either 0 or 1 and sum up to 0 or 1; therefore, each column contains encryptions of 0 and at most one 1.

$\mathbf{A_{M1}}$, *an AoK of Consistency for Matrix formation* 0/1-1*:* Similar to the above interactive argument, P sends an $l \times u$ matrix of encryptions, $E(M)$. P demonstrates to V that (unlike the above argument) each down in $E(M)$ contains *e*xactly one encryption of a 1 and the rest encryptions of a 0. To prove that an encryption (x, y) is of a 1, P sends $y' = y/g$ and proves using π_{DL} that $\log_g x = \log_h y'$. V then can see that (x, y) is an encryption of 1 only if $y/y' = g$. We assume that P has sent $E(M)$ to V. We denote by A_{M1} the five-move interactive argument where P sends (x_i, y_i') for each of the row-wise products, (x_i, y_i), of $E(M)$ and then P proves to V using $l \cdot u$ instantiations of π_{isBit} that each entry of $E(M)$ is an encryption of a 0 or a 1 and proves to V using u instantiations of π_{DL} that each row-wise product of $E(M)$ is an encryption of a 1. If V accepts the argument A_{M1}, then it accepts that each row is made up of entries that are either 0 or 1 and sum to 1; therefore, each row contains encryptions of 0 and exactly one 1.

$\mathbf{A_{PD}}$, *an AoK of Consistency for Partial Decryption:* In this interactive argument, P possesses and sends a vector of l encryptions (x_i, y_i) and a vector of their l partial decryptions $(x_i, y_i/x_i^{s_P})$, where s_P is P's private key. P demonstrates to V that he has computed the partial decryptions correctly. We assume that P has already sent the vector of l encryptions and l partial decryptions and that V already knows g^{s_P}. We denote by A_{PD} the five-move interactive argument where P sends, for each i, $x_i^{s_P}$ and proves to V using l parallel instantiations of π_{eqDL} that $\log_g g^{s_P} = \log_{x_i} x_i^{s_P}$.

$\mathbf{A_{Rand}}$, *an AoK of Consistency for Randomization:* In this interactive argument, P possesses and sends a vector of l encryptions (x_i, y_i) and a vector of their randomizations, $(x_i^{r_i}, y_i^{r_i})$, to V and demonstrates knowledge of r_i for each i. P proves using π_{eqDL} that $\log_{x_i} x_i^{r_i} = \log_{y_i} y_i^{r_I}$ for each i. We assume that P has already sent the l encryptions and l randomizations. We denote by A_{Rand} the five-move interactive argument where P proves to V using l parallel instantiations of π_{eqDL} using that each of the l randomizations are formatted correctly.

$\mathbf{A_{FD}}$, *an AoK of Consistency for Final Decryption:* In this interactive argument, P possesses and sends a vector of l encryptions (x_i, y_i), their partial decryptions $(x_i, y_i/x_i^{s_P})$ as well as g^w to V and demonstrates that either P has computed their partial decryptions $(x_i, y_i/x_i^{s_P})$ correctly *or* that he possesses the discrete logarithm w of g^w. We denote by A_{FD} the five-move interactive argument where P proves using l parallel instantiations of π_{fin} that either the l encryptions (x_i, y_i) have been partially decrypted correctly or that P knows the discrete logarithm of g^w.

6.2 π_{5PM}^{M} Protocol Specification

The 8 round protocol for the malicious model, π_{5PM}^{M}, consists of the following six subprotocols:

(1) π_{encr}: initializes an additively homomorphic threshold encryption scheme

(2) $\pi_{S,AV}$: allows Server to construct an encrypted activation vector for Client's encrypted pattern and Server's text.

(3) $\pi_{C,AV}$: allows Client to also construct an encrypted activation vector for Client's pattern and Server's encrypted text.

(4) π_{vec}: allows Client and Server to verify that their activation vectors are equal without revealing them.

(5) π_{rand}: allows Server to send an encryption of its randomized activation vector to Client.

(6) π_{ans}: demonstrates to Client where the pattern matches the text (if at all).

In what follows, we describe π_{5PM}^{M} by specifying in detail the individual subprotocols that are required and specifying for each subprotocol where each round of the subprotocol occurs in the overall (global) rounds of π_{5PM}^{M}. Table 4.6 contains the notation used to describe the subprotocols in Tables 4.7 to 4.12. The required subprotocols utilize the interactive arguments described in Section 6.1 to prove various statements; these arguments are all five-move protocols between a prover (P) and a verifier (V), where, for instance, $A_{M1}^{P,i}$ and

$A_{M1}^{V,j}$ denote the ith and jth messages sent by P and V, respectively, in interactive argument A_{M1}. We denote by $comm(s)$ as shorthand for the commitment of s, which using Pedersen commitments [75] is $g^s h^r = comm(g, h, r, s)$.

We remark that in our construction of ZK arguments of knowledge from Σ protocols, whenever a ZK subprotocol is required, the first two rounds of the five round protocol can be completed in parallel at the very beginning of the overall protocol π_{5PM}^{M}. Such "preprocessing" will not affect security, since these rounds do not involve any Σ protocol-related information from P and as long as V commits to his Σ protocol challenge prior to seeing P's first message of the underlying Σ protocol (see Section 4 for details of the ZK constructions and Section 7 for a proof that preprocessing does not affect security).

p = Pattern of length m	T = Text of length n
AV_i = Activation vector of party i	AV_S^r = Randomized activation vector
AV_i' = Blinded activation vector	M_{CDV}= Matrix encoding of p in terms of Σ
sk_C = Client's secret-key	sk_S = Server's secret-key
p_t = Pattern match threshold	$\langle , \rangle$ = Inner product over $\mathbb{G}_q$
$\pi_{rel}^{i,j}(x)$ = Party i's jth message of π_{rel} for x	$E()$ = Additively homomorphic encryption
$D_i()$ = Partial decryption by party i	h = Threshold public-key
s^* = Simulator trapdoor	M_T = Matrix encoding of T in terms of Σ
$A_{rel}^{i,j}(x)$ =Player i's jth message of A_{rel} for x	

Table 4.6: Notation used in subprotocols in Tables 4.7 through 4.12

Global Round	**Client**	**Messages**	**Server**
1	$s_C, s^* \in \mathbb{Z}_q$, $h_1 \leftarrow g^{s_C}$, $h^* \leftarrow g^{s^*}$	$\xrightarrow{h_1, h^*, \pi_{DL}^{P,1}(h_1), \pi_{DL}^{P,1}(h^*)}$	
2		$\xleftarrow{h_2, \pi_{DL}^{P,1}(h_2), \pi_{DL}^{V,1}(h_1), \pi_{DL}^{V,1}(h^*)}$	$s_S \in \mathbb{Z}_q$, $h_2 \leftarrow g^{s_S}$, $h = h_1 h_2$
3	$h = h_1 h_2$	$\xrightarrow{\pi_{DL}^{P,2}(h_1), \pi_{DL}^{P,2}(h^*), \pi_{DL}^{V,1}(h_2)}$	
4		$\xleftarrow{\pi_{DL}^{P,2}(h_2), \pi_{DL}^{V,2}(h_1), \pi_{DL}^{V,2}(h^*)}$	
5		$\xrightarrow{\pi_{DL}^{P,3}(h_1), \pi_{DL}^{P,3}(h^*), \pi_{DL}^{V,2}(h_2)}$	
6		$\xleftarrow{\pi_{DL}^{P,3}(h_2)}$	

Table 4.7: Subprotocol π_{encr}

$\pi_{\mathbf{encr}}$, shown in Table 4.7, is a two party protocol for Client and Server that initializes a threshold ElGamal encryption scheme. For simplicity we assume that Client and Server

have already agreed on $\mathbb{G}_q$ and $g \in \mathbb{G}_q$. Client input: $\mathbb{G}_q, g \in G_q$. Server input is: $\mathbb{G}_q, g \in G_q$. Output to Client: $h = g^{s_C} g^{s_S}$. Output to Server: $h = g^{s_C} g^{s_S}$, $h^* = g^{s^*}$. This subprotocol begins at the first global round and ends at global round 6.

- At global round 1 Client chooses $s_C, s^* \in \mathbb{Z}_q$ and sets $h_1 \leftarrow g^{s_C}$, $h^* \leftarrow g^{s^*}$. Client sends h_1, h^* to Server. Client sends the Server two parallel instantiations of $\pi_{DL}^{P,1}$ proving knowledge of the discrete logs of h_1 and h^* (e.g,of s_C and s^*). The last message of Client's instantiations of π_{DL} is exchanged at global round 5.
- At global round 2 Server chooses $s_S \in \mathbb{Z}_q$ and sets $h_2 \leftarrow g^{s_S}$. Server sends h_2 to Client as well as $\pi_{DL}^{P,1}$ proving knowledge of the discrete logarithm of h_2 (e.g,of s_S). The last message of Server's π_{DL} is sent at global round 6. Both parties set the public-key to be $h = h_1 h_2$.

$\pi_{\mathbf{C,AV}}$, shown in Table 4.8, is a two party protocol for Client and Server which outputs to Client an encrypted activation vector corresponding to matching Client's p against Server's T. Client input: pattern p, threshold p_t. Server input: text T and M_T, which is the $|\Sigma| \times n$ matrix encoding T in terms of Σ (see Section 3.1). Output to Server: none. Output to Client: $E(AV_C)$, an encrypted activation vector corresponding to matching p against T. This subprotocol starts at global round 2 and ends at global round 6.

- At global round 2 Server sends $E(M_T)$ to Client. Server also sends, for $E(M_T)$, $A_{M1}^{P,1}$ to prove that $E(M_T)$ is formatted correctly. The last message of Server's A_{M1} is exchanged at global round 6.
- During global round 3, Client computes $E(AV_C)$ from $E(M_T)$, M_{CDV} for p, and p_t by first computing $M_{CDV} \cdot E(M_T)$. This can be performed by recognizing that one can obtain an encryption of the inner product over $\mathbb{Z}_q$ of an unencrypted vector $(x_1, ..., x_m)$ with an encrypted vector $(E(y_1), ..., E(y_m))$ by computing $\Pi E(y_i)^{x_i} = E(\sum x_i y_i)$ (see Section 2.3.2).

Global Round	Client	Messages	Server
2		$\xleftarrow{E(M_T), A_{M1}^{P,1}(E(M_T))}$	$M_T \leftarrow T$, $E(M_T) \leftarrow M_T$
3	$E(AV_C) \leftarrow M_{CDV}, p_t, E(M_T)$	$\xrightarrow{A_{M1}^{V,1}(E(M_T))}$	
4		$\xleftarrow{A_{M1}^{P,2}(E(M_T))}$	
5		$\xrightarrow{A_{M1}^{V,2}(E(M_T))}$	
6		$\xleftarrow{A_{M1}^{P,3}(E(M_T))}$	

Table 4.8: Subprotocol $\pi_{C,AV}$

Global Round	Client	Messages	Server
1		$\xrightarrow{A_{M01}^{P,1}(E(M_{CDV}))}$	
2		$\xleftarrow{A_{M01}^{V,1}(E(M_{CDV}))}$	
3	$E(M_{CDV}) \leftarrow CDV$, $E(p_t) \leftarrow p_t$	$\xrightarrow{E(M_{CDV}), A_{M01}^{P,2}(E(M_{CDV}))}$	
4		$\xleftarrow{A_{M01}^{V,2}(E(M_{CDV}))}$	$E(AV_S) \leftarrow T, E(M_{CDV}), E(p_t)$
5		$\xrightarrow{A_{M01}^{P,3}(E(M_{CDV}))}$	

Table 4.9: Subprotocol $\pi_{S,AV}$

$\pi_{\mathbf{S,AV}}$, shown in Table 4.9, is a two party protocol for Client and Server which outputs to Server an encrypted activation vector corresponding to matching Client's p against Server's T. Client input: pattern p, M_{CDV} for p, and p_t, the matching threshold. Server input: T. Output to Client: none. Output to the Server: $E(AV_S)$, an encrypted activation vector corresponding to matching p against T. This subprotocol starts at global round 3 and ends at global round 5, with ZK preprocessing occurring during global rounds 1 and 2.

- At global round 3 Client sends $E(M_{CDV})$ and $E(p_t)$ to Server. Client also sends $A_{M01}^{P,2}$ to prove that $E(M_{CDV})$ is formatted correctly, where $A_{M01}^{P,1}$ and $A_{M01}^{V,1}$ occur during global rounds 1 and 2, respectively. The last message of Client's A_{M01} is sent at global round 5.
- During global round 4, using $E(M_{CDV})$, T and $E(p_t)$, Server computes $E(AV_S)$ (see step 5 in Section 3.2.2).

$\pi_{\mathbf{vec}}$, shown in Table 4.10, is a two party protocol for Client and Server that outputs to each party whether their respective encrypted activation vectors are equal (without revealing their values). Client input: $E(AV_C)$, $E(AV'_C)$ which is constructed from $E(AV_C)$ by multiplying each element by an encryption of 0. Server input: $E(AV_S)$, $E(AV'_S)$ which is constructed from $E(AV_S)$ by multiplying each element by an encryption of 0. Output to both Client and Server: whether $AV_C = AV_S$ or not. This subprotocol begins at global round 3 and ends at global round 8, with ZK preprocessing occurring during global rounds 1, 2 and 3. $\langle \cdot, \cdot \rangle$ denotes the inner product over $\mathbb{Z}_q$.

- At global round 3 Client chooses $r_1 \in \mathbb{Z}_q^{n-m+1}$ and $r'_1 \in \mathbb{Z}_q$. Client computes $E(AV'_C)$ by multiplying each element of AV_C with an encryption of 0 thus blinding the ciphertext. Client generates $comm(E(AV'_C))$, $comm(r_1)$, and $comm(E(r'_1))$ and sends them to Server.
- At global round 4 Server chooses $r_2 \in \mathbb{Z}_q^{n-m+1}$ and $r'_2 \in \mathbb{Z}_q$. Server computes $E(AV'_S)$ by multiplying each element of AV_S by an encryption of 0 thus blinding the ciphertext. Server sends $r_2, E(r'_2), E(AV'_S)$ to Client.
- At global round 5 Client sets $r = r_1 + r_2$ and $E(r') = E(r'_1 + r'_2)$. Client computes $z_1 = E(\langle AV_C, r \rangle + r')$ and $z_2 = E(\langle AV_S, r \rangle + r')$. Client opens the commitments of $E(AV'_C)$, r_1, and $E(r'_1)$ to Server. Client sends z_1 and z_2 to Server. Client computes partial decryptions $D_C(z_1)$, $D_C(z_2)$ and sends $D_C(z_1)$, $D_C(z_2)$ to Server as well as $A_{PD}^{P,2}$ to prove that the partial decryptions $D_C(z_1)$, $D_C(z_2)$ are computed correctly, where messages $A_{PD}^{P,1}$ and $A_{PD}^{V,1}$ are sent during global rounds 1 and 2, respectively. Execution of Client's A_{PD} continues until global round 7.
- At global round 6 Server obtains z_1, z_2 from $D_C(z_1)$ and $D_C(z_2)$. Server aborts if $z_1 \neq z_2$. Server sets $r = r_1 + r_2$ and $E(r') = E(r'_1 + r'_2)$. Server computes $z_1 = E(\langle AV_C, r \rangle + r')$ and $z_2 = E(\langle AV_S, r \rangle + r')$. Server computes partial decryptions $D_S(z_1)$, $D_S(z_2)$ and sends $D_S(z_1)$ and $D_S(z_2)$ to Client as well as $A_{PD}^{P,2}$ to prove that

the partial decryptions $D_S(z_1)$, $D_S(z_2)$ are computed correctly, where $A_{PD}^{P,1}$ and $A_{PD}^{V,1}$ are sent during global rounds 2 and 3, respectively. Execution of Server's A_{PD} continues until global round 8.

- At global round 7 Client obtains z_1, z_2 from $D_S(z_1)$ and $D_S(z_2)$. Client aborts if $z_1 \neq z_2$.

Since r, r' are uniform, the probability that z_1 and z_2 have equal decryptions for unequal vectors is negligible ($\frac{1}{q}$).

$\pi_{\mathbf{rand}}$, shown in Table 4.11, is a two party protocol for Client and Server that outputs to Client an encrypted vector $E(AV_S^r)$ that contains randomizations of the values in non-matching (non-zero) positions in $E(AV_S')$. Client input: nothing. Server input: $E(AV_S')$. Output to Client: $E(AV_S^r)$. Output to Server: Nothing. Client is assumed to already know $E(AV_S')$. This subprotocol starts at global round 6 and ends at global round 8, with ZK preprocessing occurring during global rounds 2 and 3.

- At global round 6 Server computes $E(AV_S^r)$ from $E(AV_S')$ by exponentiating each encryption in $E(AV_S')$ by a random value. Server sends $E(AV_S^r)$ to Client and sends $A_{rand}^{P,2}$ to prove that $E(AV_S^r)$ was obtained correctly from $E(AV_S')$, where $A_{rand}^{P,1}$ and $A_{rand}^{V,1}$ are sent during global rounds 2 and 3, respectively . The last message of Server's A_{rand} is exchanged at global round 8.

$\pi_{\mathbf{ans}}$, shown in Table 4.12, is a two party protocol for Client and Server that outputs to Client the randomization, AV_S^r, of Server's activation vector AV_S. Client input: none. Server input: $E(AV_S^r)$. Output to Server: none. Output for Client: AV_S^r. Client is assumed to already know $E(AV_S^r)$. This subprotocol starts at global round 6 and ends at global round 8, with ZK preprocessing occurring during global rounds 2 and 3.

- At global round 6 Server sends $comm(D_S(E(AV_S^r)))$. Server also sends the message $comm(A_{FD}^{P,2})$, where A_{FD} is the argument to prove that either $D_S(E(AV_S^r))$ was

Global Round	Client	Messages	Server
1		$\xrightarrow{A_{PD}^{P,1}(D_C(z_1)),A_{PD}^{P,1}(D_C(z_2))}$	
2		$\xleftarrow{A_{PD}^{P,1}(D_S(z_1)),A_{PD}^{P,1}(D_S(z_2)),A_{PD}^{V,1}(D_C(z_1)),A_{PD}^{V,1}(D_C(z_2))}$	
3	$r_1 \in \mathbb{Z}_q^{n-m+1}$, $r_1' \in \mathbb{Z}_q$, $E(r_1') \leftarrow r_1'$, $E(AV_C') \leftarrow E(AV_C)$	$\xrightarrow[A_{PD}^{V,1}(D_S(z_1)),A_{PD}^{V,1}(D_S(z_2))]{comm(E(AV_C')),comm(r_1),comm(E(r_1'))}$	
4		$\xleftarrow{E(AV_S'),r_2,E(r_2')}$	$r_2 \in \mathbb{Z}_q^{n-m+1}$, $r_2' \in \mathbb{Z}_q$, $E(r_2') \leftarrow r_2'$, $E(AV_S') \leftarrow E(AV_S)$
5	$r \leftarrow r_1 + r_2$, $E(r') \leftarrow E(r_1' + r_2')$ $z_1 \leftarrow E(\langle AV_C, r\rangle + r')$, $z_2 \leftarrow E(\langle AV_S, r\rangle + r')$ $D_C(z_1) \leftarrow z_1$, $D_C(z_2) \leftarrow z_2$	$\xrightarrow[z_1,z_2,D_C(z_1),D_C(z_2),A_{PD}^{P,2}(D_C(z_1)),A_{PD}^{P,2}(D_C(z_2))]{decom(E(AV_C')),decom(r_1),decom(E(r_1'))}$	
6		$\xleftarrow[A_{PD}^{V,2}(D_C(z_1)),A_{PD}^{V,2}(D_C(z_2))]{z_1,z_2,D_S(z_1),D_S(z_2),A_{PD}^{P,2}(D_S(z_1)),A_{PD}^{P,2}(D_S(z_2))}$	$D_S(D_C(z_1)) \stackrel{?}{=} D_S(D_C(z_2))$, $r \leftarrow r_1 + r_2$ $z_1 \leftarrow E(\langle AV_C, r\rangle + r')$, $z_2 \leftarrow E(\langle AV_S, r\rangle + r')$ $D_S(z_1) \leftarrow z_1$, $D_S(z_2) \leftarrow z_2$
7	$D_C(D_S(z_1)) \stackrel{?}{=} D_C(D_S(z_2))$	$\xrightarrow{A_{PD}^{P,3}(D_C(z_1)),A_{PD}^{P,3}(D_C(z_2)),A_{PD}^{V,2}(D_S(z_1)),A_{PD}^{V,2}(D_S(z_2))}$	
8		$\xleftarrow{A_{PD}^{P,3}(D_S(z_1)),A_{PD}^{P,3}(D_S(z_2))}$	

Table 4.10: Subprotocol π_{vec}

Global Round	Client	Messages	Server
2		$\xleftarrow{A^{P,1}_{rand}(E(AV^r_S))}$	
3		$\xrightarrow{A^{V,1}_{rand}(E(AV^r_S))}$	
6		$\xleftarrow{E(AV^r_S),A^{P,2}_{rand}(E(AV^r_S))}$	$E(AV^r_S) \leftarrow E(AV'_S)$
7		$\xrightarrow{A^{V,2}_{rand}(E(AV^r_S))}$	
8		$\xleftarrow{A^{P,3}_{rand}(E(AV^r_S))}$	

Table 4.11: Subprotocol π_{rand}

Global Round	Client	Messages	Server
2		$\xleftarrow{A^{P,1}_{FD}(D_S(E(AV^r_S)))}$	
3		$\xrightarrow{A^{V,1}_{FD}(D_S(E(AV^r_S)))}$	
6		$\xleftarrow[comm(A^{P,2}_{FD}(D_S(E(AV^r_S))))]{comm(D_S(E(AV^r_S))),}$	$D_S(E(AV^r_S)) \leftarrow E(AV^r_S)$
7		$\xrightarrow{A^{V,2}_{FD}(D_S(E(AV^r_S)))}$	
8	$AV^r_S \leftarrow D_C(D_S(E(AV^r_S)))$	$\xleftarrow[decom(A^{P,2}_{FD}(D_S(E(AV^r_S))))]{decom(D_S(E(AV^r_S))),A^{P,3}_{FD}(D_S(E(AV^r_S)))}$	

Table 4.12: Subprotocol π_{ans}

obtained correctly or that Server knows s^* (for h^* sent by Client in the first global round during π_{encr}), where $A^{P,1}_{FD}$ and $A^{V,1}_{FD}$ are sent during global rounds 2 and 3, respectively.

- At global round 7 Client sends $A^{V,2}_{FD}$ to Server (our AoKs are public coin so $A^{V,2}_{FD}$ is not determined by$A^{P,2}_{FD}$).
- At global round 8 Server opens commitments of $A^{P,2}_{FD}$, $D_S(E(AV^r_S))$ to Client. Server sends $A^{P,3}_{FD}$ to Client.
- Client aborts if it does not accept the argument A_{FD}.

7 Security Analysis

Here we define and prove security for π^H_{5PM} and π^M_{5PM}.

7.1 Adversarial Model

Let $\mathcal{F}_f$ be a functionality for two parties P_1 and P_2 where P_1 inputs x_1, P_2 inputs x_2, P_1 obtains $f(x_1, x_2)$ and P_2 obtains nothing. A protocol π_f that securely computes f can be defined as an interactive two-party protocol. We refer the reader to [37, 65] for further discussion of the definitions given here.

Execution of π_f^A in the real world: Let A denote the adversary model: H for honest but curious and M for malicious (static corruption). Both parties are assumed to be probabilistic (expected) polynomial time (PPT) algorithms. In particular, we denote by $\bar{P} = (P_1, P_2)$ a pair of PPT algorithms that execute π_f^A where at most one of the parties is adversarial (or corrupted). Such a pair $\bar{P}$ is called *admissible*. Let r_i be P_i's internal randomness, x_i be P_i's private input and y_i be P_i's auxiliary input. Let $P_1 \leftrightarrow P_2$ be the transcript of the public interactions between parties P_1 and P_2. Note that parties can be defined via their next message functions; see, for example, [60]. In the honest-but-curious (HBC) adversary model, before the protocol begins, the adversary can choose to corrupt one party for the duration of the entire protocol; that party may not deviate from the protocol specification of π_f^A. In the (static corruption) malicious adversary model, before the protocol begins, the adversary can choose to corrupt one party for the duration of the entire protocol; this party may deviate arbitrarily from the protocol specification of π_f^M. In particular, the corrupted party may choose to abort and to not complete the protocol at all. Denote by x_c, y_c, r_c the internal input, randomness and auxiliary input of a corrupted party, respectively (if there is no corrupted party then this sequence is the empty sequence). Denote by $\bar{x} = (x_1, x_2)$, $\bar{y} = (y_1, y_2)$, $\bar{r} = (r_1, r_2)$. We denote by $REAL_{\bar{P}}^{\pi_f^A}(\bar{x}, \bar{y}, \bar{r})$ as $(x_c, r_c, y_c, P_1 \leftrightarrow P_2)$.

Execution of π_f^A in the ideal world: In the ideal world setting, an admissible pair of two PPT parties $\bar{P}' = (P_1', P_2')$ interact with a trusted ideal functionality to jointly compute the function f specified by π_f^A. At any point, a dishonest party may send abort rather than

send what it is supposed to.

Ideal functionality $\mathcal{F}_f^H$ for the honest-but-curious (HBC) model: P_2' sends its input, x_2, to the ideal functionality. The ideal functionality sends the size of x_2, $|x_2|$, to P_1'. P_1' sends its input, x_1, to the ideal functionality. The ideal functionality sends the size of x_1, $|x_1|$, to P_2'. The ideal functionality provides the correct value of $f(x_1, x_2)$ to P_1'. An honest party must output what was output to it by the ideal functionality (in particular, if P_2' is honest, it outputs nothing); a dishonest party may output what it wishes. We denote by $IDEAL_{\bar{P}'}^{\pi_f}(\bar{x}, \bar{y}, \bar{r})$ as the pair of public outputs of P_1' and P_2'.

Ideal functionality $\mathcal{F}_f^M$ for the malicious static corruption model: P_2' sends its input, x_2, to the ideal functionality. The ideal functionality sends the size of x_2, $|x_2|$, to P_1'. P_1' sends its input, x_1, to the ideal functionality. The ideal functionality sends the size of x_1, $|x_1|$, to P_2'. P_2' sends "proceed" to the ideal functionality (note that an adversarial party can choose to abort this procedure at any time). The ideal functionality provides the correct value of $f(x_1, x_2)$ to P_1'. An honest party must output what was output to it by the ideal functionality (in particular, if P_2' is honest, it outputs nothing); a dishonest party may output what it wishes. We denote by $IDEAL_{\bar{P}'}^{\pi_f^A}(\bar{x}, \bar{y}, \bar{r})$ as the pair of public outputs of P_1' and P_2'.

Using the standard ideal/real formulation, we obtain the following definitions of security.

Definition 7.1. *π_f^H securely realizes $\mathcal{F}_f^H$ in the honest-but-curious (static corruption) model if for every admissible PPT pair $\bar{P}$ in the real world there exists an admissible PPT pair $\bar{P}'$ in the ideal world such that $REAL_{\bar{P}}^{\pi_f^H}(\bar{x}, \bar{y}, \bar{r})$ and $IDEAL_{\bar{P}'}^{\pi_f^H}(\bar{x}, \bar{y}, \bar{r})$ are computationally indistinguishable, where the distributions are over the uniformly random, independent choices of private input $\bar{x}$, randomness $\bar{r}$ and auxiliary input $\bar{y}$.*

Definition 7.2. *π_f^M securely realizes $\mathcal{F}_f^M$ in the malicious (static corruption) model if for every admissible PPT pair $\bar{P}$ in the real world there exists an admissible PPT pair $\bar{P}'$ in the ideal world such that $REAL_{\bar{P}}^{\pi_f^M}(\bar{x}, \bar{y}, \bar{r})$ and $IDEAL_{\bar{P}'}^{\pi_f^M}(\bar{x}, \bar{y}, \bar{r})$ are computationally indistinguishable, where the distributions are over the uniformly random, independent choices of private input $\bar{x}$, randomness $\bar{r}$ and auxiliary input $\bar{y}$.*

Simulation in the ideal world: In practice, for security to hold in the HBC (respectively

malicious) adversary model, for each corrupted party P_i' in the real world, there exists a PPT simulator $\mathcal{S}_{P_i'}$ in the ideal world with oracle access to P_i' such that $REAL_{\bar{P}}^{\pi_f}(\bar{x}, \bar{y}, \bar{r})$ and $IDEAL_{\bar{P}'}^{\pi_f}(\bar{x}, \bar{y}, \bar{r})$ are computationally indistinguishable, where the other (honest) party of $\bar{P}$ and $\bar{P}'$ acts honestly. In particular, $\mathcal{S}_{P_i'}$ when it is allowed to output per the ideal world specification, will attempt to output a transcript that is computationally indistinguishable from P_i''s view of the transcript in the real world- without knowing the private input of the real world honest party.

7.2 Simulator Constructions and Security for π_{5PM}^H

We provide, for each admissible pair in the real world, an admissible pair in the ideal world such that $REAL_{\bar{P}}^{\pi_{5PM}^H}(\bar{x}, \bar{y}, \bar{r})$ and $IDEAL_{\bar{P}'}^{\pi_{5PM}^H}(\bar{x}, \bar{y}, \bar{r})$ are computationally indistinguishable.

7.2.1 Simulator specification for the adversarial Server, $\mathcal{S}_S$, for π_{5PM}^H

Without loss of generality, consider the case where the matching locations are not hidden. Consider the admissible pair $\bar{P}$ = (Client,Server) in the real world. We construct $\mathcal{S}_S$ for an admissible pair $\bar{P}'$ =(Client, $\mathcal{S}_S$) in the ideal world (where Client behaves honestly in both cases) such that $REAL_{\bar{P}}^{\pi_{5PM}^H}(\bar{x}, \bar{y}, \bar{r})$ and $IDEAL_{\bar{P}'}^{\pi_{5PM}^H}(\bar{x}, \bar{y}, \bar{r})$ are computationally indistinguishable. Note that $\mathcal{S}_S$ has oracle access to real world Server.

Initial Interactions with the Ideal Functionality: We assume that the encryption scheme (Key, E, D) is fixed. Server upon oracle call to $\mathcal{S}_S$, sends its text T to $\mathcal{S}_S$, who forwards it on to the ideal functionality. Once the ideal functionality reveals the length that the pattern should be, $\mathcal{S}_S$ sets the pattern $p_t = p^*$ to be all 1s (we state this without loss of generality; in the case that $1 \notin \Sigma$, an arbitrary $a \in \Sigma$ is chosen and p_t is set to be all a's) of the right length (any arbitrary vector can be used here, but without loss of generality we use all 1s). Outputs below are what $\mathcal{S}_S$ outputs at the output phase of the ideal world

specification.

(1) $\mathcal{S}_S$ computes $(sk_C, pk_C) \leftarrow Key(1^k)$. Using p_t, $\mathcal{S}_S$ constructs $M_{CDV} \leftarrow Gen_{CDV}(p_t)$ and encrypts it to obtain $E(M_{CDV})$. $\mathcal{S}_S$ sends $E(M_{CDV})$ and pk_C to Server. In addition, $\mathcal{S}_S$ sends $E(-m)$, where $m = |p_t|$ (or $E(-m+l)$ in the single character wildcard or substring cases, where l is the threshold).

(2) Server considers each character in the text T_i $(1 \leq i \leq n)$ and retrieves the corresponding row of $E(M_{CDV})$. This is the step that corresponds to multiplying $M_T \cdot M_{CDV}$ in Section 3.1. The resulting vectors are multiplied with the encrypted activation vector $E(AV)$ element by element in positions $i, ..., i+m-1$ of the AV. This is the step corresponding to transforming $M_{T(M_{CDV})}$ to $\overline{M}_{T(M_{CDV})}$ and then performing the multiplication $[1...1]^n \cdot \overline{M}_{T(M_{CDV})}$ to get the final AV. Server then multiplies $E(-m)$ $(= E(m)^{-1})$ to each of the entries in AV and exponentiates each entry by a randomly chosen number to blind entries in $E(AV)$. We call the randomized activation vector $E(AV_S^r)$. Server sends $E(AV_S^r)$ to $\mathcal{S}_S$.

(3) $\mathcal{S}_S$ aborts the ideal functionality before Client outputs its pattern matching results obtained from the ideal functionality.

7.2.2 Simulator specification for the adversarial Client, $\mathcal{S}_C$, for π_{5PM}^H

Consider the admissible pair $\bar{P}$ = (Client,Server) in the real world. We construct $\mathcal{S}_S$ for an admissible pair $\bar{P}'$ =($\mathcal{S}_C$, Server) in the ideal world (where Server behaves honestly in both cases) such that $REAL_{\bar{P}}^{\pi_{5PM}^H}(\bar{x}, \bar{y}, \bar{r})$ and $IDEAL_{\bar{P}'}^{\pi_{5PM}^H}(\bar{x}, \bar{y}, \bar{r})$ are computationally indistinguishable. Note that $\mathcal{S}_S$ has oracle access to real world Client.

Initial Interactions with the Ideal Functionality: We assume that the encryption scheme (Key, E, D) is fixed. Upon oracle call to Client, Client sends pattern p to $\mathcal{S}_C$, who forwards it on to the ideal functionality. Once the ideal functionality reveals the length that the text should be, $\mathcal{S}_C$ sets the text $T = T^*$ to be all 1s (we note that, as above, if $1 \notin \Sigma$,

T^* can set to be all a's for any $a \in \Sigma$) of the right length (any arbitrary vector can be used here, but without loss of generality we use all 1s).

(1) Client computes $(sk_C, pk_C) \leftarrow Key(1^k)$. Using, p, Client constructs $M_{CDV} \leftarrow Gen_{CDV}(p)$ and encrypts it to obtain $E(M_{CDV})$. Client sends $E(M_{CDV})$ and pk_C to $\mathcal{S}_C$. In addition, Client sends $E(-m)$, where $m = |p|$ (or $E(-m+l)$ in the single character wildcard or substring cases, where l is the threshold).

(2) The ideal functionality sends $\mathcal{S}_C$ the pattern matching results with the correct (e.g,real world) pattern and text- e.g,all positions where the pattern should match. $\mathcal{S}_C$ constructs a new vector of encryptions, $E(AV_{IF})$, by using Client's public-key to encrypt an $(m+n-1)$-length vector with 0s where the pattern should match and random elements elsewhere. $\mathcal{S}_C$ sends $E(AV_{IF})$ to Client.

7.2.3 Security of π^H_{5PM}

We prove that π^H_{5PM} securely realizes $\mathcal{F}^H_{5PM}$ in the honest-but-curious (static corruption) model by demonstrating the computational indistinguishability of $REAL^{\pi^M_{5PM}}_{\bar{P}}(\bar{x}, \bar{y}, \bar{r})$ and $IDEAL^{\pi^M_{5PM}}_{\bar{P}'}(\bar{x}, \bar{y}, \bar{r})$.

Theorem 7.3. *Given an additively homomorphic semantically secure encryption scheme over prime-order cyclic groups (Key, E, D), π^H_{5PM} securely realizes $\mathcal{F}^H_{5PM}$ in the honest-but-curious (static corruption) model.*

Proof of Theorem 7.3. We demonstrate that the two simulators $\mathcal{S}_S$ and $\mathcal{S}_C$ output transcripts such that $REAL^{\pi^H_{5PM}}_{\bar{P}}(\bar{x}, \bar{y}, \bar{r})$ and $IDEAL^{\pi^H_{5PM}}_{\bar{P}'}(\bar{x}, \bar{y}, \bar{r})$ are computationally indistinguishable.

Case 1: Adversarial Server. The transcript in the real world is the transcript $(pk_C, T, E(M_{CDV})_p, E(-m))$, where by $E(M_{CDV})_p$ we mean the encrypted M_{CDV} matrix constructed from p. Note then that $E(M_{CDV})_p$ is the encrypted M_{CDV} matrix constructed

from p_t. The view in the ideal world is $(pk_C, T, E(M_{CDV})_{p_t}, E(m))$, where p_t is a string of m 1s. Suppose a distinguisher D can distinguish the distributions of the real and ideal transcripts with non-negligible probability. In particular, D distinguishes $(E(M_{CDV})_p)$ from $(E(M_{CDV})_{p_t})$ with non-negligible probability given pk_C and T. In particular, let a distinguisher D have as input pk_C and T. Define the distribution

$$X_i = (E(M_{CDV})_{p,1}, ..., E(M_{CDV})_{p,k}, E(M_{CDV})_{p_t,k+1}, ..., E(M_{CDV})_{p_t,m|\Sigma|}),$$

where $E(M_{CDV})_{p_t,i}$ is the ith encrypted element in $E(M_{CDV})$ constructed from the pattern p_t (where the matrix here is thought of as a string). By a hybrid argument, for some $0 \leq k \leq m|\Sigma|$, given pk_C and T, D can distinguish X_k from X_{k+1} in polynomial time with non-negligible probability. But this violates the semantic security of E since D only has the public-key of E and T, which is independent of p/p_t.

More precisely, let the hybrid experiment H_i, $0 \leq i \leq m|\Sigma|$ be such that the first i encryptions sent by the simulator come from $M_{CDV_{p_t}}$ while the remaining encryptions come from M_{CDV_p}. Note that H_0 is the distribution of the real Client while $H_{m|\Sigma|}$ is the distribution of the simulator in the ideal world. Suppose that H_i is computationally indistinguishable from H_{i+1} for some i. The we reduce to the semantic security of ElGamal encryption. Namely, we consider a player P_{enc} who encrypts and another R who receives. P_{enc} is given as auxiliary inputs the p and T such that H_i and H_{i+1} are computationally distinguishable with non-negligible probability via a distinguisher D. P_{enc} sends R the public key, which R uses to internally execute H_i and H_{i+1} using p_t, p and T. At the $i+1$ encryption, R sends P_{enc} the two plain texts from $M_{CDV_{p_t}}$ and M_{CDV_p} used for the $i+1$ encryption in the respective hybrids; P_{enc} sends back the encryptions in a randomized order. P_{enc} continues the internal execution. Note that these distributions are identical to H_i and H_{i+1}. If H_i and H_{i+1} are computationally distinguishable with non negligible probability using D, then R, using D, can distinguish the two encryptions with non negligible probability, which implies that ElGamal encryption is not semantically secure, a contradiction.

Case 2: Adversarial Client.

The transcript in the real world is the transcript (sk_C, p, m, AV_S^r), while the view in the ideal world is the transcript (sk_C, p, m, AV_{IF}^r). Indistinguishability here is statistical- namely, by construction, AV_S^r and AV_{IF}^r have zeros in the same places; their non-zero locations contain elements chosen uniformly (and independently) at random since the group has prime order. This implies statistical indistinguishability of the transcripts (in particular, in this case, security does not rely on the semantic security of the encryption scheme at all other than to hide the matching result from an eavesdropper). □

7.3 Simulator Constructions and Security for π_{5PM}^M

We provide, for each admissible pair in the real world, an admissible pair in the ideal world such that $REAL_{\bar{P}}^{\pi_{5PM}^M}(\bar{x}, \bar{y}, \bar{r})$ and $IDEAL_{\bar{P}'}^{\pi_{5PM}^M}(\bar{x}, \bar{y}, \bar{r})$ are computationally indistinguishable.

In what follows, we assume that if a message is incorrectly formatted, the simulator will simply abort; since this might occur at every individual interaction, we omit it for simplicity.

7.3.1 Simulator specification for an adversarial Server, $\mathcal{S}_S$, for π_{5PM}^M

We describe how an adversarial Server interacts with $\mathcal{S}_S$ for π_{5PM}^M. In particular, consider the admissible pair $\bar{P}$ = (Client,Server) in the real world. We construct $\mathcal{S}_S$ for an admissible pair $\bar{P}'$ =(Client, $\mathcal{S}_S$) in the ideal world (where Client behaves honestly in both cases) such that $REAL_{\bar{P}}^{\pi_{5PM}^M}(\bar{x}, \bar{y}, \bar{r})$ and $IDEAL_{\bar{P}'}^{\pi_{5PM}^M}(\bar{x}, \bar{y}, \bar{r})$ are computationally indistinguishable. Note that $\mathcal{S}_S$ will have oracle access to Server.

We list the subprotocols of π_{5PM}^M and describe how an adversarial Server interacts with $\mathcal{S}_S$. Recall that π_{encr} is a protocol to instantiate a threshold ElGamal encryption scheme, $\pi_{C,AV}$ is for Client to compute an activation vector, $\pi_{S,AV}$ is for Server to compute an activation vector, π_{vec} is for the parties to determine that their activation vectors are equal, π_{rand} is for Server to send Client an encrypted vector that only reveals matching locations

upon decryption, and π_{ans} is for Server to partially decrypt that encrypted vector. We refer the reader to Section 3.3.3 and Section 6.2 for protocol details.

Initial Interactions with the Ideal Functionality: We may view the following interaction with the ideal functionality as occurring during the execution of $\pi_{C,AV}$ between $\mathcal{S}_S$ and Server. Upon oracle call, Server reveals the length of its text to $\mathcal{S}_S$ (it will do so in the protocol when it sends $E(M_T)$ during $\pi_{C,AV}$, since this matrix is of size $|T| \times |\Sigma|$). For the purpose of ideal functionality interaction, $\mathcal{S}_S$ will set the text T' to be all 1s (here are throughout this specification, this is without loss of generality assuming $1 \in \Sigma$; else another character may be used) with length the same as Server's text and send T' and send it to the ideal functionality. The ideal functionality will return the length of Client's pattern. $\mathcal{S}_S$ sets the pattern $p_t = p^*$ to be all 1s of the length of the pattern (any arbitrary vector can be used here, but without loss of generality we use all 1s). $\mathcal{S}_S$ will then abort the ideal functionality so that (ideal world) Client does not output anything in the ideal functionality. The following is what $\mathcal{S}_S$ will output for $IDEAL^{\pi^M_{5PM}}$ (explaining how $\mathcal{S}_S$ makes oracle calls to Server).

$\pi_{\mathbf{encr}}$:

This subprotocol begins at the first global round and ends at global round 6.

(1) $\mathcal{S}_S$ chooses $s_C, s^* \in \mathbb{Z}_q$ and sets $h_1 \leftarrow g^{s_C}$, $h^* \leftarrow g^{s^*}$. $\mathcal{S}_S$ sends h_1, h^* to Server. $\mathcal{S}_S$ sends the Server two parallel instantiations of $\pi^{P,1}_{DL}$ for h_1 and h^* with witnesses s_C and s^*. This continues through global round 5.

(2) Server sends h_2 (normally equal to g^{s_S}) to $\mathcal{S}_S$ as well as $\pi^{P,1}_{DL}$ for h_2 with witness s_S. Protocol π_{DL} continues through global round 6. Both parties set $h = h_1 h_2$ as the public-key (with secret-key $s = s_C + s_S$).

$\pi_{\mathbf{C,AV}}$:

This subprotocol starts at global round 2 and ends at global round 6.

(1) Server sends $E(M_T)$ to $\mathcal{S}_S$. Server also sends, for $E(M_T)$, $A_{M1}^{P,1}$ to demonstrate that $E(M_T)$ is formatted correctly . A_{M1} continues through global round 6.

(2) During global round 3, $\mathcal{S}_S$ computes $E(AV_C)$ from $E(M_T)$, p_t and p as specified in π_{5PM}^{M}.

$\pi_{\mathbf{S,AV}}$:

This subprotocol starts at global round 3 and ends at global round 5, with ZK preprocessing occurring during global rounds 1 and 2.

(1) $\mathcal{S}_S$ sends $E(M_{CDV})$, constructed using p_t, and $E(|p_t|)$ to Server. $\mathcal{S}_S$ also sends $A_{M01}^{P,2}$ to demonstrate that $E(M_{CDV})$ is formed correctly, where $A_{M01}^{P,1}$ and $A_{M01}^{V,1}$ occur during global rounds 1 and 2, respectively. A_{M01} continues until global round 5.

$\pi_{\mathbf{vec}}$:

This subprotocol begins at global round 3 and ends at global round 8, with ZK preprocessing occurring during global rounds 1, 2 and 3. $\langle \cdot, \cdot \rangle$ denotes the inner product over $\mathbb{G}_q$.

- At global round 3 $\mathcal{S}_S$ chooses $r_1 \in \mathbb{G}_q^{n-m+1}$ and $r_1' \in \mathbb{G}_q$. $\mathcal{S}_S$ computes $E(AV_C')$ by multiplying each element of AV_C with an encryption of 0 thus blinding the ciphertext. $\mathcal{S}_S$ generates $comm(E(AV_C'))$, $comm(r_1)$, and $comm(E(r_1'))$ and sends them to Server.
- Server sends $r_2, E(r_2'), E(AV_S')$ to $\mathcal{S}_S$.
- At global round 5, $\mathcal{S}_S$ sets $r = r_1 + r_2$ and $E(r') = E(r_1' + r_2')$. $\mathcal{S}_S$ computes $z_1 = E(\langle AV_C, r \rangle + r')$ and $z_2 = E(\langle AV_S, r \rangle + r')$. $\mathcal{S}_S$ opens the commitments of $E(AV_C')$, r_1, and $E(r_1')$ to Server. $\mathcal{S}_S$ sends z_1 and z_2 to Server. $\mathcal{S}_S$ computes partial decryptions $D_C(z_1)$, $D_C(z_2)$ and sends $D_C(z_1)$, $D_C(z_2)$ to Server as well as

$A_{PD}^{P,2}$ to prove that the partial decryptions $D_C(z_1)$, $D_C(z_2)$ are computed correctly, where messages $A_{PD}^{P,1}$ and $A_{PD}^{V,1}$ are sent during global rounds 1 and 2, respectively. Execution of $\mathcal{S}_S$'s A_{PD} continues until global round 7.

- At global round 6, Server sends (for independently obtained z_1 and z_2), $D_S(z_1)$ and $D_S(z_2)$ to $\mathcal{S}_S$ as well as $A_{PD}^{P,2}$ to prove that the partial decryptions $D_S(z_1)$, $D_S(z_2)$ are computed correctly, where $A_{PD}^{P,1}$ and $A_{PD}^{V,1}$ are sent during global rounds 2 and 3, respectively. Execution of Server's A_{PD} continues until global round 8.
- At global round 7 $\mathcal{S}_S$ obtains z_1, z_2 from $D_S(z_1)$ and $D_S(z_2)$. $\mathcal{S}_S$ aborts if $z_1 \neq z_2$.

$\mathcal{S}_S$ aborts if the decryptions of z_1 and z_2 do not equal each other.

$\pi_{\mathbf{rand}}$:

This subprotocol starts at global round 6 and ends at global round 8, with ZK preprocessing occurring during global rounds 2 and 3.

(1) Server sends $E(AV_S^r)$ to $\mathcal{S}_S$ and sends $A_{rand}^{P,2}$ that $E(AV_S^r)$ was obtained correctly from $E(AV_S')$, where $A_{rand}^{P,1}$ and $A_{rand}^{V,1}$ are sent during global rounds 2 and 3, respectively. This A_{rand} continues until global round 8.

$\pi_{\mathbf{ans}}$:

This subprotocol starts at global round 6 and continues until global round 8.

(1) At global round 6, Server sends $comm(D_S(E(AV_S^r)))$. Server also sends the message $comm(A_{FD}^{P,2})$, where A_{FD} is the argument to prove that either $D_S(E(AV_S^r))$ was obtained correctly or that Server knows s^* (for h^* sent by $\mathcal{S}_S$ in the first global round during π_{encr}), where $A_{FD}^{P,1}$ and $A_{FD}^{V,1}$ are sent during global rounds 2 and 3, respectively.

(2) At global round 7, $\mathcal{S}_S$ sends $A_{FD}^{V,2}$ to Server (our AoKs are public coin so $A_{FD}^{V,2}$ is not determined by $A_{FD}^{P,2}$).

(3) At global round 8, Server opens commitments of $A_{FD}^{P,2}$, $D_S(E(AV_S^r))$ to $\mathcal{S}_S$. Server

sends $A_{FD}^{P,3}$ to $\mathcal{S}_S$.

(4) $\mathcal{S}_S$ aborts if it does not accept the argument A_{FD}.

7.3.2 Simulator specification for an adversarial Client, $\mathcal{S}_C$, for π_{5PM}^M

Consider the admissible pair $\bar{P}$ = (Client,Server) in the real world. We construct $\mathcal{S}_S$ for an admissible pair $\bar{P}'$ =($\mathcal{S}_C$, Server) in the ideal world (where Server behaves honestly in both cases) such that $REAL_{\bar{P}}^{\pi_{5PM}^M}(\bar{x}, \bar{y}, \bar{r})$ and $IDEAL_{\bar{P}'}^{\pi_{5PM}^M}(\bar{x}, \bar{y}, \bar{r})$ are computationally indistinguishable. Note that $\mathcal{S}_S$ has oracle access to real world Client.

We list the subprotocols of π_{5PM}^M and describe how Client interacts with $\mathcal{S}_C$. Unlike the case for $\mathcal{S}_S$, $\mathcal{S}_C$ must interact with the ideal functionality in the final rounds of the protocol because Client receives an output from π_{5PM}^M; thus $\mathcal{S}_C$ must retrieve this output in the final steps of the simulation. Recall that π_{encr} is a protocol to instantiate a threshold ElGamal encryption scheme, $\pi_{C,AV}$ is for Client to compute an activation vector, $\pi_{S,AV}$ is for Server to compute an activation vector, π_{vec} is for the parties to determine that their activation vectors are equal, π_{rand} is for Server to send Client an encrypted vector that only reveals matching locations upon decryption, and π_{ans} is for Server to partially decrypt that encrypted vector. We refer the reader to Section 3.3.3 and Section 6.2 for details.

Initial Interactions with the Ideal Functionality: Once the ideal functionality reveals to $\mathcal{S}_C$ the length that the text should be, $\mathcal{S}_C$ sets the text $T = T^*$ to be all 1s of the right length (any arbitrary vector can be used here, but without loss of generality we use all 1s where $1 \in \Sigma$ and some other fixed character in Σ otherwise).

$\pi_{\mathbf{encr}}$:

This subprotocol begins at the first global round and ends at global round 6.

(1) Client sends h_1, h^* to $\mathcal{S}_C$ (where in the honest execution, $h_1 = g^{s_C}$ and $h^* = g^{s^*}$).

Client sends $\mathcal{S}_C$ two parallel instantiations of $\pi_{DL}^{P,1}$ for h_1 and h^* with witnesses s_C and s^*. This continues through global round 5.

(2) $\mathcal{S}_C$ chooses $s_S \in \mathbb{Z}_q$ and sets $h_2 \leftarrow g^{s_S}$. $\mathcal{S}_C$ sends h_2 to Client as well as $\pi_{DL}^{P,1}$ for h_2 with witness s_S. Protocol π_{DL} continues through global round 6. Both parties set $h = h_1 h_2$ as the public-key (with secret-key $s = s_C + s_S$).

(3) Using as a subprotocol the extractor E guaranteed since π_{DL} is an argument of knowledge, $\mathcal{S}_C$ rewinds from global round 5 to global round 2 and interacts with Client until global round 5 and then rewinds again until it extracts s^* from Client, at which point it rewinds once more and executes π_{DL} per the usual specification. $\mathcal{S}_C$ only uses E for its responses related to the proof of knowledge π_{DL} for s^*; only messages relating to π_{DL} for s^* are affected from one rewinding to the next. $\mathcal{S}_C$, using E, is guaranteed to succeed in extracting s^* in (expected) polynomial time since π_{DL} is an argument of knowledge.

(4) Using as a subprotocol the extractor E guaranteed since π_{DL} is an argument of knowledge, $\mathcal{S}_C$ rewinds from global round 5 to global round 2 and interacts with Client until global round 5 and then rewinds again until it extracts s_C from Client and then rewinds again until it extracts s^* from Client, at which point it rewinds once more and executes π_{DL} per the usual specification. $\mathcal{S}_C$ only uses E for its responses related to the proof of knowledge π_{DL} for s_C; only messages relating to π_{DL} for s_C are affected from one rewinding to the next. $\mathcal{S}_C$, using E, is guaranteed to succeed in extracting s_C in (expected) polynomial time since π_{DL} is an argument of knowledge. Note now that $\mathcal{S}_C$ can decrypt encryptions computed by Client because $\mathcal{S}_C$ possesses both s_S and s_C.

$\pi_{\mathbf{C,AV}}$:

This subprotocol starts at global round 2 and ends at global round 6.

(1) $\mathcal{S}_C$ sends $E(M_T)$ to Client. $\mathcal{S}_C$ also sends, for $E(M_T)$, $A_{M1}^{P,1}$ that $E(M_T)$ is formatted

correctly. $A_{M,1}$ continues through global round 6.

$\pi_{\mathbf{S,AV}}$:

This subprotocol starts at global round 3 and ends at global round 5, with ZK preprocessing occurring during global rounds 1 and 2.

(1) Client sends $E(M_{CDV})$ and $E(|p|)$ to $\mathcal{S}_C$. Client also sends $A^{P,2}_{M01}$ that $E(M_{CDV})$ is formed correctly, where $A^{P,1}_{M01}$ and $A^{V,1}_{M01}$ occur during global rounds 1 and 2, respectively. A_{M01} continues until global round 5.

(2) During global round 4, using $E(M_{CDV})$, T and $E(|p|)$, $\mathcal{S}_C$ computes $E(AV_S)$ as specified in π^H_{5PM}.

Further interaction with the ideal functionality: $\mathcal{S}_C$ can decrypt encryptions computed by Client because it extracted Client's secret-key s_C during π_{encr}. During $\pi_{S,AV}$, $\mathcal{S}_C$ obtains $E(M_{CDV})$. Therefore, $\mathcal{S}_C$ obtains the pattern p that Client is trying to have matched (which may be different than the pattern Client output to the real-world transcript). $\mathcal{S}_C$ resets the ideal functionality, now using this p. The output for the ideal functionality to $\mathcal{S}_C$ will be the correct matching of the pattern with the text that real Server is using. See π_{ans}, where $\mathcal{S}_C$ uses this ideal functionality output.

$\pi_{\mathbf{vec}}$:

This subprotocol begins at global round 3 and ends at global round 8, with ZK preprocessing occurring during global rounds 1, 2 and 3. $\langle \cdot, \cdot \rangle$ denotes the inner product over $\mathbb{G}_q$.

- At global round 3 Client chooses $r_1 \in \mathbb{G}_q^{n-m+1}$ and $r'_1 \in \mathbb{G}_q$. Client computes $E(AV'_C)$. Client generates $comm(E(AV'_C))$, $comm(r_1)$, and $comm(E(r'_1))$ and sends them to $\mathcal{S}_C$.

- At global round 4 $\mathcal{S}_C$ chooses $r_2 \in \mathbb{G}_q^{n-m+1}$ and $r'_2 \in \mathbb{G}_q$. $\mathcal{S}_C$ computes $E(AV'_S)$ by multiplying each element of AV_S by an encryption of 0 thus blinding the ciphertext. $\mathcal{S}_C$ sends $r_2, E(r'_2), E(AV'_S)$ to Client.
- At global round 5, Client opens the commitments of $E(AV'_C)$, r_1, and $E(r'_1)$ to $\mathcal{S}_C$. Client sends elements z_1 and z_2 to $\mathcal{S}_C$. Client sends $D_C(z_1)$, $D_C(z_2)$ to $\mathcal{S}_C$ as well as $A_{PD}^{P,2}$ to prove that the partial decryptions $D_C(z_1)$, $D_C(z_2)$ are computed correctly, where messages $A_{PD}^{P,1}$ and $A_{PD}^{V,1}$ are sent during global rounds 1 and 2, respectively. Execution of Client's A_{PD} continues until global round 7.
- At global round 6, $\mathcal{S}_C$ obtains z_1, z_2 from $D_C(z_1)$ and $D_C(z_2)$. $\mathcal{S}_C$ aborts if $z_1 \neq z_2$. $\mathcal{S}_C$ sets $r = r_1 + r_2$ and $E(r') = E(r'_1 + r'_2)$. $\mathcal{S}_C$ computes $z_1 = E(\langle AV_C, r\rangle + r')$ and $z_2 = E(\langle AV_S, r\rangle + r')$. $\mathcal{S}_C$ computes partial decryptions $D_S(z_1)$, $D_S(z_2)$ and sends $D_S(z_1)$ and $D_S(z_2)$ to Client as well as $A_{PD}^{P,2}$ to prove that the partial decryptions $D_S(z_1)$, $D_S(z_2)$ are computed correctly, where $A_{PD}^{P,1}$ and $A_{PD}^{V,1}$ are sent during global rounds 2 and 3, respectively. Execution of $\mathcal{S}_C$'s A_{PD} continues until global round 8.

$\pi_{\mathbf{rand}}$:

This subprotocol starts at global round 6 and ends at global round 8, with ZK preprocessing occurring during global rounds 2 and 3.

(1) $\mathcal{S}_C$ computes $E(AV_S^r)$ from $E(AV'_S)$ by exponentiating each encryption in $E(AV'_S)$ by a random exponent. $\mathcal{S}_C$ sends $E(AV_S^r)$ to Client and sends $A_{rand}^{P,2}$ that $E(AV_S^r)$ was obtained correctly from $E(AV'_S)$, where $A_{rand}^{P,1}$ and $A_{rand}^{V,1}$ are sent during global rounds 2 and 3, respectively. This A_{rand} continues until global round 8.

$\pi_{\mathbf{ans}}$:

This subprotocol starts at global round 6 and continues until global round 8.

(1) At this stage (global round 6), the ideal functionality sends $\mathcal{S}_C$ the correct output

that Client should receive (e.g., the locations in a string of length $|AV_S|$ that tells Client where matches occur- see "Further interactions with the ideal functionality" after $\pi_{S,AV}$ above). Denote as AV_{IF} the string that Client should receive per the ideal functionality. $\mathcal{S}_C$ computes $E(AV_{IF}^r)$ from AV_{IF} by setting the non-matching locations of AV_{IF} to be random elements and then encrypting the vector. $\mathcal{S}_C$ computes and sends $comm(D_S(E(AV_{IF}^r)))$ to Client. $\mathcal{S}_C$ also sends the message $comm(A_{FD}^{P,2})$, where A_{FD} is the argument to prove that either $D_S(E(AV_{IF}^r))$ was obtained correctly or that $\mathcal{S}_C$ knows s^* (for h^* sent by Client in the first global round during π_{encr}), where $A_{FD}^{P,1}$ and $A_{FD}^{V,1}$ are sent during global rounds 2 and 3, respectively. Note that here, $\mathcal{S}_C$ uses the witness for knowledge of s^*.

(2) At global round 7, Client sends $A_{FD}^{V,2}$ to $\mathcal{S}_C$ (our AoKs are public coin so $A_{FD}^{V,2}$ is not determined by $A_{FD}^{P,2}$).

(3) At global round 8, $\mathcal{S}_C$ opens commitments of $A_{FD}^{P,2}$, $D_S(E(AV_{IF}^r))$ to Client. $\mathcal{S}_C$ sends $A_{FD}^{P,3}$ to Client.

7.3.3 Security of π_{5PM}^M

We prove that π_{5PM}^M securely realizes $\mathcal{F}_{5PM}^M$ in the malicious (static corruption) model.

Theorem 7.4. *Assuming the Decisional Diffie Hellman (DDH) problem is hard, π_{5PM}^M securely realizes $\mathcal{F}_{5PM}^M$ in the malicious (static corruption) model.*

Proof of Theorem 7.4. We proceed to prove Theorem 7.4 by considering two cases: the first where the Client is corrupted and the second where the Server is corrupted.

Case 1: Client is corrupted.

We prove security by examining a sequence of experiments. Note that we assume here that the subprotocols of π_{5PM}^M are run *consecutively* instead of interleaved in order to simplify the proof. We will argue why interleaving does not affect our reasoning afterwards. We

assume that Client has pattern p and Server has text T.

Intuition. The intuition for the proof is as follows: As long as $\mathcal{S}_C$ completes the last proof π_{fin} according to specification, $\mathcal{S}_C$ can use any text he wishes. What is required is a sequence of hybrid arguments that begins with the real Server's actual text and concludes with $\mathcal{S}_C$ using a dummy text T^* (namely 1^n). However, for a security reduction to the semantic security of El Gamal, the zero-knowledge arguments executed during the reduction must be executed without plain-text witnesses. Therefore, what first must be undertaken is that zero-knowledge proofs must be executed by zero-knowledge simulation in order to be executed irrespective of (plain text) witnesses, then the text T is shifted to T^*, then, per specification of $\mathcal{S}_C$, the zero-knowledge arguments are then executed using plain text witnesses again. Formal arguments follow.

$\mathbf{H}_0$. In this experiment Client and Server interact as in the real world.

$\mathbf{H}_{1,1}$. This experiment is identical to H_0 except that $\mathcal{S}_C$ executes π_{enc} with the Client and extracts s^* from the Client execution of π_{DL} for knowledge of s^*. Computational indistinguishability holds because the distribution for the outputs for π_{DL} are not affected from H_0 to $H_{1,1}$ as $\mathcal{S}_C$, after extraction, executes π_{DL} as verifier per specification of π_{DL}.

$\mathbf{H}_{1,2}$. This experiment is identical to H_0 except that $\mathcal{S}_C$ executes π_{enc} with the Client and extracts s_C from the Client execution of π_{DL} for knowledge of s_C. Computational indistinguishability holds because the distribution for the outputs for π_{DL} are not affected from $H_{1,1}$ to $H_{1,2}$ as $\mathcal{S}_C$, after extraction, executes π_{DL} as verifier per specification of π_{DL}.

$\mathbf{H}_2$. This experiment is identical to $H_{1,2}$ except that for the proof of knowledge π_{DL} of the Server-side secret key s_S, $\mathcal{S}_C$ simulates the transcript rather than actually using the witness s_S.

We prove that H_2 is computationally indistinguishable from $H_{1,2}$ by reducing to the computational indistinguishability of ZK transcripts. Assume that there existed a pattern p, Server text T, Client strategy and Server/$\mathcal{S}_C$ such that a distinguisher D could distinguish $H_{1,2}$ from H_2 in polynomial time with non-negligible probability. Then let P_{ZK} and V_{ZK} be prover and verifier for the zero knowledge argument π_{DL} for the g^{s_S}. Let V_{ZK} have as auxiliary inputs p and T. P_{ZK} also has p and T as auxiliary inputs, as well as s_S; note that these auxiliary inputs may be given to P_{ZK} after V_{ZK} has internally executed the corresponding component of π^M_{5PM}.

V_{ZK} executes π^M_{5PM} internally with input T for Server and input p for Client, and executes the protocol as H_1 and H_2 specify. However, for the proof π_{DL} for g^{s_S}, V_{ZK} interacts with P_{ZK} as follows. For one of the two executions for π_{DL}, P_{ZK} executes π_{DL} with witness s_S; for the other execution, P_{ZK} uses the simulator $\mathcal{S}_{ZK}$ (guaranteed to exist by definition of zero knowledge) to construct a valid transcript without knowledge of the plain text/witness. V_{ZK} responds to these two interactions per the output of the internally executed Client. Once the two interactions are done, V_{ZK} completes the internal execution of π^M_{5PM}. Note that the views of V_{ZK} are identical to the views of the interactions specified by H_1 and H_2. Therefore, the zero knowledge distinguisher D_{ZK} distinguishes the two cases of V_{ZK}'s interaction with non-negligible probability by running D internally, which will distinguish the two views of the ZK execution with non-negligible probability, a contradiction.

$\mathbf{H}^i_3$. This experiment is identical to H_2 except that for each of the first i executions of π_{isBit} for M_T in $\pi_{C,AV}$, $0 \leq i \leq n|\Sigma|$, $\mathcal{S}_C$ produces a valid transcript for the ZK argument without using the witness for the encryption. We note that the encryptions that $\mathcal{S}_C$ uses are unchanged from H_1; only the ZK transcripts for π_{isBit} in $\pi_{C,AV}$ are affected. Note that H_2 is identical to H^0_3.

We prove that H^i_3 is computationally indistinguishable from H^{i+1}_3 by reducing to the computational indistinguishability of ZK transcripts. Assume that there existed a pattern p, Server text T, Client strategy and Server/$\mathcal{S}_C$ such that a distinguisher D could distinguish

H_3^i from H_3^{i+1} in polynomial time with non-negligible probability. Then let P_{ZK} and V_{ZK} be prover and verifier for the zero knowledge argument π_{DL} for the ith encryption of M_T. Let V_{ZK} have as auxiliary inputs p and T. P_{ZK} also has p and T as auxiliary inputs, as well as the public key and the randomness used to encrypt the $i+1$ encryption of M_T; note that these auxiliary inputs may be given to P_{ZK} after V_{ZK} has internally executed the corresponding component of π_{5PM}^M.

V_{ZK} executes π_{5PM}^M internally with input T for Server and input p for Client, and executes the protocol as H_3^i and H_3^{i+1} specify. However, for the $i+1$ encryption of M_T, V_{ZK} interacts with P_{ZK} as follows. For one of the two executions for π_{isBit} for the $i+1$ encryption of M_T, P_{ZK} executes π_{isBit} with private input the witness for the encryption to run π_{DL}; for the other execution, P_{ZK} uses the simulator $\mathcal{S}_{ZK}$ (guaranteed to exist by definition of zero knowledge) to construct a valid transcript without knowledge of the plain text/witness. V_{ZK} responds to these two interactions per the output of the internally executed Client. Once the two interactions are done, V_{ZK} completes the internal execution of π_{5PM}^M. Note that the views of V_{ZK} are identical to the views of the interactions specified by H_3^i and H_3^{i+1}. Therefore, the zero knowledge distinguisher D_{ZK} distinguishes the two cases of V_{ZK}'s interaction with non-negligible probability by running D internally, which will distinguish the two views of the ZK execution with non-negligible probability, a contradiction.

$\mathbf{H}_4^i$. This experiment is identical to $H_2^{n|\Sigma|}$ except that for each of the i executions of π_{DL} for $\pi_{C,AV}$ (namely, that the sum of each row of M_T is a 1), $0 \leq i \leq n$, $\mathcal{S}_C$ produces a valid transcript for the ZK argument without using the witness for the encryption, which is a product of the encryptions in the ith column of $E(M_T)$. Note that H_4^0 is identical to $H_3^{n|\Sigma|}$

We prove that H_4^i is computationally indistinguishable from H_4^{i+1} by reducing to the computational indistinguishability of ZK transcripts. Assume that there existed a pattern p, Server text T, Client strategy and Server such that a distinguisher D could distinguish H_4^i from H_4^{i+1} in polynomial time with non-negligible probability. Then let P_{ZK} and V_{ZK} be prover and verifier for the zero knowledge argument π_{DL} for the product of the ith row

of $E(M_T)$. We let V_{ZK} have as auxiliary inputs p and T. P_{ZK} also has p and T as auxiliary inputs, as well as the public key and the randomness used to encrypt the $i+1$ row of encryptions of M_T; note that these auxiliary inputs may be given to P_{ZK} after V_{ZK} has internally executed the corresponding component of π^M_{5PM}.

V_{ZK} executes π^M_{5PM} internally with input T for Server and input p for Client, and executes the protocol as H^i_4 and H^{i+1}_4 specify. However, for the product of the $i+1$ row of $E(M_T)$, V_{ZK} interacts with P_{ZK} as follows. For one of the two executions, P_{ZK} uses the witness for the encryption to run π_{DL}, and for the other, P_{ZK} uses the simulator $\mathcal{S}_{ZK}$ (guaranteed to exist by definition of zero knowledge) to construct a valid transcript without knowledge of the plain text. V_{ZK} responds to these two interactions per the output of the internally executed Client. Once the two interactions are done, then V_{ZK} completes the internal execution of π^M_{5PM}. Note that the views of V_{ZK} are identical to the view of the interactions specified by H^i_4 and H^{i+1}_4. Therefore, the zero knowledge distinguisher D_{ZK} distinguishes the two cases of V_{ZK}'s interaction with non-negligible probability by running D internally, which will distinguish the two views with non-negligible probability, a contradiction.

$\mathbf{H^i_5}$. This experiment is identical to H^n_3 except that the ith execution of π_{eqDL} executed as part of A_{PD} for correct server-side partial decryption of z_i during π_{vec} is executed without using the witness for partial decryption and instead is simulated ($0 \leq i \leq 2$, where the 0 case corresponds to H^n_4).

We prove that H^i_5 is computationally indistinguishable from H^{i+1}_5 by reducing to the computational indistinguishability of ZK transcripts. Assume that there existed a pattern p, Server text T, Client strategy and Server such that a distinguisher D could distinguish H^i_5 from H^{i+1}_5 in polynomial time with non-negligible probability. Then let P_{ZK} and V_{ZK} be prover and verifier for the zero knowledge argument π_{eqDL}. We let V_{ZK} have as auxiliary inputs p and T. P_{ZK} also has p and T as auxiliary inputs, Server-side secret key s_S, the decryption of z_i and the randomness used for the encryption; note that these auxiliary

inputs may be given to P_{ZK} after V_{ZK} has internally executed the corresponding component of π^M_{5PM}.

V_{ZK} executes π^M_{5PM} internally with input T for Server and input p for Client, and executes the protocol as H^i_5 and H^{i+1}_5 specify. However, for the proof of partial decryption by the Server for z_{i+1} in π_{vec}, V_{ZK} interacts with P_{ZK} as follows. For one of the two executions, P_{ZK} uses the witness s_S for the decryption to run π_{eqDL}, and for the other, P_{ZK} uses the simulator $\mathcal{S}_{ZK}$ (guaranteed to exist by definition of zero knowledge) to construct a valid transcript without knowledge of the secret key. V_{ZK} responds to these two interactions per the output of the internally executed Client. Once the two interactions are done, then V_{ZK} completes the internal execution of π^M_{5PM}. Note that the views of V_{ZK} are indistinguishable from the view of the interactions specified by H^i_5 and H^{i+1}_5. Therefore, the zero knowledge distinguisher D_{ZK} distinguishes the two cases of V_{ZK}'s interaction with non-negligible probability by running D internally, which will distinguish the two views with non-negligible probability, a contradiction.

$\mathbf{H^i_6}$. This experiment is identical to H^n_5 except for that for each of the first i executions of π_{fin} during A_{FD}, $0 \leq i \leq n-m+1$, for the proof of partial decryption of $E(AV^r_S)$, Server (or $\mathcal{S}_C$) uses the witness for s^* to complete the execution. Note that use of different witnesses is computationally indistinguishable for ZK arguments (since the executions with a different witness are themselves indistinguishable from the execution by the simulator). Note that H^0_6 is identical to H^n_5.

We prove that H^i_6 is computationally indistinguishable from H^{i+1}_6 by reducing to the computational indistinguishability of ZK transcripts. Assume that there existed a pattern p, Server text T, Client strategy and Server such that a distinguisher D could distinguish H^i_6 from H^{i+1}_6 in polynomial time with non-negligible probability. Then let P_{ZK} and V_{ZK} be prover and verifier for the zero knowledge argument π_{fin} for the ith encryption of $E(AV^r_S)$. We let V_{ZK} have as auxiliary inputs p and T. P_{ZK} also has p and T as auxiliary inputs, as well as the $i+1$ element of $E(AV^r_S)$, the corresponding Server-side partial decryption and the

Server-side private key. Finally, P_{ZK} also has s^* as auxiliary input; note that these auxiliary inputs may be given to P_{ZK} after V_{ZK} has internally executed the corresponding component of π^M_{5PM}.

V_{ZK} executes π^M_{5PM} internally with input T for Server and input p for Client, and executes the protocol as H^i_6 and H^{i+1}_6 specify. However, for $i+1$ encryption of $E(AV^r_S)$, V_{ZK} interacts with P_{ZK} as follows. For one of the two executions, P_{ZK} uses the witness for the partial decryption encryption of the $i+1$ element of $E(AV^r_S)$ to execute π_{fin} with V_{ZK}, and for the other, P_{ZK} uses the witness s^* to execute π_{fin}. V_{ZK} responds to these two interactions per the output of the internally executed Client. Once the two interactions are done, then V_{ZK} completes the internal execution of π^M_{5PM}. Note that the views of V_{ZK} are identical to the view of the interactions specified by H^i_6 and H^{i+1}_6. Therefore, the zero knowledge distinguisher D_{ZK} distinguishes the two cases of V_{ZK}'s interaction with non-negligible probability by running D internally, which will distinguish the two views with non-negligible probability, a contradiction.

$\mathbf{H^i_7}$. This experiment is identical to H^{n-m+1}_6 except that the first i entries of AV^r_{IF}, $0 \leq i \leq n-m+1$, are used instead of the corresponding entries of AV^r_S. Note that AV^r_{IF} and AV^r_S, are, by construction, distributed identically over the choice of their respective randomized entries (e.g,in the non-matching locations), and that both contain zeros in exactly the same places. Note that by the construction of H^n_6 the proof of partial decryption π_{fin} for $D_S(E(AV^r_{IF}))$ uses as witness s^* rather than the witness for decryption. Therefore H^i_7 is computationally (indeed, statistically) indistinguishable from H^{i+1}_7 for any i because of the identical distributions of AV^r_{IF} and AV^r_S.

$\mathbf{H^i_8}$. This experiment is identical to H^{n-m+1}_7 except that the first i encryptions of $E(M_T)$, $0 \leq i \leq n|\Sigma|$, are drawn from $E(M_{T'})$, which corresponds to $T' = 1^n$. Note that the final output of π^M_{5PM} is AV^r_{IF}, so the protocol output does not change. Note also that H^0_8 is identical to H^n_5.

We prove that H_8^i is computationally indistinguishable from H_8^{i+1} by reducing to the semantic security of ElGamal encryption (that is to say, assuming the hardness of Decisional Diffie Hellman); in particular, we will use a reduction to non-threshold (e.g, standard) ElGamal encryption. Assume that there existed a pattern p, Server text T, Client strategy and Server/$\mathcal{S}_C$ such that a distinguisher D could distinguish H_8^i from H_8^{i+1} in polynomial time with non-negligible probability. Then let S_{enc} be the computer of ElGamal encryptions and R_{enc} be the receiver. R_{enc} is given as auxiliary inputs p and T. S_{enc} also has p and T as auxiliary inputs, as well as the public key and the randomness used to encrypt the $i+1$ encryption of $M_T/M_{T'}$; note that these auxiliary inputs may be given to P_{ZK} after V_{ZK} has internally executed the corresponding component of π_{5PM}^M.

R_{enc} executes π_{5PM}^M internally with input T/T' for Server and input p for Client. However, when Server must publish g^{s_S}, R_{enc} queries S_{enc}, who sends g^{s_S}; this is the publishing of the public ElGamal key. Note that R_{enc} must select s_C independently of s_S or will not be able to create a transcript computationally indistinguishable from the π_{5PM}^M transcript; because of the binding and hiding property of Pedersen commitments, real Client and Server are required to do the same. R_{enc}, as Server, sends S_{enc} and completes the ZK proof without the witness s_S; since by an earlier hybrid experiment, H_2 this is already being accomplished, R_{enc} is acting according to specification.

For each encryption of M_T except for the i and $i+1$ entries, R_{enc} sends the unencrypted value to S_{enc}, who sends back the corresponding encryptions. R_{enc} encrypts with the Client's s_C (which it obtains by running the knowledge extractor; as in hybrid H_1, this does not affect transcript indistinguishability), and uses this final encryption as the Server-side encryption; this final encryption corresponds to encryption with the secret key $s_C + s_S$.

Note that in all cases, the ZK proofs of well-formedness of the encryptions are simulated rather than using the actual witnesses per hybrid experiments $H_2^{n|\Sigma|}$ and H_3^n.

For the i and $i+1$ encryptions, R_{enc} sends both to S_{enc}, who sends back encryptions of each, without identifying which encryption corresponds to which plaintext. R_{enc} completes the internal execution of π_{5PM}^M; note that witnesses for partial decryption by the Server do

not use s_S per construction of the previous hybrids (H_3 and H_4). We note that at the end of π_{vec}, Client and Server must jointly compute decryptions of z_1 and z_2; here, we note that R_{enc} will simply use the same (randomly chosen) "secret key" s_{fake} and use it for Server-side decryption of z_1 and z_2; if the actual decryptions of z_1 and z_2 equal each other, than so will these new "decryptions"; further, by the uniformly random distribution of the hash functions used to compute z_1 and z_2, the transcripts will be identical to the hybrid experiment executions. Further, Server does not use the witness for partial decryption due to hybrid H_5^2.

Finally, for the final decryption of AV_{IF}^r, R_{enc} can simply send the Server's partial decryption, $D_S(E(AV_{IF}^r))$, as the partial *encryption* of AV_{IF}^r using the Client's public key (note that the Server-side can extract this rather than having R_{enc} simply use it by virtue of executing Client internally). Again, the witness for partial decryption isn't used because by hybrid H_6, the witness s^* is already being used.

We note that by the above reasoning, the views of R_{enc} are distributed identically to the views of an adversarial Client for H_8^i and H_8^{i+1}. Therefore, if the distinguisher D_{enc} by executes the hybrid distinguisher D internally, then D_{enc} will be able to distinguish the i and $i+1$ encryptions of M_T with non-negligible probability, a contradiction.

$\mathbf{H_9^i}$. This experiment is identical to $H_8^{n|\Sigma|}$ except that for the ith execution of π_{eqDL} executed as part of A_{PD} for correct server-side partial decryption of z_i during π_{vec} ($0 \leq i \leq 2$, where the 0 case corresponds to H_4^n), Server uses the secret key s_S rather than simulating the transcript.

That the hybrids H_9^i and H_9^{i+1} are computationally indistinguishable is essentially the same argument as for hybrids H_5^i and H_5^{i+1} (which was the same process but in reverse).

$\mathbf{H_{10}^i}$. This experiment is identical to H_9^2 except that i executions of π_{DL} in $\pi_{C,AV}$, $0 \leq i \leq n$, $\mathcal{S}_C$ uses the witness for the encryption, which is a product of the encryptions in the ith row of $E(M_{T'})$. Note that H_{10}^0 is identical to H_9^2.

That hybrids H_{10}^i and H_{10}^{i+1} are computationally indistinguishable is essentially the same argument as for hybrids H_3^i and H_3^{i+1} (which was the same process but in reverse).

$\mathbf{H_{11}^i}$. This experiment is identical to H_{10}^n except that for each of the first i executions of π_{isBit} in $\pi_{C,AV}$, $0 \leq i \leq n|\Sigma|$, $\mathcal{S}_C$ uses the witness for the encryption. Note that H_{11}^0 is identical to H_{10}^n

That hybrids H_{11}^i and H_{11}^{i+1} are computationally indistinguishable is essentially the same argument as for hybrids H_3^i and H_3^{i+1} (which was the same process but in reverse).

$\mathbf{H_{12}}$. This experiment is identical to $H_{11}^{n|\Sigma|}$ except that $\mathcal{S}_C$ uses the witness s_S for π_{DL} in π_{vec}. Note that H_{12} is the distribution of view of output of the Simulator $\mathcal{S}_C$.

That hybrids H_{12} and $H_{11}^{n|\Sigma|}$ are computationally indistinguishable is the essentially the same argument as for hybrids $H_{1,2}$ and H_2 above.

Interleaving the ZK Arguments. The ZK arguments used in π_{5PM}^M are all modified Σ protocols. The first three global rounds of π_{5PM}^M force the respective ZK provers to instantiate all needed (equivocable) commitment schemes while forcing the respective ZK Verifiers to commit to all challenges for the subsequent Σ protocols. Thus, the Verifier has *less* power than in the sequential composition case (recall that ZK arguments are closed under sequential composition), since in the sequential composition case, the Verifier could change her challenges based on previous ZK iterations while here she cannot. Therefore, zero-knowledge is not affected. Soundness is also not affected by interleaving since a dishonest Prover must break soundness by distinguishing the Verifier's (public coin) committed challenge to the underlying Σ protocol. However, the Prover cannot distinguish multiple committed challenges so long as separate randomness was used for each commitment, yielding soundness of the ZK arguments even when they are interleaved as in π_{5PM}^M.

Interleaving Subprotocols. We now demonstrate that interleaving the subprotocols into the final π_{5PM}^{M} does not affect transcript indistinguishability. We do so by considering the interleaving subprotocols and demonstrating that each new subprotocol's transcript is not affected by interleaving. We can consider this sequential case because it is already demonstrated (above) that if each subprotocol is executed sequentially without interleaving, then the protocol is secure in the malicious model.

We denote by A_{rel}^{S} and A_{rel}^{C} (also by, for instance, π_{rel}^{S}) the zero-knowledge building blocks outlined in Section 6.1 where Server and Client, respectively, act as prover. As shorthand, we will write $A_{rel}^{S}(x)$ as the argument A_{rel} where S is the prover and x is the common input.

π_{encr} starts at global round 1. The preprocessing steps for all required ZK protocols occur during π_{encr}; their outputs here are indistinguishable in the real versus ideal settings by the fact that interleaved zero-knowledge protocols still yield indistinguishable transcripts. While $\mathcal{S}_C$ must rewind to extract s^* and s_C, this does not change the distribution of $\mathcal{S}_C$'s output to Client's view.

$\pi_{C,AV}$ starts at global round 2, after the global public-key has been determined. Indistinguishability is not affected since π_{encr} is a protocol for setting up the threshold encryption scheme while $\pi_{C,AV}$ is a protocol for proper pattern formation (e.g,of $E(M_{CDV})$); since these protocols deal with independent inputs (other than $\pi_{C,AV}$ needs the existence of the threshold encryption scheme), interleaving them does not affect transcript indistinguishability.

$\pi_{S,AV}$ starts at global round 3. The only remaining outputs from π_{encr} and $\pi_{C,AV}$ are the remaining zero-knowledge argument outputs of $\pi_{DL}^{C}(h_1)$, $\pi_{DL}^{C}(h^*)$, $\pi_{DL}^{S}(h_2)$, and $A_{M1}^{S}(E(M_T))$, which are independent of the non ZK-related outputs of $\pi_{S,AV}$ and therefore do not affect indistinguishability. Interleaving the ZK-related outputs of $A_{M01}^{C}(E(M_{CDV}))$ with remaining ZK outputs of previous subprotocol does not affect their indistinguishability (since they could be considered as auxiliary inputs in the ZK security proof).

π_{vec} starts at global round 4. The only remaining outputs from the previous subprotocols are from the ZK arguments $\pi^C_{DL}(h_1)$, $\pi^C_{DL}(h^*)$, $\pi^S_{DL}(h_2)$, $A^S_{M1}(E(M_T))$, and $A^C_{M01}(E(M_{CDV}))$, whose interleaving do not affect indistinguishability since the non-ZK outputs of π_{vec} are chosen independently of previous ZK outputs and interleaving ZK arguments does not affect indistinguishability.

π_{rand} starts at global round 6, at the same time that $E(AV'_S)$ is introduced. Since π_{rand} only involves reformatting $E(AV'_S)$, namely by introducing $E(AV^r_S)$ and its associated ZK outputs, indistinguishability is not affected.

π_{ans} starts at global round 6; the ZK transcript for A_{final} does not reveal any information by ZK indistinguishability and the final decryption is only revealed (by decommitment) at the very last round; distinguishing the transcript here would imply violating the commitment scheme (e.g,Pedersen commitments, which are perfectly hiding).

This completes the proof for the indistinguishability of an adversarial Client's views.

Case 2: Server is corrupted.

We prove security by examining a sequence of experiments. Note that we assume here that the subprotocols of π^M_{5PM} are run *consecutively* instead of interleaved in order to simplify the proof. We will argue why interleaving does not meaningfully affect the proof afterwards. We assume that Client has pattern p and Server has text T.

Intuition. The intuition for the proof is as follows: By contrast to the above case of a corrupted Client, here, Client can use any encryption he wants as long as the zero-knowledge proofs hold, and the views will still be computationally indistinguishable. However, for a security reduction to the semantic security of El Gamal encryption, the zero-knowledge arguments executed during the reduction must be executed without plain-text witnesses. Therefore, what first must be undertaken is that zero-knowledge proofs must be executed by zero-knowledge simulation in order to be executed irrespective of (plain text) witnesses, then the text p is shifted to p^*, then, per specification of $\mathcal{S}_S$, the zero-knowledge arguments

are then executed using plain text witnesses again. Formal arguments follow.

$\mathbf{H}_0$. In this experiment Client and Server interact as in the real world.

$\mathbf{H}_1$. This experiment is identical to H_0 except that for the proof of knowledge π_{DL} of the Client-side secret key s_C, $\mathcal{S}_S$ simulates the transcript rather than actually using the witness s_C.

We prove that H_1 is computationally indistinguishable from H_1 by reducing to the computational indistinguishability of ZK transcripts. Assume that there existed a pattern p, Server text T, Server strategy and Client/$\mathcal{S}_S$ such that a distinguisher D could distinguish H_0 from H_1 in polynomial time with non-negligible probability. Then let P_{ZK} and V_{ZK} be prover and verifier for the zero knowledge argument π_{DL} for the g^{s_C}. Let V_{ZK} have as auxiliary inputs p and T. P_{ZK} also has p and T as auxiliary inputs, as well as s_S; note that these auxiliary inputs may be given to P_{ZK} after V_{ZK} has internally executed the corresponding component of π^M_{5PM}.

V_{ZK} executes π^M_{5PM} internally with input T for Server and input p for Client, and executes the protocol as H_0 and H_1 specify. However, proof π_{DL} for g^{s_C}, V_{ZK} interacts with P_{ZK} as follows. For one of the two executions for π_{DL}, P_{ZK} executes π_{DL} with witness s_C; for the other execution, P_{ZK} uses the simulator $\mathcal{S}_{ZK}$ (guaranteed to exist by definition of zero knowledge) to construct a valid transcript without knowledge of the plain text/witness. V_{ZK} responds to these two interactions per the output of the internally executed Server. Once the two interactions are done, V_{ZK} completes the internal execution of π^M_{5PM}. Note that the views of V_{ZK} are identical to the view of the interactions specified by H_0 and H_1. Therefore, the zero knowledge distinguisher D_{ZK} distinguishes the two cases of V_{ZK}'s interaction with non-negligible probability by running D internally, which will distinguish the two views of the ZK execution with non-negligible probability, a contradiction.

$\mathbf{H}^i_2$. This experiment is identical to H_1 except that for each of the first i executions of π_{isBit}

for M_{CDV} in $\pi_{S,AV}$, $0 \leq i \leq m|\Sigma|$, $\mathcal{S}_S$ produces a valid transcript for the ZK argument without using the witness for the encryption. We note that the encryptions that $\mathcal{S}_S$ uses are unchanged from H_0; only the ZK transcripts for π_{isBit} in $\pi_{S,AV}$ are affected. Note that H_1 is identical to H_2^0.

We prove that H_2^i is computationally indistinguishable from H_2^{i+1} by reducing to the computational indistinguishability of ZK transcripts. Assume that there existed a pattern p, Server text T, Server strategy and Client/$\mathcal{S}_S$ such that a distinguisher D could distinguish H_2^i from H_2^{i+1} in polynomial time with non-negligible probability. Then let P_{ZK} and V_{ZK} be prover and verifier for the zero knowledge argument π_{DL} for the ith encryption of M_{CDV}. Let V_{ZK} have as auxiliary inputs p and T. P_{ZK} also has p and T as auxiliary inputs, as well as the public key and the randomness used to encrypt the $i+1$ encryption of M_{CDV}; note that these auxiliary inputs may be given to P_{ZK} after V_{ZK} has internally executed the corresponding component of π_{5PM}^M.

V_{ZK} executes π_{5PM}^M internally with input T for Server and input p for Client, and executes the protocol as H_2^i and H_2^{i+1} specify. However, for the $i+1$ encryption of M_{CDV} in $\pi_{S,AV}$, V_{ZK} interacts with P_{ZK} as follows. For one of the two executions for π_{isBit} for the $i+1$ encryption of M_{CDV}, P_{ZK} executes π_{isBit} with private input the witness for the encryption to run π_{DL}; for the other execution, P_{ZK} uses the simulator $\mathcal{S}_{ZK}$ (guaranteed to exist by definition of zero knowledge) to construct a valid transcript without knowledge of the plain text/witness. V_{ZK} responds to these two interactions per the output of the internally executed Server. Once the two interactions are done, V_{ZK} completes the internal execution of π_{5PM}^M. Note that the views of V_{ZK} are identical to the view of the interactions specified by H_2^i and H_2^{i+1}. Therefore, the zero knowledge distinguisher D_{ZK} distinguishes the two cases of V_{ZK}'s interaction with non-negligible probability by running D internally, which will distinguish the two views of the ZK execution with non-negligible probability, a contradiction.

$\mathbf{H}_3^i$. This experiment is identical to $H_2^{m|\Sigma|}$ except that for each of the i executions of π_{isBit} for

$\pi_{S,AV}$ (namely, that the sum of each column of M_{CDV} is a 0 or a 1), $0 \leq i \leq m$, $\mathcal{S}_S$ produces a valid transcript for the ZK argument without using the witness for the encryption, which is a product of the encryptions in the ith column of $E(M_{CDV})$. Note that H_3^0 is identical to $H_2^{m|\Sigma|}$

We prove that H_3^i is computationally indistinguishable from H_3^{i+1} by reducing to the computational indistinguishability of ZK transcripts. Assume that there existed a pattern p, Server text T, Client/$\mathcal{S}_S$ and Server strategy such that a distinguisher D could distinguish H_3^i from H_3^{i+1} in polynomial time with non-negligible probability. Then let P_{ZK} and V_{ZK} be prover and verifier for the zero knowledge argument π_{isBit} for the product of the $i+1$th column of $E(M_{CDV})$. We let V_{ZK} have as auxiliary inputs p and T. P_{ZK} also has p and T as auxiliary inputs, as well as the public key and the randomness used to encrypt the $i+1$ column of encryptions of M_{CDV}; note that these auxiliary inputs may be given to P_{ZK} after V_{ZK} has internally executed the corresponding component of π_{5PM}^M.

V_{ZK} executes π_{5PM}^M internally with input T for Server and input p for Client, and executes the protocol as H_3^i and H_3^{i+1} specify. However, for the product of the $i+1$ column of $E(M_{CDV})$, V_{ZK} interacts with P_{ZK} as follows. For one of the two executions, P_{ZK} uses the witness for the encryption to run π_{DL}, and for the other, P_{ZK} uses the simulator $\mathcal{S}_{ZK}$ (guaranteed to exist by definition of zero knowledge) to construct a valid transcript without knowledge of the plain text. V_{ZK} responds to these two interactions per the output of the internally executed Server. Once the two interactions are done, then V_{ZK} completes the internal execution of π_{5PM}^M. Note that the views of V_{ZK} are identical to the view of the interactions specified by H_3^i and H_3^{i+1}. Therefore, the zero knowledge distinguisher D_{ZK} distinguishes the two cases of V_{ZK}'s interaction with non-negligible probability by running D internally, which will distinguish the two views with non-negligible probability, a contradiction.

$\mathbf{H_4^i}$. This experiment is identical to H_3^m except that the ith execution of π_{eqDL} executed as part of A_{PD} for correct client-side partial decryption of z_i during π_{vec} is executed without

using the witness for partial decryption and instead is simulated ($0 \leq i \leq 2$, where the 0 case corresponds to H_3^n).

We prove that H_4^i is computationally indistinguishable from H_4^{i+1} by reducing to the computational indistinguishability of ZK transcripts. Assume that there existed a pattern p, Server text T, Client strategy and Server such that a distinguisher D could distinguish H_4^i from H_4^{i+1} in polynomial time with non-negligible probability. Then let P_{ZK} and V_{ZK} be prover and verifier for the zero knowledge argument π_{eqDL}. We let V_{ZK} have as auxiliary inputs p and T. P_{ZK} also has p and T as auxiliary inputs, Client-side secret key s_C, the decryption of z_i and the randomness used for the encryption; note that these auxiliary inputs may be given to P_{ZK} after V_{ZK} has internally executed the corresponding component of π_{5PM}^M.

V_{ZK} executes π_{5PM}^M internally with input T for Server and input p for Client, and executes the protocol as H_4^i and H_4^{i+1} specify. However, for the proof of partial decryption by the Client for z_{i+1} in π_{vec}, V_{ZK} interacts with P_{ZK} as follows. For one of the two executions, P_{ZK} uses the witness s_C for the decryption to run π_{eqDL}, and for the other, P_{ZK} uses the simulator $\mathcal{S}_{ZK}$ (guaranteed to exist by definition of zero knowledge) to construct a valid transcript without knowledge of the secret key V_{ZK} responds to these two interactions per the output of the internally executed Server. Once the two interactions are done, then V_{ZK} completes the internal execution of π_{5PM}^M. Note that the views of V_{ZK} are identical to the view of the interactions specified by H_4^i and H_4^{i+1}. Therefore, the zero knowledge distinguisher D_{ZK} distinguishes the two cases of V_{ZK}'s interaction with non-negligible probability by running D internally, which will distinguish the two views with non-negligible probability, a contradiction.

$\mathbf{H_5^i}$. This experiment is identical to H_4^n except that the first i encryptions of $E(M_{CDV})$, $0 \leq i \leq m|\Sigma|$, correspond to $E(M_{CDV'})$ corresponding to $p' = 1^n$. Note that H_5^0 is identical to H_4^m.

We prove that H_5^i is computationally indistinguishable from H_5^{i+1} by reducing to the

semantic security of ElGamal encryption (that is to say, assuming the hardness of Decisional Diffie Hellman); in particular, we will use a reduction to non-threshold (e.g,standard) ElGamal encryption. Assume that there existed a pattern p, Server text T, Server strategy and Client/$\mathcal{S}_S$ such that a distinguisher D could distinguish H_5^i from H_5^{i+1} in polynomial time with non-negligible probability. Then let S_{enc} be the computer of ElGamal encryptions and R_{enc} be the receiver. R_{enc} is given as auxiliary inputs p and T. S_{enc} also has p and T as auxiliary inputs, as well as the public key and the randomness used to encrypt the $i+1$ encryption of $M_{CDV}/M_{CDV'}$; note that these auxiliary inputs may be given to P_{ZK} after V_{ZK} has internally executed the corresponding component of π_{5PM}^M.

R_{enc} executes π_{5PM}^M internally with input T for Server and input p/p' for Client. However, when Client must publish g^{s_C}, R_{enc} queries S_{enc}, who sends g^{s_C}; this is the publishing of the public ElGamal key. Note that R_{enc} must select s_S independently of s_C or will not be able to create a transcript computationally indistinguishable from the π_{5PM}^M transcript; because of the binding and hiding property of Pedersen commitments, real Client and Server are required to do the same. R_{enc}, as Server, sends S_{enc} and completes the ZK proof without the witness s_C as per hybrid H_1; since by an earlier hybrid experiment this is already being accomplished, R_{enc} is acting according to specification.

For each encryption of M_{CDV} except for the i and $i+1$ entries, R_{enc} sends the unencrypted value to S_{enc}, who sends back the corresponding encryptions. R_{enc} encrypts with the Server's s_S (which it obtains by running the knowledge extractor; as in hybrid H_1, this does not affect transcript indistinguishability), and uses this final encryption as the Client-side encryption; this final encryption corresponds to encryption with the secret key $s_C + s_S$.

Note that in all cases, the ZK proofs of well-formedness of the encryptions are simulated rather than using the actual witnesses per hybrid experiments $H_1^{m|\Sigma|}$ and H_2^m.

For the i and $i+1$ encryptions, R_{enc} sends both to S_{enc}, who sends back encryptions of each, without identifying which encryption corresponds to which plaintext. R_{enc} completes the internal execution of π_{5PM}^M; note that witnesses for partial decryption by the Client do not use s_C per construction of the previous hybrids (H_1 and H_2). We note that at the end

of π_{vec}, Client and Server must jointly compute decryptions of z_1 and z_2; here, we note that R_{enc} will simply use the same (randomly chosen) "secret key" s_{fake} and use it for Client-side decryption of z_1 and z_2; if the actual decryptions of z_1 and z_2 equal each other, than so will these new "decryptions"; by the uniformly random distribution of the hash functions used to compute z_1 and z_2, the transcripts will be identical to the hybrid experiment executions. Further, Server does not use the witness for partial decryption due to hybrid H_4^2.

Note that no protocol output is part of the view/distribution of the hybrid experiment here, and so that the pattern matching output changes is not a concern (the ideal functionality does not output to an adversarial Server). In particular, because the adversarial Server never learns the pattern match output, she can never output it out of turn at some point and skew the distribution of the hybrid experiments.

We note that by the above reasoning, the views of R_{enc} are computationally indistinguishable from the views of an adversarial Client for H_5^i and H_5^{i+1}. Therefore, if the distinguisher D_{enc} by executes the hybrid distinguisher D internally, then D_{enc} will be able to distinguish the i and $i+1$ encryptions of M_{CDV} with non-negligible probability, a contradiction.

$\mathbf{H_6^i}$. This experiment is identical to $H_5^{m|\Sigma|}$ except that for the ith execution of π_{eqDL} executed as part of A_{PD} for correct server-side partial decryption of z_i during π_{vec} ($0 \leq i \leq 2$, where the 0 case corresponds to H_3^n), Client uses the secret key s_C rather than simulating the transcript.

That the hybrid H_6^i and H_6^{i+1} are computationally indistinguishable is essentially the same argument as for hybrids H_4^i and H_4^{i+1} (which was the same process but in reverse).

$\mathbf{H_7^i}$. This experiment is identical to H_6^2 except that i executions of π_{isBit} in $\pi_{S,AV}$, $1 \leq i \leq m$, $\mathcal{S}_C$ uses the witness for the encryption, which is a product of the encryptions in the ith row of $E(M_{CDV'})$. Note that H_7^0 is identical to H_6^2.

That hybrid H_7^i and H_7^{i+1} are computationally indistinguishable is essentially the same

argument as for hybrids H_3^i and H_3^{i+1} (which was the same process but in reverse).

$\mathbf{H_8^i}$ This experiment is identical to H_7^m except that for each of the first i executions of π_{isBit} in $\pi_{S,AV}$, $0 \leq i \leq m|\Sigma|$, $\mathcal{S}_S$ uses the witness for the encryption. Note that H_8^0 is identical to H_7^m; further, $H_8^{m|\Sigma|}$ is the distribution of view of output of the Simulator $\mathcal{S}_S$.

That hybrid H_8^i and H_8^{i+1} are computationally indistinguishable is essentially the same argument as for hybrids H_1^i and H_1^{i+1} (which was the same process but in reverse).

$\mathbf{H_9}$. This experiment is identical to $H_8^{m|\Sigma|}$ except that $\mathcal{S}_S$ uses the witness s_C for π_{DL} in π_{vec}. Note that H_9 is the distribution of view of output of the Simulator $\mathcal{S}_S$.

That hybrids H_9 and $H_8^{n|\Sigma|}$ are computationally indistinguishable is the essentially the same argument as for hybrids H_1 and H_2 above.

Interleaving the ZK Arguments. The argument for why interleaving ZK arguments does not break their security properties is explained in the proof-case of an adversarial Client (see Case 1 in this proof).

Interleaving Subprotocols. The reason why interleaving subprotocols does not affect indistinguishability is explained in the proof-case of an adversarial Client (see Case 1 in this proof).

This completes the proof for the indistinguishability of an adversarial Server's views.

Caveat for $\mathcal{S}_S$ In the case of an adversarial Server, a soundness-type argument is needed: if Server obtains s^* independently, it would be impossible to prove that Server had actually supplied the correct output for the functionality $\mathcal{F}_{5PM}^M$ during π_{ans} because of the witness hiding property of π_{fin}. To show that such an independent generation of s^* is impossible except with negligible probability, we assume by way of contradiction that Server had indeed generated s^* independently, and we let the simulator $\mathcal{S}_S$ adopt the following strategy:

(1) At global round 1, $\mathcal{S}_S$ sends Server h^* without knowing its discrete logarithm (which is s^*).

(2) Since Server must commit to its challenge, which occurs in global round 2 and which is decommitted in global round 4, $\mathcal{S}_S$ can rewind from global round 4 to global round 3 and change its output for π_{DL} proving knowledge of the discrete logarithm of h^* so that the $\mathcal{S}_S$ provides a valid proof for h^* without actually knowing s^*. This is due to the fact that π_{DL} is constructed from a Σ protocol (Σ_{DL}), which is honest-verifier zero-knowledge as defined in Section 4.

(3) At global round 6, Server begins a proof of knowledge A_{final} that either its decryption is correct or that it knows s^*. In this case, Server uses its knowledge of s^*. $\mathcal{S}_S$ can extract this s^* using a subprotocol E_{fin} for π_{fin} guaranteed by definition of argument of knowledge:

- In order to extract, $\mathcal{S}_S$ must be able to decommit to any message. In order to do so, $\mathcal{S}_S$, using the knowledge extractor E as a subprotocol, must rewind from global round 8 (where Server answers $\mathcal{S}_S$'s challenge to π_{final}) to the related outputs at global rounds 2 and 3, where the ZK preprocessing of A_{final} occurs, then interact with Server anew until round 8, and then rewind again to round 2, per E's specifications. Note that rewinding again and again back to round 2 does not affect either the extraction or Server's strategy, since Server's strategy is fixed at the outset of the overall protocol and s^*/h^* is used nowhere else in the protocol.

The success of this strategy implies that there exists an expected polynomial time algorithm, namely $\mathcal{S}_S$ running π^M_{5PM} against a Server, that is able to extract a discrete logarithm s^* in polynomial time. Therefore, the discrete logarithm problem is solvable in polynomial time, which is a contradiction. Accordingly, it must not be the case that Server can generate s^* independently, which completes the proof. □

References

[1] A. V. Aho and M. J. Corasick. Efficient string matching: an aid to bibliographic search. *Commun. ACM*, 18(6):333–340, June 1975.

[2] S. Al-Khalifa, H. V. Jagadish, J. M. Patel, Y. Wu, N. Koudas, and D. Srivastava. Structural joins: A primitive for efficient xml query pattern matching. In *ICDE'02*, pages 141–152, 2002.

[3] M. Alekhnovich. More on average case vs approximation complexity. volume 20, pages 755–786, 2011.

[4] M. Alekhnovich, E. A. Hirsch, and D. Itsykson. Exponential lower bounds for the running time of dpll algorithms on satisfiable formulas. *J. Autom. Reason.*, 35(1-3):51–72, Oct. 2005.

[5] B. Applebaum. Pseudorandom generators with long stretch and low locality from random local one-way functions. In *Proceedings of the 44th symposium on Theory of Computing*, STOC '12, pages 805–816, New York, NY, USA, 2012. ACM.

[6] B. Applebaum, Y. Ishai, and E. Kushilevitz. Cryptography in nc^0. *SIAM J. Comput.*, 36(4):845–888, 2006.

[7] B. Applebaum, Y. Ishai, and E. Kushilevitz. On pseudorandom generators with linear stretch in nc0. *Comput. Complex.*, 17(1):38–69, Apr. 2008.

[8] P. Baldi, R. Baronio, E. De Cristofaro, P. Gasti, and G. Tsudik. Countering gattaca: efficient and secure testing of fully-sequenced human genomes. In *CCS'11*, pages 691–702. ACM, 2011.

[9] B. Barak, O. Goldreich, S. Goldwasser, and Y. Lindell. Resettably-sound zero-knowledge and its applications. In *FOCS 2001*.

[10] E. Ben-Sasson, O. Goldreich, P. Harsha, M. Sudan, and S. P. Vadhan. Robust PCPs of proximity, shorter PCPs, and applications to coding. *SIAM J. Comput.*, 36(4):889–974, 2006.

[11] D. Betel and C. Hogue. Kangaroo - a pattern-matching program for biological sequences. *BMC Bioinformatics*, 3(1):20, 2002.

[12] C. Blundo, G. Persiano, A.-R. Sadeghi, and I. Visconti. Improved security notions and protocols for non-transferable identification. In S. Jajodia and J. Lopez, editors, *ESORICS '08*, volume 5283 of *LNCS*, pages 364–378. Springer Berlin / Heidelberg, 2008.

[13] A. Bogdanov and Y. Qiao. On the security of goldreich's one-way function. *Comput. Complex.*, 21(1):83–127, Mar. 2012.

[14] F. Brandt. Efficient cryptographic protocol design based on distributed el gamal encryption. In D. Won and S. Kim, editors, *ICISC'05*, volume 3935 of *LNCS*, pages 32–47. Springer Berlin / Heidelberg, 2006.

[15] R. Canetti, Y. Dodis, S. Halevi, E. Kushilevitz, and A. Sahai. Exposure-resilient functions and all-or-nothing transforms. In *Proceedings of the 19th international conference on Theory and application of cryptographic techniques*, EUROCRYPT'00, pages 453–469, Berlin, Heidelberg, 2000. Springer-Verlag.

[16] R. Canetti, O. Goldreich, S. Goldwasser, and S. Micali. Resettable zero-knowledge (extended abstract). In *STOC '00*.

[17] R. Canetti, J. Kilian, E. Petrank, and A. Rosen. Black-box concurrent zero-knowledge requires $\tilde{\Omega}(\log n)$ rounds. In *STOC '01*.

[18] C. Cho, R. Ostrovsky, A. Scafuro, and I. Visconti. Simultaneously resettable arguments of knowledge. In R. Cramer, editor, *TCC'12*, volume 7194 of *LNCS*, pages 530–547. Springer Berlin / Heidelberg, 2012.

[19] J. Cook, O. Etesami, R. Miller, and L. Trevisan. Goldreich's one-way function candidate and myopic backtracking algorithms. In *Proceedings of the 6th Theory of Cryptography Conference on Theory of Cryptography*, TCC '09, pages 521–538, Berlin, Heidelberg, 2009. Springer-Verlag.

[20] R. Cramer, I. Damgrd, and B. Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In Y. Desmedt, editor, *CRYPTO'94*, volume 839 of *LNCS*, pages 174–187. Springer Berlin / Heidelberg, 1994.

[21] R. Cramer, R. Gennaro, and B. Schoenmakers. A secure and optimally efficient multiauthority election scheme. In W. Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 103–118. Springer Berlin / Heidelberg, 1997.

[22] G. Crescenzo, G. Persiano, and I. Visconti. Constant-round resettable zero knowledge with concurrent soundness in the bare public-key model. In M. Franklin, editor, *CRYPTO'04*, volume 3152 of *LNCS*, pages 237–253. Springer Berlin/Heidelberg, 2004.

[23] G. Crescenzo, G. Persiano, and I. Visconti. Improved setup assumptions for 3-round resettable zero knowledge. In P. Lee, editor, *ASIACRYPT'04*, volume 3329 of *LNCS*, pages 530–544. Springer Berlin/Heidelberg, 2004.

[24] I. Damgård. On Σ protocols. `www.daimi.au.dk/~ivan/Sigma.pdf`.

[25] I. Damgrd and C. Orlandi. Multiparty computation for dishonest majority: From passive to active security at low cost. In T. Rabin, editor, *CRYPTO'10*, volume 6223 of *LNCS*, pages 558–576. Springer Berlin / Heidelberg, 2010.

[26] Y. Deng, V. Goyal, and A. Sahai. Resolving the simultaneous resettability conjecture and a new non-black-box simulation strategy. In *FOCS '09*.

[27] Y. Deng and D. Lin. Instance-dependent verifiable random functions and their application to simultaneous resettability. In *EUROCRYPT '07*.

[28] Y. Desmedt and Y. Frankel. Threshold cryptosystems. In G. Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 307–315. Springer Berlin / Heidelberg, 1990.

[29] C. Dwork and M. Naor. Zaps and their applications. In *FOCS '00*.

[30] C. Dwork, M. Naor, and A. Sahai. Concurrent zero-knowledge. In *STOC '98*.

[31] T. El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Proceedings of CRYPTO 84 on Advances in cryptology*, pages 10–18, New York, NY, USA, 1985. Springer-Verlag New York, Inc.

[32] M. Freedman, Y. Ishai, B. Pinkas, and O. Reingold. Keyword search and oblivious pseudorandom functions. In J. Kilian, editor, *TCC'05*, volume 3378 of *LNCS*, pages 303–324. Springer Berlin / Heidelberg, 2005.

[33] K. Frikken. Practical private dna string searching and matching through efficient oblivious automata evaluation. In E. Gudes and J. Vaidya, editors, *Data and Applications Security XXIII*, volume 5645 of *LNCS*, pages 81–94. Springer Berlin / Heidelberg, 2009.

[34] S. Garg, R. Ostrovsky, I. Visconti, and A. Wadia. Resettable statistical zero knowledge. In R. Cramer, editor, *TCC'12*, volume 7194 of *LNCS*, pages 494–511. Springer Berlin/Heidelberg, 2012.

[35] R. Gennaro, C. Hazay, and J. Sorensen. Text search protocols with simulation based security. In P. Nguyen and D. Pointcheval, editors, *PKC'10*, volume 6056 of *LNCS*, pages 332–350. Springer Berlin / Heidelberg, 2010.

[36] C. Gentry. Fully homomorphic encryption using ideal lattices. In *STOC '09*, pages 169–178. ACM, 2009.

[37] O. Goldreich. *Foundations of Cryptography: Basic Tools*. Cambridge University Press, New York, NY, USA, 2000.

[38] O. Goldreich. Candidate one-way functions based on expander graphs. In *Studies in Complexity and Cryptography*, pages 76–87. 2011.

[39] O. Goldreich, H. Krawczyk, and M. Luby. On the existence of pseudorandom generators. *SIAM J. Comput.*, 22(6):1163–1175, 1993.

[40] O. Goldreich and L. A. Levin. A hard-core predicate for all one-way functions. In *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, STOC '89, pages 25–32, New York, NY, USA, 1989. ACM.

[41] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *STOC '87*, pages 218–229. ACM, 1987.

[42] I. Haitner. *New Implications and Improved Efficiency of Constructions Based on One-way Functions*. PhD thesis, Wiezmann Institute of Science.

[43] I. Haitner, D. Harnik, and O. Reingold. Efficient pseudorandom generators from exponentially hard one-way functions. In *Proceedings of the 33rd international conference on Automata, Languages and Programming - Volume Part II*, ICALP'06, pages 228–239, Berlin, Heidelberg, 2006. Springer-Verlag.

[44] I. Haitner, D. Harnik, and O. Reingold. On the power of the randomized iterate. In *Proceedings of the 26th annual international conference on Advances in Cryptology*, CRYPTO'06, pages 22–40, Berlin, Heidelberg, 2006. Springer-Verlag.

[45] I. Haitner, D. Harnik, and O. Reingold. On the power of the randomized iterate. *SIAM J. Comput.*, 40(6):1486–1528, 2011.

[46] I. Haitner, O. Reingold, and S. Vadhan. Efficiency improvements in constructing pseudorandom generators from one-way functions. In *Proceedings of the 42nd ACM symposium on Theory of computing*, STOC '10, pages 437–446, New York, NY, USA, 2010. ACM.

[47] J. Hastad, R. Impagliazzo, L. A. Levin, and M. Luby. A pseudorandom generator from any one-way function. *SIAM J. Comput.*, 28(4):1364–1396, Mar. 1999.

[48] C. Hazay and Y. Lindell. Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. In R. Canetti, editor, *TCC'08*, volume 4948 of *LNCS*, pages 155–175. Springer Berlin / Heidelberg, 2008.

[49] C. Hazay and T. Toft. Computationally secure pattern matching in the presence of malicious adversaries. In M. Abe, editor, *ASIACRYPT'10*, volume 6477 of *LNCS*, pages 195–212. Springer Berlin / Heidelberg, 2010.

[50] H. Hoffmann, M. D. Howard, and M. J. Daily. Fast pattern matching with time-delay neural networks. In *IJCNN'11*, pages 2424–2429, 2011.

[51] T. Holenstein. Pseudorandom generators from one-way functions: a simple construction for any hardness. In *Proceedings of the Third conference on Theory of Cryptography*, TCC'06, pages 443–461, Berlin, Heidelberg, 2006. Springer-Verlag.

[52] T. Holenstein, U. M. Maurer, and J. Sjödin. Complete classification of bilinear hard-core functions. In *CRYPTO'04*, pages 73–91, 2004.

[53] S. Hoory, N. Linial, A. Wigderson, and A. Overview. Expander graphs and their applications. *Bull. Amer. Math. Soc. (N.S*, 43:439–561, 2006.

[54] R. Impagliazzo, L. A. Levin, and M. Luby. Pseudo-random generation from one-way functions (extended abstracts). In *STOC*, pages 12–24, 1989.

[55] Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai. Cryptography with constant computational overhead. In *Proceedings of the 40th annual ACM symposium on Theory of computing*, STOC '08, pages 433–442, New York, NY, USA, 2008. ACM.

[56] Y. Ishai, M. Prabhakaran, and A. Sahai. Founding cryptography on oblivious transfer efficiently. In D. Wagner, editor, *CRYPTO'08*, volume 5157 of *LNCS*, pages 572–591. Springer Berlin / Heidelberg, 2008.

[57] A. Jarrous and B. Pinkas. Secure hamming distance based computation and its applications. In M. Abdalla, D. Pointcheval, P.-A. Fouque, and D. Vergnaud, editors, *ACNS'09*, volume 5536 of *LNCS*, pages 107–124. Springer Berlin / Heidelberg.

[58] R. M. Karp and M. O. Rabin. Efficient randomized pattern-matching algorithms. *IBM J. Res. Dev.*, 31:249–260, March 1987.

[59] J. Katz and L. Malka. Secure text processing with applications to private DNA matching. In *CCS '10*, pages 485–492. ACM, 2010.

[60] J. Katz and R. Ostrovsky. Round-optimal secure two-party computation. In *CRYPTO 2004*, pages 335–354. Springer-Verlag, 2004.

[61] J. Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *STOC '92*, pages 723–732. ACM, 1992.

[62] J. Kilian and E. Petrank. Concurrent and resettable zero-knowledge in poly-logarithmic rounds. In *STOC '01*.

[63] E. Kiltz. A primitive for proving the security of every bit and about universal hash functions & hard core bits. In *Proceedings of the 13th International Symposium on Fundamentals of Computation Theory*, FCT '01, pages 388–391, London, UK, UK, 2001. Springer-Verlag.

[64] D. E. Knuth, J. H. M. Jr., and V. R. Pratt. Fast pattern matching in strings. *SIAM J. Comput.*, 6(2):323–350, 1977.

[65] Y. Lindell and B. Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In *Proceedings of the 26th Annual International Conference on Advances in Cryptology*, EUROCRYPT '07, pages 52–78, Berlin, Heidelberg, 2007. Springer-Verlag.

[66] S. Micali. Cs proofs. In *FOCS'94*, pages 436–453. IEEE Computer Society, 1994.

[67] S. Micali and L. Reyzin. Soundness in the public-key model. In *CRYPTO '01*.

[68] D. Micciancio and E. Petrank. Simulatable commitments and efficient concurrent zero-knowledge. In *EUROCRYPT '03*.

[69] P. Mohassel, S. Niksefat, S. Sadeghian, and B. Sadeghiyan. An efficient protocol for oblivious dfa evaluation and applications, 2012.

[70] K. Namjoshi and G. Narlikar. Robust and fast pattern matching for intrusion detection. In *INFOCOM'10*, pages 740–748. IEEE Press, 2010.

[71] M. Näslund. Universal hash functions & hard core bits. In *Proceedings of the 14th annual international conference on Theory and application of cryptographic techniques*, EUROCRYPT'95, pages 356–366, Berlin, Heidelberg, 1995. Springer-Verlag.

[72] M. Osadchy, B. Pinkas, A. Jarrous, and B. Moskovich. Scifi - a system for secure face identification. In *IEEE S&P '10*, pages 239–254. IEEE Computer Society, 2010.

[73] R. Ostrovsky, O. Pandey, and I. Visconti. Efficiency preserving transformations for concurrent non-malleable zero knowledge. In D. Micciancio, editor, *TCC'10*, volume 5978 of *LNCS*, pages 535–552. Springer Berlin / Heidelberg, 2010.

[74] S. K. Panjwani. An experimental evaluation of goldreich's one-way function. Technical report, IIT, Bombay, 2001.

[75] T. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In J. Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 129–140. Springer Berlin / Heidelberg, 1992.

[76] M. Prabhakaran, A. Rosen, and A. Sahai. Concurrent zero knowledge with logarithmic round-complexity. In *FOCS '02*.

[77] R. Richardson and J. Kilian. On the concurrent composition of zero-knowledge proofs. In *EUROCRYPT '99*.

[78] A. Scafuro and I. Visconti. On round-optimal zero knowledge in the bare public-key model. In D. Pointcheval and T. Johansson, editors, *EUROCRYPT'12*, volume 7237 of *LNCS*, pages 153–171. Springer Berlin/Heidelberg, 2012.

[79] C.-P. Schnorr. Efficient identification and signatures for smart cards. In *Proceedings of the 9th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '89, pages 239–252, London, UK, UK, 1990. Springer-Verlag.

[80] J. R. Troncoso-Pastoriza, S. Katzenbeisser, and M. Celik. Privacy preserving error resilient DNA searching through oblivious automata. In *CCS'07*, pages 519–528. ACM, 2007.

[81] T.-H. Tsai. Average case analysis of the Boyer-Moore algorithm. *Random Struct. Algorithms*, 28:481–498, July 2006.

[82] A. Tumeo and O. Villa. Accelerating dna analysis applications on gpu clusters. In *SASP '10*, pages 71–76. IEEE Computer Society, 2010.

[83] S. Vadhan and C. J. Zheng. Characterizing pseudoentropy and simplifying pseudorandom generator constructions. In *Proceedings of the 44th symposium on Theory of Computing*, STOC '12, pages 817–836, New York, NY, USA, 2012. ACM.

[84] D. Vergnaud. Efficient and secure generalized pattern matching via fast fourier transform. In A. Nitaj and D. Pointcheval, editors, *AFRICACRYPT '11*, volume 6737 of *LNCS*, pages 41–58. Springer Berlin / Heidelberg, 2011.

[85] A. C. Yao. Theory and application of trapdoor functions. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, SFCS '82, pages 80–91, Washington, DC, USA, 1982. IEEE Computer Society.

[86] A. C.-C. Yao. How to generate and exchange secrets. In *FOCS'86*, pages 162–167. IEEE Computer Society, 1986.