EXPLORING THE COMPOSITION OF CODING THEORY AND CRYPTOGRAPHY THROUGH SECURE COMPUTATION, SUCCINCT ARGUMENTS, AND LOCAL CODES

by

Alexander R. Block

A Dissertation

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the degree of

Doctor of Philosophy

Department of Computer Science

West Lafayette, Indiana

August 2022

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

Dr. Jeremiah Blocki, Chair

Department of Computer Science, Purdue University

Dr. Mikhail Atallah

Department of Computer Science, Purdue University

Dr. Christina Garman

Department of Computer Science, Purdue University

Dr. Elena Grigorescu

Department of Computer Science, Purdue University

Dr. Ron Rothblum

Department of Computer Science, Technion

Dr. David Gleich, External

Department of Computer Science, Purdue University

Approved by:

Dr. Kihong Park

To my family, friends, loved ones, colleagues,
and my cat Groot for their unwavering support.

ACKNOWLEDGMENTS

The author fully acknowledges and is extremely thankful for the support of his advisor, committee members, colleagues, family, friends, and loved ones. Without these people, this dissertation would not have come to fruition.

The research presented in this dissertation was generously supported by a variety of sources, including NSF CRII Award CNS-1566499, NSF SMALL Award CNS-1618822, REU CNS-1724673, and NSF grant CCF-1910659. Some of this research was also generously supported by the FACT Research Center at Reichmann University (IDC Herzliya), Israel.

PREFACE

Coding theory and cryptography (in its modern form) were conceived by Claude Shannon in the late 1940's [238, 239], enjoying a deep and interconnected history since their conception. Constructions of both coding-theoretic objects and cryptographic objects are often inspired by, make explicit use of, or make implicit use of each other. Both fields continue to have a mutually beneficial relationship. This is particularly true in my various research projects, which has inspired me to write this dissertation. The goal of this dissertation is to examine new ways in which coding theory and cryptography compose with each other—both explicitly and implicitly. Towards this goal, in this dissertation I examine the bulk of my prior research, listed here [13, 45, 46, 48, 49, 51, 52]. Many proofs and preliminary sections are taken from my prior works (more or less) verbatim. However, much of my introductory treatment of these works has a shifted focus towards their relations with coding theory or cryptography, and, as such, differ from the original works in these regards. All copyrights for these respective works belong to their respective publishers.

I would also like to specifically acknowledge the works of [48, 49]. For the work of [49], I have received permission from fellow co-author and student Hai Nguyen to use this work as part of my dissertation, and acknowledge that he may use this work as part of his dissertation as well. For the work of [48], I have received permission from fellow co-author and student Minshen Zhu to use this work as part of my dissertation, and acknowledge that he may use this work as part of his dissertation as well.

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

We examine new ways in which coding theory and cryptography continue to be composed together, and show that the composition of these two fields yield new constructions in the areas of Secure Computation Protocols, Succinct Interactive Arguments, and Locally Decodable Codes. This dissertation is a continuation of several decades of research in composing coding theory and cryptography; examples include secret sharing, encryption schemes, randomness extraction, pseudo-random number generation, and the PCP theorem, to name a few.

In Part I of this dissertation, we examine the composition of coding theory with cryptography, explicitly and implicitly. On the explicit side, we construct a new family of linear error-correcting codes, based on algebraic geometric codes, and use this family to construct new *correlation extractors* (Ishai et al., FOCS 2009). Correlation extractors are two-party secure computation protocols for distilling samples of a leaky correlation (e.g., pre-processed secret shares that have been exposed to side-channel attacks) into secure and fresh shares of another correlation (e.g., shares of oblivious transfer). Our correlation extractors are (nearly) optimal in all parameters. On the implicit side, we use coding theoretic arguments to show the security of *succinct interactive arguments* (Micali, FOCS 1994). Succinct interactive arguments are a restriction of interactive proofs (Goldwasser, Micali, Rackoff, STOC 1985) for which security only holds against computationally bounded provers (i.e., probabilistic polynomial time), and where the proofs are sub-linear in the size of the statement being proven. Our new succinct interactive arguments are the first public-coin, zero-knowledge arguments with time *and* space efficient provers: we give two protocols where any NP statement that is verifiable by a time-T space-S RAM program in is provable time $\widetilde{O}(T)$ and space $S \cdot \mathrm{polylog}(T)$.

In Part II of this dissertation, we examine the composition of cryptography with coding theory, again explicitly and implicitly, focusing specifically on *locally decodable codes* (Katz and Trevisan, STOC 2000). Locally decodable codes, or LDCs, are error-correcting codes with super-efficient probabilistic decoding procedures that allow for decoding individual symbols of the encoded message, without decoding the entire codeword. On the implicit side, we utilize cryptographic analysis tools to give a conceptually simpler proof of the so-called

"Hamming-to-InsDel" compiler (Ostrovsky and Paskin-Cherniavsky, ITS 2015). This compiler transforms any Hamming LDC (i.e., a code that is resilient to bit-flip errors) to another LDC that is resilient to the broad class of insertion-deletion errors, approximately preserving the rate and error-tolerance of the code at the cost of a poly-logarithmic increase in the query complexity. We further extend this compiler to both the *private LDC* setting (Ostrovsky, Pandey, and Sahai, ICALP 2007), where the encoder and decoder are assumed to share a secret key unknown to the adversarial channel, and the *resource-bounded LDC* setting (Blocki, Kulkarni, and Zhou, ITC 2020), where the adversarial channel is assumed to be resource constrained. On the explicit side, we utilize two cryptographic primitives to give new constructions of alternative notions of LDCs. First, we use *cryptographic puzzles* (Bitansky et al., ITCS 2016) to construct resource-bounded Hamming LDCs in the standard model without random oracles, answering an open question of Blocki, Kulkarni, and Zhou (ITC 2020); we then naturally extend these LDCs to the InsDel setting via our previously mentioned compiler. Second, we use digital signature schemes to directly construct *computationally relaxed LDCs* (Blocki et al., ITIT 2021) that are resilient to both Hamming errors and insertion-deletion errors. Computationally relaxed LDCs allow the decoder to output an extra symbol signifying it does not know the correct output and are only secure against probabilistic polynomial time adversarial channels. Our construction is conceptually simpler than the construction of Blocki et al. (ITIT 2021) and does not require a trusted setup for seeding a collision-resistant hash function.

1. INTRODUCTION

Coding Theory and Cryptography are two subsets of computer science that seek to understand different aspects of the fundamental problem of (electronic) communication through various communication channels. Coding theory seeks to ensure that messages can be *encoded* in some manner and later *decoded* correctly in a variety of scenarios. For example, one goal is to encode messages such that even if random and/or adversarial errors are introduced, one can still decode the original message, so long as a bounded number of errors occur (i.e., *error-correction*); another goal is to encode messages as smaller messages for storage and then expand to the original message as needed (i.e., *data compression*). Generally speaking, *Hamming errors*, or bit-flip errors, are most often considered in the context of error-correction. In contrast (though not necessarily mutually exclusive from coding theory), cryptography at its core seeks to ensure that messages are kept *private* from unwanted parties. For example, two parties may wish to secretly communicate without any other parties knowing the contents of their communication (i.e., *encryption*). Fundamentally, both fields handle the transmission of data across communication channels where different properties are desirable, such as worst-case error-tolerance or privacy of the communication.

Both coding theory and cryptography have long and studied histories. Cryptography arguably has existed since the times of the ancient Greeks in the form of simple ciphers—encoding schemes that, essentially, "shuffle" letters according to some pattern(s) (i.e., according to the cipher)—with the oldest known recorded cipher, the Caesar Cipher, dating back to 110 CE. More formal treatment of cryptography can be seen towards the end of the 19th century with Kerckhoffs [171, 172], and the formal introduction of the one-time pad by Vernam in 1917 [253] (first described by Miller in 1882 [26]). Many attribute Claude Shannon as the founder of both coding theory [239] and modern cryptography [238]. Both of Shannon's works have served as the basis for both coding theory and modern cryptography to this day, introducing rigorous definitions and problems to both fields. Since their inception, both areas continue to expand and evolve to tackle new, challenging, and exciting problems beyond what were originally studied.

Unsurprisingly, coding theory and cryptography evolved together over the years. Even at their inception, Shannon, with his introduction of *information-theoretic security* (i.e., security in the presence of computationally unbounded adversaries), proved that the one-time pad was a perfectly secure (one-time) communication method [238, 239]. Ever since, coding theory and cryptography continue to mutually benefit each other through composition, both explicitly and implicitly. Coding theory explicitly composed in cryptography has an innumerable number of success stories, including secret-sharing [44, 198, 202, 237], digital signature schemes [18, 19, 96, 98, 113, 212, 224], public-key cryptography [201, 212], succinct (non-)interactive arguments [125, 175, 206], randomness extraction [1, 147, 235] and pseudo-random number generation [163, 235, 240, 246, 249, 250], resilient functions [38, 93], and secure multi-party computation [129, 174, 176, 263]. Through implicit compositions, coding theoretic arguments appear in many results that are related to the seminal PCP theorem [15, 16, 22, 114, 216] such as (multi-prover) interactive proofs [20, 27], succinct (non-)interactive arguments [175, 206], and interactive oracle proofs [35, 226], to name a few. The PCP theorem itself can be viewed as leveraging properties of locally decodable codes, which themselves have strong cryptographic influences [21, 22, 63, 128, 189, 192, 236, 257].

The composition of cryptography in coding theory has many success stories as well. Locally decodable codes are a prominent example of this strong connection. Such codes have strong cryptographic motivations and influences, such as hard-core bits [128], program testing [22, 63, 189], arithmetization [21, 22, 192, 236], and private information retrieval [12, 25, 92, 127, 170, 173]. For example, the proof of the Goldreich-Levin Theorem [128] immediately yields a local list decoder for the Walsh-Hadamard code [17]. Various alternative models for locally decodable codes (and general error-correcting codes) with roots in cryptography have been studied, such as private-key codes [46, 214], public-key codes [155, 156], and codes secure against computationally bounded adversaries [60, 146, 188, 207, 234], all composing cryptography in coding theory explicitly. Similarly, the implicit use of cryptography is rampant throughout coding theory (and information theory in general) in the form of cryptographic (or adversarial) thinking. For example, constructing codes that protect against adversarial errors versus random errors [239], and the use of cryptographic thinking in various analyses [46, 48, 60, 146, 188, 207, 214, 234].

1.1 Our Contribution

In this dissertation, we examine new ways in which coding theory and cryptography continue to be composed together, and show that the composition of these two fields yield new constructions in the areas of Secure Computation Protocols, Succinct Interactive Arguments, and Locally Decodable Codes. Our examination falls under two categories. The first category (Part I) examines the composition coding theory in cryptography, explicitly and implicitly. On the explicit side, in Chapter 3 we study how coding theory is explicitly used in the context of *secure two-party computation* [129, 174, 176]—one of the hardest cases of secure multi-party computation since there is an absence of any honest majority. Coding theory is often used to great success in secure multi-party computation, and secure two-party computation is no exception. On the implicit side, in Chapter 4 we study how coding theory is used implicitly in the context of *interactive arguments* [125, 175, 206]—a relaxation of interactive proofs [132] where the (adversarial) prover is assumed to be computationally bounded (i.e., probabilistic polynomial time). Here, it is often the case that the security of the argument is reduced (in some way) to the PCP theorem, and hence relies implicitly on properties of the underlying code representing the PCP proof string.

The second category (Part II) examines the composition of cryptography in coding theory, implicitly and explicitly. On the implicit side, in Chapter 5 we study how cryptographic thinking simplifies the analysis of a so-called "Hamming-to-InsDel" compiler, which compiles any (private, resource-bounded) locally decodable code that is secure against Hamming errors to a locally decodable code secure against *insertion-deletion errors*; such adversarial error patterns introduce arbitrary insertions, deletions, and transpositions of symbols in the encoded message. On the explicit side, we examine two results. In Chapter 6, we study how a cryptographic primitive known as a cryptographic puzzle [43] is explicitly utilized to construct the notion of resource-bounded locally decodable codes [60] under standard cryptographic assumptions. In Chapter 7, we study how digital signature schemes can be explicitly utilized to construct computationally relaxed locally decodable codes [57] that are resilient to Hamming errors and resilient to insertion-deletion errors *without* relying on the compiler of Chapter 5.

2. GENERAL PRELIMINARIES

We give general preliminaries that are relevant to all aspects of this dissertation. In later chapters, we introduce additional preliminaries as necessary whenever they are needed.

2.1 General Notation

We let "$\circ$" denote the string concatenation operator and let ϵ denote the empty string unless otherwise stated. That is, for any string s, we have $s \circ \epsilon = \epsilon \circ s = s$. We let $\mathbb{N} = \{0, 1, 2, \dots\}$ denote the set of non-negative integers, $\mathbb{Z}$ denote the set of all integers, and $\mathbb{Z}^+ = \{1, 2, 3, \dots\}$ denote the set of positive integers. We also let $\mathbb{R}$ denote the set of all real numbers and $\mathbb{R}_{\geqslant 0}$ denote the set of non-negative real numbers. For positive integer $N \in \mathbb{Z}^+$, we let $\mathbb{Z}_N := \mathbb{Z}/(N\mathbb{Z})$ be the set of integers modulo N. We also let $\mathbb{Z}_N^\times$ denote the set elements of $\mathbb{Z}_N$ with multiplicative inverses modulo N. For any $n \in \mathbb{Z}^+$, we let $[n] := \{1, 2, \dots, n\}$ denote the set of integers from 1 to n. A function $\vartheta \colon \mathbb{N} \to \mathbb{R}_{\geqslant 0}$ is said to be *negligible* if $\vartheta(n) = o(1/|p(n)|)$ for any fixed non-zero polynomial p. Unless otherwise stated, we let $\log := \log_2$.

For a finite, non-empty set S, we let $x \xleftarrow{\$} S$ denote the process of sampling an element x uniformly at random from S. We also let U_S denote the uniform distribution over a set S; in particular, $x \leftarrow U_S$ is identically distributed to $x \xleftarrow{\$} S$. If S is a distribution or randomized process, we let $x \leftarrow S$ denote the process of sampling x according to the distribution or randomized process S. For any $N \in \mathbb{N}$, we let S^N denote the set of all sequences/vectors $\{(s_1, \dots, s_N) \colon s_i \in S\}$ of length N containing elements of S, and by convention define $S^0 := \{\epsilon\}$. We also let S^* denote the set of all sequences of S of any length. As usual, we define $\sum_{i=j}^{k} a_i = 0$ and $\prod_{i=j}^{k} a_i = 1$ whenever $j > k$.

For simplicity, unless otherwise stated, we assume that any vector is a row vector; i.e., $b \in \{0,1\}^n$ is a row vector for $n \in \mathbb{N}$. For $n \in \mathbb{N}$, we let 0^n and 1^n denote the all-zero and all-one vectors of length n, respectively. For any $x \in S^n$ for set S, we borrow from array notation and let $x[i]$ denote the i^{th} element of x; we also use x_i to denote the i^{th} element of x. For $i < j$, we let $x[i, j] := (x_i, x_{i+1}, \dots, x_j)$ and reserve the notation $x(i, j)$ to denote $x_{i,j}$ whenever x is a matrix.

For any set S for which addition and multiplication are defined and vectors $x, y \in S^n$ for $n \in \mathbb{Z}^+$, we let $\langle x, y \rangle := \sum_i x_i \cdot y_i \in S$ denote the inner product of x and y. We also let $x * y \in S^n$ denote the coordinate-wise product of x and y; that is, $(x * y)_i = x_i \cdot y_i$ for all i. For $n \in \mathbb{Z}^+$, we let $\mathcal{S}_n$ denote the set of all permutations $\pi \colon [n] \to [n]$ on the set $[n]$. Furthermore, for any vector $x = (x_1, \ldots, x_n)$, we define $\pi(x) := (x_{\pi(1)}, \ldots, x_{\pi(n)})$. For any set S containing the element 0, we let $\mathsf{wt}(x)$ denote the number of non-zero entries of $x \in S^n$. Formally, $\mathsf{wt}(x) := \{i \in [n] : x_i \neq 0\}$.

We write PPT as a shorthand for probabilistic polynomial time. For any (randomized) algorithm A, we let $y \leftarrow A(x)$ denote the process of running A on some input x and storing the output in y; when A is randomized, then $A(x)$ denotes a distribution and $y \leftarrow A(x)$ denotes sampling y via the distribution $A(x)$. For any two distributions X and Y over the same finite sample space Ω, we define the *statistical distance* between X and Y as $\mathrm{SD}(X, Y) := \frac{1}{2} \cdot \sum_{\omega \in \Omega} |X(\omega) - Y(\omega)|$.

2.2 Group and Field Theory Preliminaries

We borrow much of this section from [169]. We assume basic knowledge of group and field theory and state some definitions and theorems here for completeness. We begin with the definition of a group.

Definition 2.2.1. *A* group *is a set $\mathbb{G}$ along with a binary operation $\diamond(\cdot, \cdot)$, which we denote as $(\mathbb{G}, \diamond)$ or just $\mathbb{G}$ when $\diamond$ is clear from context, for which the following hold:*

- *(Closure) For all $g, h \in \mathbb{G}$, $g \diamond h \in \mathbb{G}$ and $h \diamond g \in \mathbb{G}$.*
- *(Identity) There exists an identity element $1_{\mathbb{G}} \in \mathbb{G}$ such that for all $g \in \mathbb{G}$, we have $1_{\mathbb{G}} \diamond g = g \diamond 1_{\mathbb{G}} = g$.*
- *(Inverses) For all $g \in \mathbb{G}$, there exists $h \in \mathbb{G}$ such that $g \diamond h = h \diamond g = 1_{\mathbb{G}}$. Here, h is called the* inverse *of g.*
- *(Associativity) For all $g_1, g_2, g_3 \in \mathbb{G}$, we have $(g_1 \diamond g_2) \diamond g_3 = g_1 \diamond (g_2 \diamond g_3)$.*

When $\mathbb{G}$ has a finite number of elements, we say that $\mathbb{G}$ is finite *and let $|\mathbb{G}|$ denote the order of the group (i.e., the number of elements of $\mathbb{G}$). We say that $\mathbb{G}$ is* abelian *if for all $g, h \in \mathbb{G}$, we have $g \diamond h = h \diamond g$.*

Note that $(\mathbb{Z}, +)$ and $(\mathbb{R}, +)$ are abelian groups of infinite order, where "+" denote standard addition. Most often, we assume a group $\mathbb{G}$ is a finite abelian group with multiplication as the group operation; e.g., $\mathbb{G} = \mathbb{Z}_N^\times$ for some positive integer N.

We now define fields.

Definition 2.2.2. *A* field *is a set $\mathbb{F}$ along with two binary operations $+(\cdot,\cdot)$ and $\cdot(\cdot,\cdot)$ with the following properties:*

- *$(\mathbb{F}, +)$ is an abelian group with identity 0.*
- *$(\mathbb{F}^\times, \cdot)$ is an abelian group with identity 1, where $\mathbb{F}^\times := \mathbb{F} \setminus \{0\}$. We often write ab in place of $a \cdot b$.*
- *For all $a, b, c \in \mathbb{F}$, we have $a(b + c) = ab + ac$.*

If $\mathbb{F}$ has a finite number of elements then we say $\mathbb{F}$ is a finite field*, and let $|\mathbb{F}|$ denote the* order *(i.e., number of elements) of $\mathbb{F}$.*

Note that $(\mathbb{R}, +, \cdot)$ is a field of infinite order, as well as $(\mathbb{C}, +, \cdot)$, where $\mathbb{C}$ here denotes the set of complex numbers.

Remark 2.2.1. *In Section 2.4 and Chapter 3, we let $\mathbb{C}$ denote the set of complex numbers. In Part II, we let $\mathbb{C}$ denote some class of algorithms, and not the set of complex numbers.*

It is a well-known and used result from number theory that the order of any finite field is a power of a prime number.

Theorem 2.2.2. *If $\mathbb{F}$ is a finite field, then $|\mathbb{F}| = p^a$ for some prime number $p \in \mathbb{Z}^+$ and positive integer $a \in \mathbb{Z}^+$. Conversely, for every $q = p^a$ for some prime number p and positive integer a, there exists a unique (up to relabeling of elements) finite field of order q.*

We most often deal with the binary field $\mathbb{F}_2 = \mathbb{Z}_2$ of integers modulo 2, and let $\{0,1\}^n := \mathbb{F}_2^n$ denote the n-dimensional vector space over $\mathbb{F}_2$. We conclude by defining *extension fields*.

Definition 2.2.3. *For two fields $\mathbb{F}$ and $\mathbb{K}$, we say that $\mathbb{K}$ is an* extension field *of $\mathbb{F}$ if $\mathbb{F}$ is a sub-field of $\mathbb{K}$; i.e., there exists a subset $S \subset \mathbb{K}$ that is isomorphic to $\mathbb{F}$ and closed under addition and multiplication.*

It is well-known that for a finite field $\mathbb{F}$ of size q, any field $\mathbb{K}$ of size q^a for $a \geqslant 1$ is isomorphic to the polynomial ring $\mathbb{F}[X]$ modulo an irreducible polynomial of degree a with coefficients in $\mathbb{F}[X]$.

Corollary 2.2.3. *For any finite field $\mathbb{F}$ of size q and extension field $\mathbb{K}$ of $\mathbb{F}$ of size q^a, there exists an irreducible degree a polynomial $p(X)$ over $\mathbb{F}[X]$ such that $\mathbb{K} \cong \mathbb{F}[X]/\langle p(X) \rangle$.*

2.3 Vector Spaces

We borrow much of this section from [136]. We assume basic knowledge about vector spaces and state some definitions and theorems here for completeness. We begin by defining a vector space.

Definition 2.3.1. *A* vector space *V over a finite field $\mathbb{F}$ is an abelian group under the operation "$+$" with an additional scalar product "$\cdot$" satisfying the following properties: for all $\alpha, \beta \in \mathbb{F}$ and $u, v \in V$, we have*

- $\alpha \cdot (\beta \cdot v) = (\alpha \cdot \beta) \cdot v$;
- $(\alpha + \beta) \cdot v = \alpha v + \beta v$;
- $\alpha(u + v) = \alpha u + \alpha v$; *and*
- $1 \cdot v = v$.

The elements of V are called vectors *and the elements of $\mathbb{F}$ are called* scalars. *Moreover, n is called the* dimension *of V.*

Note that "$+$" for vector spaces is coordinate-wise addition. As stated at the start of this chapter, we also consider the coordinate-wise product between vectors, which we denote as $u * v \in V$ for $u, v \in V$. We also define subspaces of a vector space V.

Definition 2.3.2. *Let V be a vector space over $\mathbb{F}$. A subset $W \subseteq V$ is a* subspace *of V if W is closed under both addition and scalar multiplication. That is, for all $u, v \in W$ and $\alpha \in \mathbb{F}$, $u\alpha \in W$ and $u + v \in W$.*

We next work towards defining the dimension of V. We define the span of a set of vectors.

Definition 2.3.3. *Let $S = \{v_1, \ldots, v_n\} \subset V$. Then the* span of S, *denoted as* $\text{span}(S)$, *is defined as*

$$\text{span}(S) := \left\{ \sum_i \alpha_i v_i \colon \alpha_i \in \mathbb{F} \right\} \subset V.$$

It is well-known that the span of a set of vectors is a subspace.

Proposition 2.3.1. *For any $S \subset V$, $\text{span}(S)$ is a subspace of V.*

We now define linear (in)dependence.

Definition 2.3.4. *Let $S = \{v_1, \ldots, v_n\} \subset V$. We say that S is* linearly dependent *if there exist scalars $\alpha_1, \ldots, \alpha_n \in \mathbb{F}$ that are all not zero and $\sum_{i=1}^n \alpha_i \cdot v_i = 0_V$, where $0_V \in V$. If S is not linearly dependent, we say that S is* linearly independent.

The following proposition follows directly from the above definition.

Proposition 2.3.2. *A set $S = \{v_1, \ldots, v_n\} \subset V$ of vectors in linearly dependent if and only if there exists $i \in [n]$ and $\alpha_j \in \mathbb{F}$ for all $j \neq i$ such that $v_i = \sum_{j \neq i} \alpha_j v_j$.*

We now define a basis and the dimension of V.

Definition 2.3.5. *Let $S = \{v_1, \ldots, v_n\} \subset V$. If $\text{span}(S) = V$ and S is linearly independent, then we say S is a* basis *for V. Moreover, we say that the* dimension of V *is n.*

Note that $\mathbb{F}$ itself is a 1-dimensional vector space over $\mathbb{F}$. While V may have many different bases, the dimension of V is unique.

Proposition 2.3.3. *If $\{u_1, \ldots, u_m\}$ and $\{v_1, \ldots, v_n\}$ are both bases for V, then $m = n$.*

It is well-known that n-dimensional vector spaces are isomorphic to $\mathbb{F}^n$.

Theorem 2.3.1. *Any n-dimensional vector space V over $\mathbb{F}$ is isomorphic to $\mathbb{F}^n$.*

We conclude by defining a linear transformation/function over vector spaces.

Definition 2.3.6. *Let V, W be vector spaces over $\mathbb{F}$. A function $T\colon V \to W$ is a* linear transformation/function *if for all $\alpha \in \mathbb{F}$ and $u, v \in V$, we have (1) $T(u+v) = T(u) + T(v)$; and (2) $T(\alpha v) = \alpha T(v)$.*

2.4 Fourier Analysis over Finite Fields

We give some basic Fourier definitions and properties over finite fields. This section is more or less copied verbatim from [49] and follows the conventions of [225].

Let $\mathbb{F}$ be a finite field and let $n \in \mathbb{Z}^+$. We begin by defining the inner product of two complex-valued functions.

Definition 2.4.1. *Let $f, g\colon \mathbb{F}^n \to \mathbb{C}$ be two functions. Then the* inner product *of f and g is defined as*

$$\langle f, g \rangle := \mathop{\mathbb{E}}_{x \xleftarrow{\$} \mathbb{F}^n} \left[f(x) \cdot \overline{g(x)} \right] = \frac{1}{|\mathbb{F}|^n} \cdot \sum_{x \in \mathbb{F}^n} f(x) \cdot \overline{g(x)},$$

where $\overline{g(x)}$ is the complex conjugate of $g(x)$.

Given the inner product of complex functions, we can define character functions of both $\mathbb{F}$ and $\mathbb{F}^n$.

Definition 2.4.2. *Let $\psi\colon \mathbb{F} \to \mathbb{C}^\times$ be a group homomorphism from $(\mathbb{F}, +)$ to $\mathbb{C}^\times := (\mathbb{C}, \cdot)$. Then ψ is a* character function *of $\mathbb{F}$.*

Let $\chi\colon \mathbb{F}^n \times \mathbb{F}^n \to \mathbb{C}^\times$ be a function with the following properties:

- *(Bilinear) For every $X \in \mathbb{F}^n$, $\chi(X, \cdot), \chi(\cdot, X)\colon \mathbb{F}^n \to \mathbb{C}^\times$ are both group homomorphisms.*
- *(Non-degenerate) For every $X \in \mathbb{F}^n$, both $\chi(X, \cdot) \neq 1$ and $\chi(\cdot, X) \neq 1$ (i.e., they are not 1 everywhere).*
- *(Symmetric) For all $X, Y \in \mathbb{F}^n$, $\chi(X, Y) = \chi(Y, X)$.*

Then for any $S \in \mathbb{F}^n$, the function $\chi(S,\cdot) := \chi_S(\cdot)$ is a character function of $\mathbb{F}^n$. In particular, whenever ψ is non-degenerate, then $\chi(S,\cdot) := \psi(\langle S,\cdot\rangle)$ is a character function of $\mathbb{F}^n$ for any fixed $S \in \mathbb{F}^n$.

We can now define the Fourier transformation of a function $f\colon \mathbb{F}^n \to \mathbb{C}$.

Definition 2.4.3 (Fourier Transformation)**.** *Let $f\colon \mathbb{F}^n \to \mathbb{C}$ be a function and let $\chi_S\colon \mathbb{F}^n \to \mathbb{C}^\times$ be a character function for $S \in \mathbb{F}^n$. Let $\widehat{f}\colon \mathbb{F}^n \to \mathbb{C}$ be a function defined as $\widehat{f}(S)\langle f, \chi_S\rangle$. We say that $\widehat{f}(S)$ is the* Fourier Coefficient of f at S*, and that the linear map $f \mapsto \widehat{f}$ is the* Fourier Transformation *of f.*

The Fourier Transformation is an invertible linear map, which is characterized by the following lemma.

Lemma 2.4.1 (Fourier Inversion)**.** *For any $f\colon \mathbb{F}^n \to \mathbb{C}$, we have $f(x) = \sum_{S\in\mathbb{F}^n} \widehat{f}(S)\cdot\chi_S(x)$.*

2.5 Basic Notions of Entropy

Much of this section is more or less copied verbatim from [49]. For a probability distribution X over a sample space U, the *entropy of* $x \in X$ is defined as $H_X(x) = -\log \Pr[X = x]$. The *min-entropy of* X, represented by $\mathbf{H}_\infty(X)$, is defined to be $\min_{x\in\mathsf{Supp}(X)} H_X(x)$. The *binary entropy function*, denoted by $\mathbf{h}_2(x) = -x\log(x) - (1-x)\log(1-x)$ for every $x \in (0,1)$. In general, for any $x \in (0,1)$, we define the *q-entropy function* as $\mathbf{h}_q(x) := -x\log_q(x) - (1-x)\log_q(1-x)$.

Given a joint distribution (X,Y) over sample space $U \times V$, the *marginal distribution* Y is a distribution over sample space V such that, for any $y \in V$, the probability assigned to y is $\sum_{x\in U} \Pr[X = x, Y = y]$. The *conditional distribution* $(X|y)$ represents the distribution over sample space U such that the probability of $x \in U$ is $\Pr[X = x|Y = y]$. The *average min-entropy* [105] is defined as $\widetilde{\mathbf{H}}_\infty(X|Y) := -\log\left(\mathbb{E}_{y\sim Y}[2^{-\mathbf{H}_\infty(X|y)}]\right)$. There is a well-known correspondence between the average min-entropy of a min-entropy distribution with respect to some arbitrary leakage distribution.

Lemma 2.5.1 ([105])**.** *If $\mathbf{H}_\infty(X) \geqslant k$ and L is an arbitrary ℓ-bit leakage on X, then $\widetilde{\mathbf{H}}_\infty(X|L) \geqslant k - \ell$.*

Finally, if X is min-entropy distribution, then we can bound the sum of squares of Fourier coefficients by a factor proportional to the min-entropy.

Lemma 2.5.2 (Fourier Coefficients of a Min-Entropy Distribution)**.** *Let $X\colon \mathbb{F}^\eta \to \mathbb{R}$ be a min-entropy source such that $\mathbf{H}_\infty(X) \geqslant k$. Then $\sum_S |\widehat{X}(S)|^2 \leqslant |\mathbb{F}|^{-\eta} \cdot 2^{-k}$.*

2.6 Small-Bias Sets

Much of this section is again more or less copied verbatim from [49]. Given the definition of a Fourier Coefficient and the Fourier Transformation, we can now define the bias of a distribution. For our purposes, we are only concerned with the bias of distributions defined over $\mathbb{F}^n$, where $\mathbb{F}$ is a finite field and n is some positive integer.

Definition 2.6.1 (Bias of a Distribution)**.** *Let X be a distribution over $\mathbb{F}^n$. Then the* bias of X at $S \in \mathbb{F}^n$ *is defined as* $\mathrm{Bias}(X, S) = \mathrm{Bias}_S(X) := |\mathbb{F}|^n \cdot |\widehat{X}(S)|$.

We can now define a small-biased family of distributions.

Definition 2.6.2 (Small-Bias Distribution Family [106])**.** *A family of distributions $\mathcal{X} = \{X_1, X_2, \ldots, X_k\}$ over sample space $\mathbb{F}^n$ is said to be a* ρ^2-biased family *if for every non-zero vector $S \in \mathbb{F}^n$ we have*

$$\mathop{\mathbb{E}}_{i \xleftarrow{\$} [k]} \mathrm{Bias}_S(X_i)^2 \leqslant \rho^2.$$

It is well-known that small-biased families of distributions can be used to extract almost uniform randomness from randomness sources with high average min-entropy.

Theorem 2.6.1 ([3, 106, 130, 210])**.** *Let $\mathcal{X} = \{X_1, \ldots, X_k\}$ be a ρ^2-biased family of distributions over $\mathbb{F}^n$. Let (M, L) be a joint distribution such that the marginal distribution M is over $\mathbb{F}^n$ and $\widetilde{\mathbf{H}}_\infty(M|L) \geqslant m$. Then, if J is a uniform distribution over the set $[k]$, we have*

$$\mathrm{SD}\left((F_J \oplus M, L, J), (U_{\mathbb{F}^n}, L, J)\right) \leqslant \frac{\rho}{2} \cdot \left(\frac{|\mathbb{F}|^n}{2^m}\right)^{1/2}.$$

2.7 General Coding Theory Preliminaries

We state general coding theory definitions here and begin by defining a metric.

Definition 2.7.1. *Let Σ be a set and let $\Delta\colon \Sigma^* \times \Sigma^* \to \mathbb{R}_{\geqslant 0}$ be a function. Then we say that Δ is a* metric *if the following hold: for all $x, y, z \in \Sigma^*$, we have*

- $\Delta(x, y) = 0$ *if and only if* $x = y$;
- $\Delta(x, y) = \Delta(y, x)$; *and*
- $\Delta(x, y) \leqslant \Delta(x, z) + \Delta(z, y)$.

When $0 \leqslant \Delta(x, y) \leqslant 1$ for all $x, y \in \Sigma^*$, we say that Δ is a normalized metric.

The two main metrics of interest in this work are the (normalized) Hamming distance and the (normalized) Edit distance. For a set[1] Σ and positive integer K, the *normalized Hamming distance* between two strings $x, y \in \Sigma^K$ is defined as $\mathsf{HAM}(x, y) := |\{i \in [K]\colon x_i \neq y_i\}|/K$. For two strings $x, y \in \Sigma^*$, the *normalized Edit distance* between x and y is the minimum number of insertions and deletions needed to transform x into y, normalized by $|x| + |y|$. We let $\mathsf{ED}(x, y)$ denote the normalized Edit distance. Note that ED may not be a metric; however, this is not an issue for us as our proofs never use the fact that the non-normalized edit distance is a metric.

We now define error-correcting codes.

Definition 2.7.2 (Error-Correcting Codes)**.** *A* coding scheme *$C[K, k, q_1, q_2]$ is defined by a (possibly randomized) encoding function $\mathsf{Enc}\colon \Sigma_1^k \to \Sigma_2^K$, where $|\Sigma_i| = q_i \geqslant 2$.*

For normalized metric Δ and parameter δ, a $C[K, k, q_1, q_2]$ coding scheme is a (Δ, δ) error-correcting code *if there exists a deterministic decoding function $\mathsf{Dec}\colon \Sigma_2^* \to \Sigma_1^k$ satisfying the following property: for every $x \in \Sigma_1^k$ and any $y \in \Sigma_2^*$ such that $\Delta(\mathsf{Enc}(x), y) \leqslant \delta$, we have $\mathsf{Dec}(y) = x$ with probability 1. If $q_1 = q_2$, we simply write $C[K, k, q]$. If $q_2 = 2$, we say that C is a* binary code*. When clear from context, we omit q, q_1, and q_2.*

The rate *of a code $C[K, k, q]$ is defined as $R := k/K$, the* error tolerance *of C is δ, and the* minimum distance *of C is $d := \delta \cdot K$. Without loss of generality, we assume $K \geqslant k$.*

[1] ↑Often we call Σ an *alphabet*.

When $\Delta = \mathsf{HAM}$ *we say that* C *is a* Hamming code*; when* $\Delta = \mathsf{ED}$ *we say that* C *is an* insertion-deletion (InsDel) code.

Given a $C[K, k, q_1, q_2]$ error-correcting code, we often abuse notation and let $C \subseteq \Sigma_2^K$ denote the set of all codewords; that is, $C = \mathrm{Im}(\mathsf{Enc})$ (the image of Enc). We note that sampling $y \xleftarrow{\$} C$ is equivalent to the randomized process of first sampling $x \xleftarrow{\$} \Sigma_1^k$ then outputting $y \leftarrow \mathsf{Enc}(x)$.

We are also interested in linear error-correcting codes.

Definition 2.7.3 (Linear Error-Correcting Code)**.** *A* (Δ, δ) *code* $C[K, k, q_1, q_2]$ *is a* linear code *if* Σ_1^k *and* Σ_2^K *are vector spaces and* Enc *is a linear transformation. Without loss of generality, we assume* Σ *is a finite field and* Enc *is injective whenever* C *is a linear code.*

By definition, a linear code always has $0^K \in C$. Moreover, the minimum distance d of a linear code C is given by the minimum weight non-zero codewords.

Lemma 2.7.1. *Let* $C[K, k, q_1, q_2]$ *be a linear code. Then* $d = \min_{c \in C}\{\mathsf{wt}(c) > 0\}$*, where* $\mathsf{wt}(c)$ *denotes the number of non-zero entries of* c*.*

Every linear Hamming code C has a dual/orthogonal code, which is a set of vectors that are orthogonal to every codeword $c \in C$.

Definition 2.7.4. *Let* $C[K, k, q_1, q_2]$ *be a linear Hamming code with distance* d*. Then the set* $C^\perp := \{u \in \Sigma_2^K \colon \langle u, v\rangle = 0 \ \forall v \in C\}$ *is called the* dual/orthogonal code *of* C*. Moreover,* $C^\perp = C^\perp[K, K - k, q_1, q_2]$ *is a linear Hamming code.*

The following propositions are two well-studied and used bounds when analyzing and using linear codes.

Proposition 2.7.1 (Singleton Bound [244])**.** *Any* $C[K, k, q]$ *linear Hamming code has distance* $d \leqslant K - k + 1$*. Furthermore, if* $d = K - k + 1$*, then* C *is called* maximum distance separable *(or MDS).*

Proposition 2.7.2 (Gilbert-Varshamov Bound for Linear Codes [126, 252])**.** *For every* $q \geqslant 2$*, every* $0 \leqslant \delta < 1 - 1/q$*, and every* $0 < \varepsilon \leqslant 1 - \mathbf{h}_q(\delta)$*, there exists a linear* (HAM, δ) *code* $C[K, k, q]$ *with rate* $R \geqslant 1 - \mathbf{h}_q(\delta) - \varepsilon$*.*

2.7.1 Locally Decodable Codes

We are also interested in the notion of *locally decodable codes*.

Definition 2.7.5 (Locally Decodable Codes)**.** *A coding scheme* $C[K, k, q_1, q_2] = (\mathsf{Enc}, \mathsf{Dec})$ *is an* $(\ell, \rho, p, \mathrm{dist})$-locally decodable code *(LDC) if for all* $x \in \Sigma_1^k$ *and* $y \in \Sigma_2^*$ *such that* $\mathrm{dist}(\mathsf{Enc}(x), y) \leqslant \rho$*, the algorithm* Dec*, with query access to word* y*, on input index* $i \in [k]$*, makes at most* ℓ *queries to* y *and outputs* x_i *with probability at least* p *over the randomness of the decoder. Here,* ℓ *is the* locality *of* C *and* p *is the* success probability*.*

In later chapters, we augment the above definition with other notions of locally decodable codes; e.g., private LDCs (Definitions 5.5.1 and 6.5.2), resource-bounded LDCs (Definitions 5.5.2 and 6.5.3), and computationally relaxed LDCs (Definition 7.1.1).

Part I

COMPOSING CODING THEORY CONSTRUCTIONS AND TECHNIQUES WITH CRYPTOGRAPHIC PRIMITIVES

3. CORRELATION EXTRACTORS: CONSTRUCTING NEW ERROR-CORRECTING CODES FOR CRYPTOGRAPHIC PRIMITIVES

A portion of this chapter appears in The International Association for Cryptologic Research Cryptology ePrint Archive [50], available https://ia.cr/2018/372. The article [50] is the full version of the article which appears in the proceedings of the 2018 Theory of Cryptography Conference, published by The International Association for Cryptologic Research and Springer-Verlag [49], available https://doi.org/10.1007/978-3-030-03810-6_2.

Numerous cryptographic primitives used in both theory and practice have constructions based on error-correcting codes. Such primitives include (threshold) secret-sharing [44, 198, 202, 237], digital signature schemes [18, 19, 96, 98, 113, 212, 224], public-key cryptography [201, 212], succinct (non-)interactive arguments [14, 125, 175, 206], randomness extraction [1, 147, 235] and pseudo-random number generation [163, 235, 240, 246, 249, 250], resilient functions [38, 93], and secure multi-party computation [129, 174, 176, 263]. Cryptographic primitives based on error-correcting codes often have information-theoretic guarantees (i.e., protection against computationally unbounded adversaries) given by the underlying coding theoretic objects. For example, Shamir t out of n secret sharing [237] is precisely characterized by Reed-Solomon Codes of block length n and message length t, and such a secret sharing scheme guarantees that any adversary with only $t-1$ shares cannot learn *any* secret of the honest parties. Further, any t parties can work together to reconstruct the entire secret, which is guaranteed by the erasure recovery property of Reed-Solomon codewords (i.e., polynomial interpolation).

In this dissertation, we examine the composition of coding theory with *secure multi-party computation* (MPC) protocols. This composition is not new: various MPC protocols [129, 263] use coding theoretic primitives such as linear secret sharing [44, 237] to maintain privacy/security among participants. In a nutshell, a MPC protocol allows mutually distrusting parties $P_1, \ldots, P_n$ with respective private inputs $x_1, \ldots, x_n$ to compute $f(x_1, \ldots, x_n)$, where f is some function. Security of an MPC protocol ensures that any dishonest parties only learn the output $f(x_1, \ldots, x_n)$ and do not learn *any information* about the private inputs of the honest parties. Unfortunately, securely computing most functionalities in the information-theoretic

plain model is impossible, even in the presence of semi-honest adversaries—parties who follow the protocol honestly, but want to find additional information about the private input of other parties [129]. One can overcome this barrier by assuming an honest majority of parties [28, 82, 99, 223], assuming the computational power of the parties is bounded (e.g., PPT) [129, 160], or by using a trusted setup [77, 81, 101, 134, 160, 168, 209] or correlated private randomness [97, 177, 195, 259]. While these approaches allow parties to securely compute functionalities, many come with great computational costs. For example, honest majority MPC protocols are often (concretely) inefficient [167], protocols in which the parties have bounded computational power often make heavy use of (public-key) cryptography [100, 102, 135], and trusted setup assumptions are often undesirable.

3.1 Correlation Extractors

Protocols that utilize *correlated private randomness* (somewhat) address all of the above issues. Such protocols were introduced primarily to address the efficiency concerns of MPC protocols [158]. In this dissertation, we focus on two-party secure computation (2PC) between parties Alice and Bob using correlated private randomness. Protocols utilizing correlated private randomness offload most of the computational and cryptographic complexity to an offline pre-processing phase, where a trusted dealer samples secret shares (r_A, r_B) from some joint distribution (R_A, R_B), called the *correlated private randomness* or *correlation* in short. Note that this trusted dealer is presented for ease of discussion, and one can easily replace this dealer with some other method of pre-processing, including another MPC protocol to generate the secret shares. The dealer then provides share r_A to Alice and share r_B to Bob. During an online phase, Alice and Bob use their respective shares in an interactive protocol to compute some desired function f of their private inputs in a secure way (i.e., the MPC definition of security). This pre-processing phase is independent of the function computed in the online phase, and this independence is crucial for both efficiency and security. For efficiency, secret shares that are independent of the functionality to be computed means that many shares can be generated via an expensive pre-processing phase *once* and then parties can use these shares across many different protocols, amortizing the pre-processing costs.

For security, the independence of the correlation shares from the functionality to be computed ensures that Alice and Bob cannot violate the other's security/privacy by gaining information on the other party's private input when using the shares in the online phase; i.e., having secret shares that depend on the functionality to be computed can reveal information about the other party's inputs. However, there is a caveat: the storage of the secret shares r_A and r_B are subject to *leakage attacks*, where parties can leak information about the secret shares of the other parties, but otherwise to not tamper with these shares.[1] Leakage concerns stem from real-world incidents of information leakage, such as Rowhammer attacks [178], Heartbleed attacks [107, 247], and Meltdown/Spectre attacks [181, 187]. In our security model, leakage is not restricted to individual bits: it can be arbitrary leakage that encodes global information, which is stronger than individual bit leakage. For example, r_A and r_B could be shares of oblivious transfer [176, 222, 262], and receiver Bob can leak the other bit of Alice that he did not receive, thus violating any security guarantee given by oblivious transfer.

We model leakage for correlations as the following multi-step process.

1. The trusted dealer samples secret shares $(r_A, r_B) \leftarrow (R_A, R_B)$;
2. the adversarial party sends a bounded leakage function L to the dealer and receives leakage $L(r_A, r_B)$; and
3. the dealer then sends r_A to Alice and r_B to Bob.

Note that for two-party secure computation, there is only a single adversarial party. If both parties are adversarial, there are no privacy guarantees. For t bit leakage function $L\colon \{0,1\}^* \to \{0,1\}^t$, we represent the above *leaky correlation hybrid* (i.e., leaky correlation) as $(R_A, R_B)^{[t]}$. Note that t need not be as large as the entire secret, since leaking even a single bit could render a protocol insecure. Further, t may be bounded for a number of other reasons, such as bounds on the bandwidth or storage capabilities of the adversary [139, 199].

Correlation extractors were introduced by Ishai et al. [158] to address these leakage concerns. Intuitively, correlation extractors are MPC protocols which take leaky correlations

[1] ↑One may naturally consider a stronger adversary that can tamper with secret shares, but this is beyond the scope of this dissertation.

as input and output samples of some (possibly different) correlation.[2] The key feature of these extractors is that the output of the protocol is *independent* of the leaky inputs, and leakage on the original shares does not correspond to leakage on the new shares.

It suffices for any correlation extractor to output new shares of the well-known and well-studied *random oblivious transfer correlation*, which we denote by ROT. The ROT correlation is the randomized version of the *oblivious transfer* functionality, which is complete for MPC: assuming access to an ideal oblivious transfer functionality, *any* functionality admits a MPC protocol [174, 176]. For example, given oblivious transfer, one can compute more oblivious transfers, majority, or any other functionality of interest. More formally, the ROT correlation uniformly and independently samples three bits $x_0, x_1, b \xleftarrow{\$} \{0,1\}$ and provides $r_A = (x_0, x_1)$ to Alice and $r_B = (b, x_b)$ to Bob. We let $\mathsf{ROT}^{m/2}$ denote the process of independently sampling $m/2$ instances of the aforementioned ROT correlation. Notice that Alice does not know Bob's bit b, and Bob does not know Alice's bit x_{1-b}, which is identical to the security definition of oblivious transfer [176, 222]. As our goal is to produce secret shares which can be used by parties to compute *any* functionality, it suffices for correlation extractors to output fresh shares of the ROT correlation.

Let (R_A, R_B) be a correlation such that the secret share of each party is n bits. Then we define a (n, m, t, ε)-*correlation extractor* for (R_A, R_B) as a two-party interactive protocol in the $(R_A, R_B)^{[t]}$-hybrid that securely implements the $\mathsf{ROT}^{m/2}$ functionality against information-theoretic semi-honest[3] adversaries with ε-security. Briefly, ε-security states that given t bits of leakage, a corrupt party cannot (statistically) distinguish between the honest parties newly generated ROT samples and random bits, except with probability at most ε; see Definition 3.5.1 for the formal definition. Here, the parameters of interest are the *production rate* m/n (resp., *production* m), the *leakage rate* t/n (resp., *leakage resilience* t), the *round complexity*, and the *security* ε. A *round* of the protocol corresponds to the process where Bob sends a message to Alice, then Alice sends a message to Bob.

[2] ↑It can be the case that the input and output correlations are the same.
[3] ↑The existence of *malicious* secure correlation-extractors is an interesting and challenging open question, but is beyond the scope of this dissertation.

3.2 Our Results

Our main result is obtaining a correlation extractor for the ROT correlation with asymptotically optimal parameters.

Theorem 3.2.1 ([49]). *There exists a 1-round (n, m, t, ε)-correlation extractor for $(R_A, R_B) = \mathsf{ROT}^{n/2}$ such that $m = \Theta(n)$, $t = \Theta(n)$, and $\varepsilon = 2^{-\Theta(n)}$.*

Constructing fresh ROT samples requires both parties to send at least 1 message, hence 1 round is optimal. Note that it is impossible to obtain more than n bits of output when given n bits of input, else this would reduce the security (i.e., there isn't enough entropy), so the production m is (asymptotically) optimal as well. For leakage, there is a known upper bound on the leakage resilience of any correlation extractor of $t = (1/4 - g) \cdot n$ for any gap $g \in (0, 1/4]$ [159]. In fact, for any gap $g \in (0, 1/4]$, we obtain a correlation extractor with optimal rate, security, and round complexity; we present this result in Theorem 3.2.3. Finally, the security is optimal as the outputs of each party are $m = \Theta(n)$ bits, so any adversary can simply guess the outputs and be correct with $2^{-\Theta(n)}$ probability.

At the core of Theorem 3.2.1 is another correlation extractor for the *random oblivious linear-function evaluation* correlation [211, 259]—a generalization of the ROT correlation to finite fields. For a finite field $\mathbb{F}$, the random oblivious linear-function evaluation correlation, denoted by $\mathsf{ROLE}(\mathbb{F})$, samples $a, b, x \xleftarrow{\$} \mathbb{F}$ and defines $r_A = (a, b)$ and $r_B = (x, z := a \cdot x + b)$. For $\mathbb{F} = \mathbb{F}_2$, observe that $(x_0 + x_1) \cdot b + x_0 = x_b$, which implies that ROT and $\mathsf{ROLE}(\mathbb{F}_2)$ are identical. One share of the $\mathsf{ROLE}(\mathbb{F})$ correlation has length $2\log(|\mathbb{F}|)$ bits, and thus we normalize the correlation to output shares of length n bits. That is, $\mathsf{ROLE}(\mathbb{F})^{n/(2\log(|\mathbb{F}|))}$ outputs n bit shares to each party, where each share consists of $n/(2\log(|\mathbb{F}|))$ independent samples of $\mathsf{ROLE}(\mathbb{F})$. Without loss of generality, we choose n appropriately such that $n/(2\log(|\mathbb{F}|))$ is an integer. We construct a correlation extractor for $\mathsf{ROLE}(\mathbb{F})$ over a suitable constant-sized field $\mathbb{F}$ with asymptotically optimal parameters, which we later use to realize Theorem 3.2.1.

Theorem 3.2.2 ([49]). *There exists a 1-round (n, m, t, ε)-correlation extractor for $(R_A, R_B) = \mathsf{ROLE}(\mathbb{F})^{n/2\log|\mathbb{F}|}$ such that $m = \Theta(n)$, $t = \Theta(n)$, and $\varepsilon = 2^{-\Theta(n)}$.*

Theorem 3.2.2 is a 2PC protocol that relies on the existence of suitable *algebraic geometric (AG) error-correcting codes* [121, 133, 254] over constant-sized finite fields such that $|\mathbb{F}| \geqslant 49$ and is an even power of a prime. For simplicity, we use $\mathbb{F}$ of characteristic 2. These codes have several key properties necessary for the construction of Theorem 3.2.2, which we discuss in Section 3.3. Again, the parameters of Theorem 3.2.2 are optimal, and, in fact, we can achieve optimal parameters for any $t = (1/4 - g)n$ for any $g \in (0, 1/4]$, albeit at a cost of the field-size increasing and depending on how small of a gap g is given.

Theorem 3.2.3. *For every $g \in (0, 1/4]$, there exists a finite field $\mathbb{F}$ of characteristic 2 and a 1-round (n, m, t, ε)-correlation extractor for $(R_A, R_B) = \mathsf{ROLE}(\mathbb{F})^{n/(2\log(|\mathbb{F}|))}$ such that $m = \Theta(n)$, $t = (1/4 - g)n$, and $\varepsilon = 2^{-\Theta(n)}$.*

3.3 Technical Overview

The correlation extractor of Theorem 3.2.2 is built upon a $\mathsf{ROLE}(\mathbb{F})$-*to*-$\mathsf{ROLE}(\mathbb{F})$ *correlation extractor*, where both the input and output correlations are the $\mathsf{ROLE}(\mathbb{F})$ correlation for suitable $\mathbb{F}$. In particular, each party starts with t-leaky shares of $\mathsf{ROLE}(\mathbb{F})^{n/2\log|\mathbb{F}|}$ and the extractor outputs fresh, independent samples of the $\mathsf{ROLE}(\mathbb{F})^{m/2\log|\mathbb{F}|}$ correlation.

This $\mathsf{ROLE}(\mathbb{F})$-to-$\mathsf{ROLE}(\mathbb{F})$ extractor relies on a suitable family of linear Hamming codes over $\mathbb{F}$, which we eventually instantiate via a family algebraic geometric codes. For the remainder of this chapter, we only concern ourselves with Hamming codes and simply refer to them as codes. We first discuss how to obtain fresh samples of $\mathsf{ROLE}(\mathbb{F})$ by utilizing a linear code $C \subseteq \mathbb{F}^s$, where $s \in \mathbb{N}$, and the *Schur-product code* $C * C$. Then we analyze the necessary properties of this linear code C we need for our $\mathsf{ROLE}(\mathbb{F})$ extractor to work. Briefly, for linear codes $C_1, C_2 \subseteq \mathbb{F}^s$, the Schur-product code $C_1 * C_2$ is the linear span of $c * c' = (c_1 \cdot c'_1, \ldots, c_s \cdot c'_s) \in \mathbb{F}^s$ for all $c \in C_1$ and $c' \in C_2$. Suppose that the Schur-product code $C * C$ supports erasure recovery of any s_1-coordinates and let $s_1 + s_2 = s$. Then we can construct s_1 fresh $\mathsf{ROLE}(\mathbb{F})$ samples as follows.

1. Sample $(u_1, u_2), (r_1, r_2) \xleftarrow{\$} C$, where $u_i, r_i \in \mathbb{F}^{s_i}$.
2. Sample $(v_1, v_2) \xleftarrow{\$} C * C$, where $v_i \in \mathbb{F}^{s_i}$.

3. Compute $t_2 := (u_2 * r_2) + v_2 \in C * C$.

4. Perform erasure recovery on t_2 to obtain $t_1 \in \mathbb{F}^{s_1}$. The corresponding $\mathsf{ROLE}(\mathbb{F})$ shares are $r_A = (u_1, v_1)$ and (r_1, t_1), and the s_1-erasure recovery of $C * C$ ensures that $t_1 = (u_1 * r_1) + v_1$.

When given s_2 (leaky) shares of $\mathsf{ROLE}(\mathbb{F})$, we can compose the above methodology with our leaky shares to construct our $\mathsf{ROLE}(\mathbb{F})$-to-$\mathsf{ROLE}(\mathbb{F})$ extractor that outputs s_1 fresh $\mathsf{ROLE}(\mathbb{F})$ shares. We present the actual protocol in Figure 3.3.

Remark 3.3.1 (Erasures vs. Deletion Errors)**.** *Erasures are a type of error pattern where symbols are erased the indices of erased symbols are known. This is in contrast to deletion errors in insertion-deletion codes: in these error patterns, the indices of deleted symbols are unknown. Intuitively, one can view erasures as insertion-deletion errors where a symbol $\perp \notin \Sigma$ is inserted whenever a symbol of the codeword is deleted.*

For security purposes, it does not suffice to have a single fixed code C. Instead, we need a family of codes $\mathcal{C} = \{C_j\}_{j \in \mathcal{J}}$ such that any code in the family can be used in the above process. Given such a family, informally our correlation extractor is given by the following protocol. Suppose that Alice is the semi-honest party and Bob is the honest party. First Alice and Bob both receive the initial $\mathsf{ROLE}(\mathbb{F})$ correlation samples, and Alice performs t bits of leakage on Bob's secret shares. Then, the first step of the protocol is for Bob to sample a random code from this family and use it with the above methodology (along with his leaky shares) to construct new $\mathsf{ROLE}(\mathbb{F})$ samples.

To see why a single fixed code C does not suffice, let $\mathcal{C} = \{C_j\}_{j \in \mathcal{J}}$ a the family of linear codes of block length $s \in \mathbb{N}$ that we can use for our correlation extractor. Observing the above process for constructing fresh $\mathsf{ROLE}(\mathbb{F})$ samples, the family $\mathcal{C}$ must satisfy the following properties.

1. **Multiplication Friendly Good Codes.** For every $j \in \mathcal{J}$, the code $C_j \subseteq \mathbb{F}^s$ is a *good code*; that is, both the rate and error-tolerance of C_j are $\Theta(1)$ (i.e., C_j has distance $\Theta(s)$). Furthermore, each code C_j is *multiplication friendly*; that is, the Schur-product code $C_j * C_j$ is a linear code with distance $\Theta(s)$. Multiplication friendly codes are useful

in the context of secret sharing: they allow one to perform the multiplication of two secrets by multiplying their respective secret shares.

2. **Small-Bias Family.** Informally, a *small-bias family* defines a pseudo-random distribution for linear tests (i.e., linear functions) [2, 3, 106, 130, 210] (see Definition 2.6.2 for the formal definitions). For $S \in \mathbb{F}^s$, let $L_S \colon \mathbb{F}^s \to \mathbb{F}$ be the linear test defined as $L_S(x) := \langle S, x \rangle$. Consider the distribution D defined by first sampling $j \xleftarrow{\$} \mathcal{J}$, then sampling $c \xleftarrow{\$} C_j$, then outputting $L_S(c)$. Then the family $\mathcal{C}$ is a *ρ-biased family* if D has statistical distance at most ρ from the distribution $L_S(u)$ for $u \xleftarrow{\$} \mathbb{F}^s$. For ease of presentation, we say that $\mathcal{C}$ *ρ-fools* the linear test L_S.

   One property of any linear code $C \subseteq \mathbb{F}^s$ is that any $c \xleftarrow{\$} C$ can 0-fool every linear test L_S such that $S \notin C^\perp$, the dual code of C. However, if $S \in C^\perp$, then $L_S(c) = 0$ for any $c \in C$, and thus the test is not fooled at all since there exists sets S such that $L_S(c)$ is trivially distinguishable.

   This property shows us that one fixed linear code cannot fool *all* linear tests; however, by considering an appropriate family of linear codes, then a randomly chosen codeword from a *randomly chosen code* in the family can fool *every* linear test.

We construct the family $\mathcal{C}$ by utilizing an appropriate AG code C. Suppose C is an AG code that is multiplication-friendly [84, 121, 133]. Given such a code, we apply random "twist-then-permute" operations on C to construct the family $\mathcal{C}$. The "twist-then-permute" operation first requires a "twist" followed by a "permute". Let $\alpha \in (\mathbb{F}^\times)^s$. Then an *$\alpha$-twist* of C is defined as the linear code

$$C_\alpha := \{\alpha * c \colon c \in C\}. \tag{3.1}$$

For permutation $\pi \colon [s] \to [s]$, we define a *π-permutation* of the α-twisted code C_α as

$$C_{\pi,\alpha} = \{(\alpha_{\pi(1)} \cdot c_{\pi(1)}, \ldots, \alpha_{\pi(s)} \cdot c_{\pi(s)}) \colon c \in C\}. \tag{3.2}$$

Note that for any π and α, the code $C_{\pi,\alpha}$ enjoys all the same properties of the code C since $C_{\pi,\alpha}$ is an *equivalent code*.[4] In particular, $C_{\pi,\alpha}$ has the same distance and rate as C, and if C is a multiplication friendly code, then so is $C_{\pi,\alpha}$. Defining $\mathcal{J} = \{(\pi, \alpha)\}$ such that π is a permutation on the set $[s]$ and $\alpha \in (\mathbb{F}^\times)^s$, we define our code family as $\mathcal{C} := \{C_j\}_{j\in\mathcal{J}}$.

Finally, we observe that $\mathcal{C}$ is a small-bias family of linear codes. To see this, first consider the following two distributions. For fixed S, let D_S be the following distribution: (1) sample $j \xleftarrow{\$} \mathcal{J}$; (2) sample $c \xleftarrow{\$} C_j$; (3) output $L_S(c)$. Further, let $D_{\mathsf{wt}(S)}$ be the following distribution: (1) sample $T \xleftarrow{\$} \{T \in \mathbb{F}^s : \mathsf{wt}(T) = \mathsf{wt}(S)\}$; (2) sample $c \xleftarrow{\$} C$; (3) output $L_T(c)$. Arguing that $\mathcal{C}$ is a small-bias family relies on the observation that D_S and D'_S are *identical distributions*. Based on this observation, the bias of $\mathcal{C}$ is simply the ratio between the number of codewords in $C^\perp$ of weight $w = \mathsf{wt}(S)$ and the number of elements of $\mathbb{F}^s$ of weight w. In more detail, $\binom{s}{w} \cdot (q-1)^w$ elements of $\mathbb{F}^s$ of weight w. Let A_w be the number of elements of $C^\perp$ of weight w. Then $\mathcal{C}$ ρ-fools the linear test L_S for $\rho = A_w / \left(\binom{s}{w} \cdot (q-1)^w\right)$. In fact, the quantity A'_w is called the *weight enumerator* [193, 254] and is an important quantity in the study of linear codes. To estimate the bias of our codes, we obtain precise asymptotic bounds on A_w of the dual code $C^\perp$. The precise bounds on A_w correspond to higher production, leakage resilience, and exponentially low security error for our resulting correlation extractor. The formal analysis of this weight enumerator is deferred to [50], the full version of [49].

3.3.1 Constructing our ROLE($\mathbb{F}$)-to-ROLE($\mathbb{F}$) Correlation Extractor.

Given a family of multiplication-friendly, small-biased linear code distributions $\{C_j\}_{j\in\mathcal{J}}$ defined above, we outline our ROLE($\mathbb{F}$)-to-ROLE($\mathbb{F}$) correlation extractor, noting that the full protocol is presented in Figure 3.4 and all formal definitions and proofs are given in Section 3.7.1. Suppose that every code $C_j \subseteq \mathbb{F}^{\gamma+\eta}$ for positive integers γ, η, and every Schur-product code $C_j * C_j$ supports erasure recovery of any γ coordinates. First, the trusted dealer samples $((A, B), (X, Z)) \leftarrow \mathsf{ROLE}(\mathbb{F})^\eta$ and gives secret share (A, B) to Alice and secret share (X, Z) to Bob, where $A, B, X \xleftarrow{\$} \mathbb{F}^\eta$ and $Z_i := A_i \cdot X_i + B_i$ for every $i \in [\eta]$. Then the

[4] ↑In the literature there are multiple definitions for the *equivalence* of two linear codes. In particular, one such notion (cf., [220]), states that two codes are equivalent to each other if one can be twisted-and-permuted into the other code. For clarity, we have chosen to explicitly define the "twist then permute" operation.

semi-honest corrupt party performs arbitrary t bits of leakage on the honest party's secret share. Now Alice and Bob engage in the following protocol.

1. Bob samples $j \xleftarrow{\$} \mathcal{J}$. Then Bob samples codeword $(R, R') \xleftarrow{\$} C_j$ such that $R \in \mathbb{F}^\gamma$ and $R' \in \mathbb{F}^\eta$. Bob computes $M = X + R' \in \mathbb{F}^\eta$ and sends (M, j) to Alice.

2. Alice samples $(U, U') \xleftarrow{\$} C_j$ and $(V, V') \xleftarrow{\$} C_j * C_j$ such that $U, V \in \mathbb{F}^\gamma$ and $U', V' \in \mathbb{F}^\eta$. Alice computes $\alpha = U' - A \in \mathbb{F}^\eta$ and $\beta = (A * M) + B + V' \in \mathbb{F}^\eta$. Alice sends (α, β) to Bob.

3. Bob computes $W' = (\alpha * R') + \beta - Z \in \mathbb{F}^\eta$. Bob then uses the erasure recovery algorithm of $C_j * C_j$ to recover $W \in \mathbb{F}^\gamma$ such that $(W, W') \in C_j * C_j$.

4. Alice outputs share (U, V) and Bob outputs share (R, W).

Since the corrupt party is semi-honest, this party follows the protocol honestly after performing t bits of leakage. By the γ-erasure recovery of $C_j * C_j$ and working out the above algebra we observe that $W_i = U_i \cdot R_i + V_i$ for every $i \in [\gamma]$, and further that $U_i, R_i, V_i \xleftarrow{\$} \mathbb{F}^\gamma$. Thus the shares output by parties Alice and Bob are a new instance of the $\mathsf{ROLE}(\mathbb{F})$ correlation, as desired.

3.3.2 Obtaining Theorem 3.2.2.

To construct our correlation extractor for $\mathsf{ROLE}(\mathbb{F})^{n/(2\log(|\mathbb{F}|))}$ which realizes Theorem 3.2.2, we augment our $\mathsf{ROLE}(\mathbb{F})$-to-$\mathsf{ROLE}(\mathbb{F})$ correlation extractor with the ROT-*embedding protocol* of Block, Maji, and Nguyen [54], which is a type of *reverse multiplication-friendly embedding protocol* [54, 55, 78]. Informally, a reverse multiplication-friendly embedding computes multiple multiplications over a finite field $\mathbb{F}$ by computing a single multiplication over an extension field $\mathbb{K}$ of $\mathbb{F}$. For example, there is a simple embedding that computes 2 multiplications over $\mathbb{F}_2$ using a single multiplication over $\mathbb{F}_8 := \mathbb{F}_2[X]/(X^3 + X + 1)$. Let $(a_1, a_2), (b_1, b_2) \in \mathbb{F}_2^2$ and suppose we want to compute $a_i \cdot b_i$ for $i \in [2]$ and Define $A = a_1 + a_2 \cdot X \in \mathbb{F}_8$ and $B = b_1 + b_2 \cdot X \in \mathbb{F}_8$, and compute $A \cdot B = a_1 \cdot b_1 + (a_1 \cdot b_2 + a_2 \cdot b_1) \cdot X + a_2 \cdot b_2 \cdot X^2 \in \mathbb{F}^8$. Then we can extract out $a_i \cdot b_i$ by taking the coefficient of the constant term and the X^2

term, respectively for $i \in [2]$. Note that embedding protocols naturally extend to computing multiple linear equations of the form $a_i \cdot b_i + c_i$; in the above example, one would define $C = c_1 + c_2 \cdot X^2$, compute $A \cdot B + C$, and extract out the same coefficients to obtain $a_i \cdot b_i + c_i$ for $i \in [2]$.

We can re-imagine a reverse multiplication-friendly protocol as a protocol which computes multiple instances of an *oblivious linear function evaluation* over a finite field $\mathbb{F}$, which we denote as $\mathsf{OLE}(\mathbb{F})$, by computing a single instance of $\mathsf{OLE}(\mathbb{K})$, where $\mathbb{K}$ is an finite extension field of $\mathbb{F}$. Note that $\mathsf{OLE}(\mathbb{F})$ is the non-randomized version of $\mathsf{ROLE}(\mathbb{F})$. Since $\mathsf{OLE}(\mathbb{F})$ and ROT are equivalent for $\mathbb{F} = \mathbb{F}_2$, an ROT-embedding protocol is simply a reverse multiplication-friendly protocol for OLE.

More formally, an ROT-embedding protocol is a protocol between two parties that implements the $\mathsf{ROT}^{m/2}$ functionality in the $\mathsf{OLE}(\mathbb{F})$-hybrid, where $\mathbb{F}$ is a field of characteristic 2. In particular, sender Alice samples $a \xleftarrow{\$} \{0,1\}^{m/2}$ and $b \xleftarrow{\$} \{0,1\}^{m/2}$ and receiver Bob sampling $x \xleftarrow{\$} \{0,1\}^{m/2}$. Then Alice embeds a and b into a field elements $A \in \mathbb{F}$ and $B \in \mathbb{F}$, respectively, while Bob embeds x into $X \in \mathbb{F}$. Alice sends A, B to the $\mathsf{OLE}(\mathbb{F})$ functionality and Bob sends X to the $\mathsf{OLE}(\mathbb{F})$ functionality and receives $Z = A \cdot X + B$. The embedding specifies a decoding procedure which on input Z allows Bob to obtain $z_i = (a_i + b_i) \cdot x_i + a_i$ for every $i \in [m/2]$, thus realizing the ROT correlation. The embedding protocol is used within our $\mathsf{ROLE}(\mathbb{F})$-to-$\mathsf{ROLE}(\mathbb{F})$ protocol so that upon recovering the fresh $\mathsf{ROLE}(\mathbb{F})$ samples, the receiver Bob can then use this decoding procedure to recover multiple instances of the ROT correlation. We emphasize that this embedding is composed in parallel with our $\mathsf{ROLE}(\mathbb{F})$-to-$\mathsf{ROLE}(\mathbb{F})$ extractor and that the embedding procedure is a 1-round protocol where Bob sends the first message. Thus composing it in parallel preserves the round complexity.

We compose this embedding protocol with our $\mathsf{ROLE}(\mathbb{F})$-to-$\mathsf{ROLE}(\mathbb{F})$ protocol as follows. Let $\mathsf{Emb}_{\mathsf{ROT}} \colon \{0,1\}^{m/2} \to \mathbb{F}^\gamma$ and $\mathsf{Ext}_{\mathsf{ROT}} \colon \mathbb{F}^\gamma \to \{0,1\}^{m/2}$ be the embedding and extraction functions of the ROT-embedding.[5] Then our protocol realizing Theorem 3.2.2 is the following protocol, where underline the differences with the ROLE-to-ROLE protocol presented previously.

[5]↑Note in our context, we are interested in embedding $m/2$ bits into γ field elements. We achieve this by running a single embedding instance γ times.

1. Bob samples $j \xleftarrow{\$} \mathcal{J}$ and samples $\tilde{b} \xleftarrow{\$} \{0,1\}^{n/2}$ random bits. Then Bob samples codeword $(R, R') \xleftarrow{\$} C_j$ such that $R \in \mathbb{F}^\gamma$ and $R' \in \mathbb{F}^\eta$ and $R = \mathsf{Emb}_{\mathsf{ROT}}(\tilde{b})$. Bob computes $M = X + R' \in \mathbb{F}^\eta$ and sends (M, j) to Alice.

2. Alice samples $\tilde{x}_0, \tilde{x}_1 \xleftarrow{\$} \{0,1\}^{n/2}$ random bits and samples $(U, U') \xleftarrow{\$} C_j$ and $(V, V') \xleftarrow{\$} C_j * C_j$ such that $U, V \in \mathbb{F}^\gamma$ and $U', V' \in \mathbb{F}^\eta$ and $U = \mathsf{Emb}_{\mathsf{ROT}}(\tilde{x}_0)$ and $V = \mathsf{Emb}_{\mathsf{ROT}}(\tilde{x}_1)$. Alice computes $\alpha = U' - A \in \mathbb{F}^\eta$ and $\beta = (A * M) + B + V' \in \mathbb{F}^\eta$. Alice sends (α, β) to Bob.

3. Bob computes $W' = (\alpha * R') + \beta - Z \in \mathbb{F}^\eta$. Bob then uses the erasure recovery algorithm of $C_j * C_j$ to recover $W \in \mathbb{F}^\gamma$ such that $(W, W') \in C_j * C_j$. Bob computes $\tilde{x} = \mathsf{Ext}_{\mathsf{ROT}}(W)$.

4. Alice outputs shares $(\tilde{x}_0, \tilde{x}_1)$ and Bob outputs shares $(\tilde{b}, \tilde{x})$.

The correctness of the protocol (due to semi-honest security) and the correctness of the ROT-embedding protocol guarantees that for every $i \in [m/2]$, we have $\tilde{x}_i = \tilde{x}_{\tilde{b}_i, i}$, as desired. See Section 3.7.2 for more details.

3.3.3 Obtaining Theorem 3.2.1.

To realize Theorem 3.2.1, we compose the correlation extractor of Theorem 3.2.2 with a protocol to efficiently compute multiplications over an extension field using multiplications over the base field (of the extension). This is known as a *multiplication-friendly embedding protocol* [80, 94], and is opposite of reverse multiplication-friendly embedding protocols discussed above. This is one of several applications of algebraic function fields first discovered by Chundovsky and Chundovsky [94]. For example, 6 multiplications over $\mathbb{F}_2$ suffice to perform a single multiplication over $\mathbb{F}_{2^3}$, and 15 multiplications over $\mathbb{F}_2$ suffice to perform a single multiplication over $\mathbb{F}_{2^6}$ (cf., [80, Table 1]).

At a high-level, a multiplication friendly embedding protocol operates as follows. Let $\mathbb{F}$ be a finite field and $\mathbb{K}$ be an extension field of $\mathbb{F}$ and let $A, B, X \in \mathbb{K}$. The protocol then embeds A into a vector $a \in \mathbb{F}^k$, B into a vector $b \in \mathbb{F}^k$, and X into a vector $x \in \mathbb{F}^k$, for

some k. Finally, the protocol computes $a * x + b \in \mathbb{F}^k$, then extracts the linear function $A \cdot X + B \in \mathbb{K}$ from this vector.

In particular, we can model a multiplication-friendly embedding as a multiplication-friendly linear code D with encoding algorithm $\mathsf{Enc}_D \colon \mathbb{K} \to \mathbb{F}^k$ and decoding algorithm $\mathsf{Dec}_D \colon \mathbb{F}^k \to \mathbb{K}$. Recall that D is multiplication friendly if the code $D * D$ is a linear code with distance $\Theta(k)$. Given such an embedding, we outline the construction of our correlation extractor which realizes Theorem 3.2.1.

Being slightly more general, suppose D is multiplication-friendly linear code with encoding algorithm $\mathsf{Enc}_D \colon \mathbb{F}^\eta \to \{0,1\}^{n/2}$ and decoding algorithm $\mathsf{Dec}_D \colon \{0,1\}^{n/2} \to \mathbb{F}^\eta$ such that $\mathbb{F}$ is a field of characteristic 2. Further let $\mathsf{Enc}_{D^{(2)}}, \mathsf{Dec}_{D^{(2)}}$ be the encoder and decoder of $D^{(2)} := D * D$, respectively. Let $\mathsf{Emb}_{\mathsf{ROT}} \colon \{0,1\}^{m/2} \to \mathbb{F}^\gamma$ and $\mathsf{Ext}_{\mathsf{ROT}} \colon \mathbb{F}^\gamma \to \{0,1\}^{m/2}$ be the ROT embedding and extraction functions, respectively, from our $\mathsf{ROLE}(\mathbb{F})$ correlation extractor. Finally, let $\{C_j\}_{j \in \mathcal{J}}$ be the family of linear multiplication-friendly good codes of block length $\gamma + \eta$. Our correlation extractor is constructed as follows.

1. First the trusted dealer samples $((a,b),(x,z)) \xleftarrow{\$} \mathsf{ROLE}^{n/2}$ and gives secret share (a,b) to Alice and secret share (x,z) to Bob. Recall that $\mathsf{ROLE} := \mathsf{ROLE}(\mathbb{F}_2)$ and that for every $i \in [n/2]$, we have that $a_i, b_i, x_i \xleftarrow{\$} \{0,1\}$ and $z_i := a_i \cdot x_i + b_i$, and further recall that ROLE and ROT are functionally equivalent.

2. The semi-honest party performs t bits of leakage on the honest party's secret share.

3. Bob samples $j \xleftarrow{\$} \mathcal{J}$ and $\tilde{b} \xleftarrow{\$} \{0,1\}^{m/2}$. Bob then samples codeword $(R, R') \xleftarrow{\$} C_j$ such that $R \in \mathbb{F}^\gamma$, $R' \in \mathbb{F}^\eta$, and $R = \mathsf{Emb}_{\mathsf{ROT}}(\tilde{b})$. Finally, Bob computes $r' = \mathsf{Enc}_D(R') \in \{0,1\}^{n/2}$, computes $m = x + r' \in \{0,1\}^{n/2}$, and sends (m, j) to Alice.

4. Alice samples $\tilde{x}_0, \tilde{x}_1 \xleftarrow{\$} \{0,1\}^{n/2}$, and samples $(U, U') \xleftarrow{\$} C_j$ and $(V, V') \xleftarrow{\$} C_j * C_j$ such that $U, V \in \mathbb{F}^\gamma$, $U', V' \in \mathbb{F}^\eta$, and $U = \mathsf{Emb}_{\mathsf{ROT}}(\tilde{x}_0)$ and $V = \mathsf{Emb}_{\mathsf{ROT}}(\tilde{x}_1)$. Alice computes $u' = \mathsf{Enc}_D(U') \in \{0,1\}^{n/2}$ and $v' = \mathsf{Enc}_{D^{(2)}}(V') \in \{0,1\}^{n/2}$. Finally, Alice computes $\alpha = u' - a \in \{0,1\}^{n/2}$ and $\beta = (a * m) + b + v' \in \{0,1\}^{n/2}$ and sends (α, β) to Bob.

5. Bob computes $w' = (\alpha * r') + \beta - z \in \{0,1\}^{n/2}$. Bob then computes $W' = \mathsf{Dec}_{D^{(2)}}(w')$ and uses the erasure recovery algorithm of $C_j * C_j$ to recover $W \in \mathbb{F}^\gamma$ such that $(W, W') \in C_j * C_j$. Finally, Bob computes $\tilde{x} = \mathsf{Ext}_{\mathsf{ROT}}(W)$.

6. Alice outputs shares $(\tilde{x}_0, \tilde{x}_1)$ and Bob outputs shares $(\tilde{b}, \tilde{x})$.

The correctness of the protocol (due to semi-honest security), the decoding properties of the code $D^{(2)}$, and the correctness of the ROT-embedding protocol guarantees that for every $i \in [m/2]$, we have $\tilde{x}_i = \tilde{x}_{\tilde{b}_i, i}$, as desired. See Section 3.7.3 for more details.

3.4 Additional Related Work

As mentioned before, the study of correlation extractors was initiated by Ishai et al. [158]. The construction gives both parties $n/2$ samples of the ROT correlation, extracts $m/2 = \Theta(n)$ fresh samples of ROT, is resilient to $t = \Theta(n)$ bits of leakage, and has exponentially low security error $\varepsilon = 2^{-\Theta(n)}$, but the extractor has message complexity 4. This is due to the fact that their construction relies on a one-side secure correlation extractor; that is, security of this extractor would only hold if a single party was corrupt, and not the other party. This one-side secure extractor is a 2-message protocol, and they construct their extractor by running the one-side secure extractor twice to handle either party being corrupt, leading to message complexity 4. Gutpa et al. [140] constructed two correlation extractors. The first construction gives both parties $n/2$ shares of the ROT correlation and outputs $m/2 = n/\operatorname{polylog}(n)$ fresh samples of ROT. The extractor is resilient to $t = (1/4 - g)n$ bits of leakage for any gap $g \in (0, 1/4]$, has security error $2^{-gn/m}$, and is a 2-message protocol. The second correlation extractor utilized a new correlation introduced by the authors: the *inner-product correlation*. Briefly, the inner-product correlation over vector space $\mathbb{F}^s$ for finite field $\mathbb{F}$ samples uniformly random vectors $U, V \xleftarrow{\$} \mathbb{F}^s$ such that $\langle U, V \rangle = 0$. The second correlation extractor of Gupta et al. gives both parties a single sample of the inner-product correlation such that each sample is n-bits, extracts a *single* new sample of the ROT correlation (i.e., $m/2 = 1$), is resilient to $t = (1/2 - g)n$ bits of leakage for any $g \in (0, 1/2]$, and has security 2^{-gn}. While this extractor only produced a single new sample of the ROT correlation, it was the first correlation extractor to achieve higher than fractional leakage resilience $1/4$.

Block, Maji, and Nguyen [54] also constructed a correlation extractor for the inner-product correlation. The construction gives both parties a single sample of the inner-product correlation such that each sample is n-bits, extracts $m/2 = n^{1-o(1)}$ fresh samples of the ROT correlation, is resilient to $t = (1/2 - g)n$ bits of leakage for any $g \in (0, 1/2]$, has security error 2^{-gn}, and is a 2-message protocol. Further, the authors proved that any correlation extractor for the inner-product correlation has fractional leakage resilience $t/n < 1/2$, showing that their their extractor and the extractor of Gupta et al. [140] is optimal. In a follow-up work, the same authors in [55] improve the production rate of this inner-product correlation to $m/2 = \Theta(n)$, giving an optimal correlation extractor for the inner-product correlation (optimal in terms of production, resilience, security, and message complexity).

All of the above works utilize small-biased families of linear codes with appropriate properties. Similar to our construction, Ishai et al. [158] use multiplication-friendly algebraic-geometric codes to construct their extractor. The extractors of Gutpa et al. [140] and Block, Maji, and Nguyen [54, 55] all utilize random linear codes over $\mathbb{F}$ which are described by random Toeplitz matrices, and their dual and parity codes, along with (some form or another) of reverse multiplication-friendly embedding protocols. The reverse multiplication-friendly embedding of Block, Maji, and Ngyuen [55] was introduced independently and concurrently with the reverse multiplication-friendly embedding of Cascudo et al. [78] (the name also comes from Cascudo et al.); both have asymptotically optimal parameters and use similar ideas from the field of algebraic function theory.

3.4.1 OT Combiners

Oblivious Transfer combiners, or OT combiners in short, are a restriction of correlation extractors to the setting where an adversary can only leak individual bits of the secret share of the honest party; that is, the performed leakage cannot be functions of the bits of the secret share (e.g., cannot be the sum of two bits). OT combiners were first introduced and studied by Harnik et al. [154], leaking to many works on several variants and extensions of OT combiners [153, 160, 203, 204, 221]. Ishai et al. [159] constructed an OT combiner with nearly optimal leakage resilience. The additionally proved that *any* correlation extractor for

ROT (that is, given ROT as input) has fractional leakage resilience $t/n < 1/4$. The most relevant works to our result are the combiners due to Meier, Przydatek, and Wullschleger [204] and Przydatek and Wullschleger [221]. Using Reed-Solomon codes, they construct two-message error-tolerant combiners to produce fresh samples of $\mathsf{ROLE}(\mathbb{F})$ from shares of $\mathsf{ROLE}(\mathbb{F})$, where $\mathbb{F}$ is a large field. In particular, the field size increases with n, the size of the secret shares produced by the preprocessing step, and error-tolerant means that the combiner is secure even when a few of the correlation samples given as input are erroneous (i.e., not in the support of the correlation). Note that constructions similar to these can be constructed over appropriate constant-sized fields using multiplication-friendly secret sharing schemes based on algebraic geometric codes introduced by Chen and Cramer [84]. The malicious setting for OT combinders is examined in the work of Ishai, Prabhakaran, and Sahai [160]. Cascudo et al. [79] construct a high-resilience OT combiner, but it only produces a single OT (i.e., $m/2 = 1$). To date, malicious-secure correlation extractors remains a difficult and interesting open problem, for both positive and negative directions.

3.5 Correlation Extractor Preliminaries

We present preliminaries specific to correlation extractors.

3.5.1 Non-standard Notation

We use some non-standard notation in our exposition of correlation extractors for the purpose of clarity. For a positive integer $n \in \mathbb{Z}^+$, we let $[-n] := \{-1, -2, \ldots, -n\}$. For positive integers n_1, n_2 and vector $x \in \mathbb{F}^{n_1+n_2}$, we let $x_{[-n_1]}$ denote the first n_1 elements of x and $x_{[n_2]}$ denote the last n_2 elements of x. In other words, $x = (x_{[-n_1]}, x_{[n_2]})$.

3.5.2 Functionalities and Correlations

Oblivious Transfer. The 2-*choose*-1-*bit oblivious transfer*, denoted by OT. is a two-party functionality which takes input $(x_0, x_1) \in \{0,1\}^2$ from Alice and input $b \in \{0,1\}$ from Bob and outputs x_b to Bob.

Random Oblivious Transfer Correlation. The *random 2-choose-1-bit oblivious transfer*, denoted by ROT, is an inputless two-party correlation that samples $x_0, x_1, x_b \xleftarrow{\$} \{0,1\}$ uniformly and independently at random. It outputs secret share $r_A = (x_0, x_1)$ to Alice and $r_B = (b, x_b)$ to Bob. The joint distribution of Alice and Bob secret shares is called a ROT*-correlation.*

Oblivious Linear-Function Evaluation. Given a finite field $\mathbb{F}$, the *oblivious linear-function evaluation correlation*, denoted by $\mathsf{OLE}(\mathbb{F})$, is a two-party functionality that takes input $(a, b) \in \mathbb{F}$ from Alice and input $x \in \mathbb{F}$ from Bob and outputs $ax + b$ to Bob. Moreover we let $\mathsf{OLE} := \mathsf{OLE}(\mathbb{F}_2)$.

Random Oblivious Linear-Function Evaluation. Given a finite field $\mathbb{F}$, the *random oblivious linear-function evaluation*, denoted by $\mathsf{ROLE}(\mathbb{F})$, is a two-party correlation that samples field elements $a, b, x \xleftarrow{\$} \mathbb{F}$ uniformly and independently at random. It outputs secret share $r_A = (a, b)$ to Alice and $(b, ax + b)$ to Bob. Moreover we let $\mathsf{ROLE} := \mathsf{ROLE}(\mathbb{F}_2)$.

Fact 3.5.1 ([49, 54])**.** ROLE *and* ROT *are functionally equivalent.*

3.5.3 Correlation Extractors

Definition 3.5.1 (Correlation Extractors [49, 54, 55, 140, 158])**.** *Let* (R_A, R_B) *be a correlation such that the secret share of each party is* n*-bits. An* (n, m, t, ε)-correlation extractor *for* (R_A, R_B) *is a two-party interactive protocol between a sender and a receiver in the* $(R_A, R_B)^{[t]}$*-hybrid that securely implements the* $\mathsf{ROT}^{m/2}$ *functionality and satisfies the following properties.*

1. ***Correctness.*** *The correctness requirement states that the receiver's output is correct in all* $m/2$ *instances of* ROT*.*

2. (t, ε)***-Security.*** *We define our leakage model for correlation extractors in Figure 3.1. Let* $(s_{i,0}, s_{i,1})$ *and* (b_i, z_i) *be the output shares of sender Alice and receiver Bob, respectively, in the* i^{th} ROT *instance. Then* (t, ε)*-security states that a corrupt sender (resp., receiver) who participates in the corruption and* t*-bit leakage phase of Figure 3.1 cannot disntiguish between* $\{b_i\}_{i\in[m/2]}$ *(resp.,* $\{s_{i,1-b_i}\}_{i\in[m/2]}$*) and* $r \xleftarrow{\$} \{0,1\}^{m/2}$ *with advantage more than* ε*.*

Let $t \in \mathbb{N}$ be a parameter and let (R_A, R_B) be a correlation such that the secret shares given to each party are n-bits.

1. **Correlation Generation Phase.** Sender Alice receives r_A and receiver Bob receives r_B for $(r_A, r_B) \leftarrow (R_A, R_B)$.
2. **Corruption and t-bit Leakage Phase.** An information-theoretic semi-honest adversary corrupts either the sender or the receiver, then sends a leakage function $L\colon \{0,1\}^n \to \{0,1\}^t$, and receives $L(r_B)$ or $L(r_A)$, respectively. Note that this leakage is *arbitrary*: it is not limited to leakage on individual bits of the shares and can encode crucial global information.

Figure 3.1. Leakage model for correlation extractors.

We define m to be the production*, m/n to be the* production rate*, t to be the* leakage resilience*, and t/n to be the* leakage rate*.*

Through the remainder of this chapter, we fix parameter $n \in \mathbb{N}$ only consider $(R_A, R_B) = \mathsf{ROT}^{n/2}$ or $(R_A, R_B) = \mathsf{ROLE}(\mathbb{F})^\eta$, where $\mathbb{F}$ is a suitable constant-sized field and η is an integer such that $2\eta \log |\mathbb{F}| = n$. That is, Alice and Bob each begin with $n/2$ samples of the ROT correlation or with η samples of the $\mathsf{ROLE}(\mathbb{F})$ correlation.

3.5.4 Distributions over Linear Codes

Throughout the remainder of this chapter, we focus on linear Hamming codes; in particular, whenever we write "linear code" or "code", we mean it to be a Hamming code.

We adopt the following non-standard notation. For a (HAM, δ) linear code $C[\eta, \kappa, q]$ over a finite field $\mathbb{F} = \mathbb{F}_q$, we write $C = C[\eta, \kappa, d, d^\perp, d^{(2)}]$ to denote a linear code with generator matrix $G \in \mathbb{F}^{\kappa \times \eta}$, message length κ, block length η, and distance d such that $C^\perp$ has distance $d^\perp$ and $C^{(2)}$ has distance $d^{(2)}$. Here, we let $H \in \mathbb{F}^{\eta-\kappa \times \eta}$ denote the generator matrix of $C^\perp$, and $C^{(2)}$ denotes the *Schur-product code* of C, defined as $\operatorname{span} c * c' \colon c, c' \in C \subseteq \mathbb{F}^\eta$. For any permutation $\pi \in \mathcal{S}_\eta$, we define $G_\pi := \pi(G)$ as the generator matrix obtained by permuting the columns of G under π.

We also let C denote the uniform distribution over codewords generated by G; that is, $C = \{x \cdot G \colon x \xleftarrow{\$} \mathbb{F}^\kappa\}$, and abuse notation to also let C denote the uniform distribution over its support, and let $C(c) \in [0,1]$ denote the probability that $c \xleftarrow{\$} C$ for $c \in \mathbb{F}^\eta$.

3.6 Small-bias Distributions of Linear Codes with Erasure Recovery

The heart of correlation extractor relies on the construction of a family of small-bias distributions $\{C_j\}_{j\in\mathcal{J}}$, for some set $\mathcal{J}$, such that (1) C_j is a linear code for every j; and (2) $C_j * C_j$ supports erasure recovery for every j. Formally, the requirements of this family of distributions is given in Property 3.6.1.

Property 3.6.1. *A family of linear code distributions $\mathcal{C} = \{C_j\}_{j\in\mathcal{J}}$ over $\mathbb{F}^\eta$ satisfy this property with parameters δ and γ if the following hold.*

1. ***$2^{-\delta/2}$-biased family of distributions**. The code family $\mathcal{C}$ is a $2^{-\delta/2}$-biased family of distributions (see Definition 2.6.2).*
2. ***γ-erasure recovery in Schur product**. For all $j \in \mathcal{J}$, the code $C_j^{(2)} = C_j * C_j$ supports erasure recovery of the first γ coordinates. Moreover, the first γ coordinates of C_j and $C_j^{(2)}$ are linearly independent.*

3.6.1 Our Construction

We present our construction of a family of linear codes which satisfy Property 3.6.1 in Figure 3.2. At a high level, we obtain our family of linear codes by first fixing a suitable algebraic geometric code [84, 121, 133] C instantiated over a constant-size field $\mathbb{F}$. Moreover, the code has block length $\eta^* = \gamma + \eta$, and the parameters of C are chosen appropriately such that (1) under the "twist-then-permute" operation, the constructed family is a $2^{-\delta}$-biased family of distributions; and (2) the code $C * C$ (and thus any permuted version) supports erasure recovery of any γ-coordinates. The linear code family of Figure 3.2 satisfies the following theorem.

Theorem 3.6.2. *The family of linear code distributions $\{C_{\pi,\lambda} \colon \pi \in \mathcal{S}_{\eta^*},\ \lambda \in (\mathbb{F}^\times)^{\eta^*}\}$ over $\mathbb{F}^{\eta^*}$ given in Figure 3.2 satisfies Property 3.6.1 for any $\gamma < d^{(2)}$, where $d^{(2)}$ is the distance of the Schur-product code of C, and*

$$\delta = \left[\left(d^\perp + \frac{\eta*}{\sqrt{q}-1} - 1\right) \cdot \left(\log(q-1) - \mathbf{h}_2\left(\frac{1}{q+1}\right)\right)\right] - \left(\frac{\eta^*}{\sqrt{q}-1}\right) \cdot \log(q).$$

Family of small-bias distributions with erasure recovery in the product distribution:
Fix a linear code $C = [\eta^*, \kappa, d, d^\perp, d^{(2)}]_{\mathbb{F}}$ with generator matrix $G \in \mathbb{F}^{\kappa \times \eta^*}$, where $|\mathbb{F}| = q$ and $\kappa \geqslant d^{(2)}$, where $d^{(2)}$ is the distance of $C * C$. Let γ be a fixed natural number such that $C * C$ supports γ-erasure recovery. We construct the family of small-bias distributions $\{C_{\pi,\lambda} \colon \pi \in \mathcal{S}_{\eta^*},\ \lambda \in (\mathbb{F}^\times)^{\eta^*}\}$ over $\mathbb{F}^{\eta^*}$ as follows.

1. Let $\lambda \in (\mathbb{F}^\times)^{\eta^*}$. Define $G_\lambda = [\lambda_1 \cdot G_1, \ldots, \lambda_{\eta^*} \cdot G_{\eta^*}] \in \mathbb{F}^{\kappa \times \eta^*}$, where G_i is the i^{th} column of G and $\lambda_i \cdot G_i$ is the multiplication of G_i by λ_i.
2. Let $\pi \in \mathcal{S}_{\eta^*}$. Define $G_{\pi,\lambda} = \pi(G_\lambda) \in \mathbb{F}^{\kappa \times \eta^*}$, where $\pi(G_\lambda)$ is the permutation of the columns of G_λ according to permutation π. Then $C_{\pi,\lambda}$ is the uniform distribution over the linear code generated by $G_{\pi,\lambda}$.

$(\mathsf{Enc}, \mathsf{Dec})$ for $C_{\pi,\lambda}$: Let $(\mathsf{Enc}_C, \mathsf{Dec}_C)$ be the Encoder and Decoder for the linear code C.

- $\mathsf{Enc}(m)$: Compute $c = (c_1, \ldots, c_{\eta^*}) = \mathsf{Enc}_C(m)$. Compute $c * \lambda = (\lambda_1 \cdot c_1, \ldots, \lambda_{\eta^*} \cdot c_{\eta^*})$. Output $\pi(c * \lambda)$.
- $\mathsf{Dec}(x)$: Compute $c' = (c'_1, \ldots, c'_{\eta^*}) = \pi^{-1}(x)$. Compute $c' * \lambda' = (\lambda_1^{-1} \cdot c'_1, \ldots, \lambda_{\eta^*}^{-1} \cdot c'_{\eta^*})$. Output $\mathsf{Dec}_C(c' * \lambda')$.

$(\mathsf{Enc}, \mathsf{Dec})$ for $(C_{\pi,\lambda} * C_{\pi,\lambda})$: Let $(\mathsf{Enc}_{C^{(2)}}, \mathsf{Dec}_{C^{(2)}})$ be the Encoder and Decoder for the linear code $C^{(2)} = C * C$.

- $\mathsf{Enc}(m)$: Compute $c = (c_1, \ldots, c_{\eta^*}) = \mathsf{Enc}_{C^{(2)}}(m)$. Compute $c * \lambda * \lambda = (\lambda_1^2 \cdot c_1, \ldots, \lambda_{\eta^*}^2 \cdot c_{\eta^*})$. Output $\pi(c * \lambda * \lambda)$.
- $\mathsf{Dec}(x)$: Compute $c' = (c'_1, \ldots, c'_{\eta^*}) = \pi^{-1}(x)$. Compute $c' * \lambda' * \lambda' = (\lambda_1^{-2} \cdot c'_1, \ldots, \lambda_{\eta^*}^{-2} \cdot c'_{\eta^*})$. Output $\mathsf{Dec}_{C^{(2)}}(c' * \lambda' * \lambda')$.

Figure 3.2. Construction of small-biased linear code distributions.

We prove Theorem 3.6.2 and give an overview of the (asymptotic) calculation of the parameters of C in Section 3.A.1. Since the precise calculation of these parameters are beyond the scope of this dissertation, we refer the curious reader to [50] for these details.

3.7 Correlation Extractor Constructions

We first construct our correlation extractor for the $\mathsf{ROLE}(\mathbb{F})$ correlation, which realizes Theorem 3.2.2. Then we use this correlation extractor to construct our correlation extractor for the ROT correlation, realizing Theorem 3.2.1.

Let $t \in \mathbb{N}$ be a parameter and let $\mathsf{ROLE}(\mathbb{F})^\eta$ be the $\mathsf{ROLE}(\mathbb{F})$ correlation such that each party is given η independent samples of $\mathsf{ROLE}(\mathbb{F})$.

1. **Correlation Generation Phase.** Sender Alice receives (a_i, b_i) and receiver Bob receives $(x_i, z_i = a_i \cdot x_i + b_i)$ for $a_i, b_i, x_i \xleftarrow{\$} \mathbb{F}$ for every $i \in [\eta]$. Let $a = (a_1, \ldots, a_\eta)$, $b = (b_1, \ldots, b_\eta)$, $x = (x_1, \ldots, x_\eta)$ and $z = (z_1, \ldots, z_\eta)$.
2. **Corruption and t-bit Leakage Phase.** An information-theoretic semi-honest adversary corrupts either the sender or the receiver, then sends a leakage function $L \colon \mathbb{F}^\eta \to \{0,1\}^t$, and receives $L(x)$ or $L(a)$, respectively. Note that this leakage is *arbitrary*: it is not limited to leakage on individual bits of the shares and can encode crucial global information.

Figure 3.3. Leakage model for our $\mathsf{ROLE}(\mathbb{F})$-to-$\mathsf{ROLE}(\mathbb{F})$ correlation extractor.

3.7.1 $\mathsf{ROLE}(\mathbb{F})$-to-$\mathsf{ROLE}(\mathbb{F})$ Correlation Extractor

To prove Theorem 3.2.2, we first detour and construct a $\mathsf{ROLE}(\mathbb{F})$-*to*-$\mathsf{ROLE}(\mathbb{F})$ *extractor*—a correlation extractor which initially takes leaky $\mathsf{ROLE}(\mathbb{F})$ shares and outputs secure $\mathsf{ROLE}(\mathbb{F})$ shares. This special correlation extractor is the technical heart of our correlation extractor for $\mathsf{ROLE}(\mathbb{F})$ satisfying Theorem 3.2.2.

Definition 3.7.1 ($\mathsf{ROLE}(\mathbb{F})$-to-$\mathsf{ROLE}(\mathbb{F})$ Correlation Extractor)**.** *An $(\eta, \gamma, t, \varepsilon)$-$\mathsf{ROLE}(\mathbb{F})$-to-$\mathsf{ROLE}(\mathbb{F})$ correlation extractor is a two-party interactive protocol between a sender and a receiver in the $(\mathsf{ROLE}(\mathbb{F})^\eta)^{[t]}$-hybrid that securely implements the $\mathsf{ROLE}(\mathbb{F})^\gamma$ functionality and satisfies the following properties.*

1. ***Secret Shares.*** *Each party receives η independent shares of the $\mathsf{ROLE}(\mathbb{F})$ correlation.*
2. ***Correctness.*** *The receiver's output is correct in all γ instances of $\mathsf{ROLE}(\mathbb{F})$.*
3. ***(t, ε)-Security.*** *We define a simpler leakage model in Figure 3.3 that is equivalent to the model of Figure 3.1. Let (u_i, v_i) and (r_i, z_i) be the output shares of sender Alice and receiver Bob, respectively, in the i^{th} $\mathsf{ROLE}(\mathbb{F})$ instance. Then (t, ε)-security states that a corrupt sender (resp., receiver) who participates in the corruption and t-bit leakage phase of Figure 3.3 cannot distinguish between $\{r_i\}_{i \in [\gamma]}$ (resp., $\{u_i\}_{i \in [\gamma]}$) and $s \xleftarrow{\$} \mathbb{F}^\gamma$ with advantage more than ε.*

Let $\{C_j\}_{j\in\mathcal{J}}$ be a family of linear code distributions over $\mathbb{F}^{\gamma+\eta}$ satisfying Property 3.6.1 for appropriate values of δ and γ.

Hybrid: Client A receives (a, b) for $a, b \xleftarrow{\$} \mathbb{F}^\eta$ and Client B receives (x, z) for $x \xleftarrow{\$} \mathbb{F}^\eta$ such that $z_i = a_i \cdot x_i + b_i$ for every $i \in [\eta]$.

1. *Code Generation.* Client B samples $j \xleftarrow{\$} \mathcal{J}$.
2. $\mathsf{ROLE}(\mathbb{F})$ *Extraction Protocol.*

   (a) Client B samples random codeword $(r_\gamma, \ldots, r_{-1}, r_1, \ldots, r_\eta) \xleftarrow{\$} C_j$ and sets $r = (r_1, \ldots, r_\eta)$. Client B computes $m = r + x \in \mathbb{F}^\eta$ and sends (m, j) to Client A.

   (b) Client A samples random codewords $(u_{-\gamma}, \ldots, u_{-1}, u_1, \ldots, u_\eta) \xleftarrow{\$} C_j$ and $(v_{-\gamma}, \ldots, v_{-1}, v_1, \ldots, v_\eta) \xleftarrow{\$} (C_j * C_j)$. Client A sets $u = (u_1, \ldots, u_\eta)$ and $v = (v_1, \ldots, v_\eta)$ and computes

   $$\alpha = u - a \in \mathbb{F}^\eta \qquad\qquad \beta = (a * m) + b + v \in \mathbb{F}^\eta.$$

   Client A sends (α, β) to Client B.

   (c) Client B computes $t' = (\alpha * r') + \beta - z \in \mathbb{F}^\eta$. Client B performs erasure recovery on t' for code $C_j * C_j$ and obtains $t'' = (t_{-\gamma}, \ldots, t_{-1}) \in \mathbb{F}^\gamma$.

   (d) Client A outputs $\{(u_i, v_i)\}_{i\in\{-\gamma,\ldots,-1\}}$ and Client B outputs $\{(r_i, t_i)\}_{i\in\{-\gamma,\ldots,-1\}}$.

Figure 3.4. Our $\mathsf{ROLE}(\mathbb{F})$-to-$\mathsf{ROLE}(\mathbb{F})$ Correlation Extractor.

We present our $\mathsf{ROLE}(\mathbb{F})$-to-$\mathsf{ROLE}(\mathbb{F})$ correlation extractor in Figure 3.4, which is inspired by the Massey secret sharing scheme [197]. The protocol is message-optimal (i.e., we send 2 messages, one from each party), and uses a family of linear code distributions $\{C_j\}_{j\in\mathcal{J}}$ which satisfy Property 3.6.1 for appropriate δ and γ.

We first prove the correctness of Figure 3.4, which is characterized by the following lemma.

Lemma 3.7.1. *If $\{C_j\}_{j\in\mathcal{J}}$ satisfies Property 3.6.1, then for all $i \in \{-\gamma, \ldots, -1\}$, it holds that $t_i = u_i \cdot r_i + v_i$.*

Proof. First, we prove the following claim.

Claim 3.7.2. *For all $i \in [\eta]$, it holds that $t'_i = u_i \cdot r_i + v_i$.*

This claim follows from the following derivation.

$$\begin{aligned} t'_i &= \alpha_i r_i + \beta_i - z_i = (u_i - a_i) r_i + (a_i m_i + b_i + v_i) - z_i \\ &= u_i r_i - a_i r_i + a_i (r_i + x_i) + b_i + v_i \\ &= u_i r_i + a_i x_i + b_i + v_i - z_i = u_i r_i + v_i. \end{aligned}$$

From the protocol, we have that $(u', u) \in C_j$, $(r', r) \in C_j$, and $(v', v) \in C_j^{(2)}$ for $u' = (u_{-\gamma}, \ldots, u_{-1})$, $r' = (r_{-\gamma}, \ldots, r_{-1})$, and $v' = (v_{-\gamma}, \ldots, v_{-1})$. Consider $\tilde{t} = (u', u) * (r' * r) + (v', v) \in C_j^{(2)}$. Note that $t'_i = \tilde{t}_i$ for all $i \in [\eta]$. Hence, when client B performs erasure recovery on t' for a codeword in $C_j^{(2)}$, it obtains $t'' = (\tilde{t}_{-\gamma}, \ldots, \tilde{t}_{-1})$. This follows from erasure recovery guarantee for any γ coordinates by Property 3.6.1. □

Next we capture the (t, ε)-security of Theorem 3.2.2 in the following lemma.

Lemma 3.7.3. *After the correlation generation and t-bit leakage phase of Figure 3.3, the protocol presented in Figure 3.4 has simulation error $\varepsilon \leqslant \sqrt{\frac{|\mathbb{F}|^\gamma \cdot 2^t}{2^\delta}}$, where γ and δ are the parameters for the family $\{C_j\}_{j \in \mathcal{J}}$ satisfying Property 3.6.1.*

We prove Lemma 3.7.3 in Section 3.A.1. We use Lemmas 3.7.1 and 3.7.3 to aid in the proof of Theorem 3.2.2, presented next.

3.7.2 Correlation Extractor for ROLE($\mathbb{F}$) (Theorem 3.2.2)

We now construct our ROLE($\mathbb{F}$) correlation extractor which satisfies Theorem 3.2.2; that is, given leaky shares of ROLE($\mathbb{F}$), our correlation extractor outputs fresh samples of ROT. In particular, we compose the ROLE($\mathbb{F}$)-to-ROLE($\mathbb{F}$) correlation extractor of Figure 3.4 with the *(random) oblivious transfer embedding protocol* of Block, Maji, and Nguyen [54]. We refer to this embedding protocol as the BMN embedding protocol. This protocol embeds a constant number of ROT samples into one sample of ROLE($\mathbb{F}$), where $\mathbb{F}$ is a finite field of characteristic 2. The BMN embedding protocol is a two-message perfectly semi-honest secure protocol that, asymptotically, embeds $(s)^{1-o(1)}$ samples of ROT into one sample of the ROLE($\mathbb{F}_{2^s}$) correlation. However, for reasonable values of s, say for $s \leqslant 2^{50}$, a recursive embedding

embeds $s^{\log(10)/\log(38)}$ samples of ROT into one sample of the $\mathsf{ROLE}(\mathbb{F}_{2^s})$ correlation, and this embedding is more efficient than the asymptotically good one. We argue that this protocol composes in parallel with our protocol in Figure 3.4 to give our overall round optimal protocol for (n, m, t, ε)-correlation extractor for $\mathsf{ROLE}(\mathbb{F})$ correlation satisfying Theorem 3.2.2.

We note that the BMN embedding protocol satisfies the following additional properties. (1) The first message is sent by Client B; and (2) this message depends only on the first share of Client B in $\mathsf{ROLE}(\mathbb{F})$ (this refers to r_i in Figure 3.4) and does not depend on the second share (this refers to t_i in Figure 3.4). With these properties, the BMN embedding protocol can be run in parallel with the protocol in Figure 3.4, maintaining message complexity 2. Also, since the BMN protocol satisfies perfect correctness and perfect security, to prove overall security, it suffices to prove the correctness and security of our protocol in Figure 3.4. This holds because we are in the semi-honest information theoretic setting. Thus by Lemmas 3.7.1 and 3.7.3 and the above observation, we have established Theorem 3.2.2. We present the correlation extractor satisfying Theorem 3.2.2 in Figure 3.5.

3.7.3 Correlation Extractor for ROT (Theorem 3.2.1)

To obtain a correlation extractor for ROT satisfying Theorem 3.2.1, we describe a protocol which constructs shares of the leaky correlation $(\mathsf{ROLE}(\mathbb{F})^\eta)^{[t]}$ from shares of the leaky correlation $(\mathsf{ROLE}^n)^{[t]}$. The leaky shaers of the $(\mathsf{ROLE}(\mathbb{F})^\eta)^{[t]}$ correlation are then used in the correlation extractor of Theorem 3.2.2 (described in Section 3.7.2). This gives us our final ROT correlation extractor satisfying Theorem 3.2.1 since ROLE and ROT are equivalent (see Fact 3.5.1). Recall that $\mathsf{ROLE} := \mathsf{ROLE}(\mathbb{F}_2)$.

This protocol relies on one of many applications of algebraic functions fields pioneered by the seminal work of Chudnovsky and Chudnovsky [94]. The application we are interested in is efficiently computing multiplications over an extension field via multiplications over the base field. For example, 6 multiplications over $\mathbb{F}_2$ suffices to perform a single multiplication over $\mathbb{F}_{2^3}$ (cf., Table 1 in [80] for more examples). This application is crucial for our final correlation extractor.

Let $\{C_j\}_{j\in\mathcal{J}}$ be a family of linear code distributions over $\mathbb{F}^{\gamma+\eta}$ satisfying Property 3.6.1 for appropriate values of δ and γ, where $\mathbb{F}$ is a finite field of characteristic 2.
Let $\mathsf{Enc}_{\mathrm{BMN}}\colon \{0,1\}^{\ell} \to \mathbb{F}$ and $\mathsf{Dec}_{\mathrm{BMN}}\colon \mathbb{F} \to \{0,1\}^{\ell}$ be the encoding and decoding algorithms for the BMN Embedding [54] for appropriate $\ell \in \mathbb{N}$.

Hybrid: Client A receives (a,b) for $a,b \xleftarrow{\$} \mathbb{F}^{\eta}$ and Client B receives (x,z) for $x \xleftarrow{\$} \mathbb{F}^{\eta}$ such that $z_i = a_i \cdot x_i + b_i$ for every $i \in [\eta]$.

1. *Code Generation.* Client B samples $j \xleftarrow{\$} \mathcal{J}$.
2. ROLE *Extraction Protocol.*

   (a) Client B samples $x^{(i)} \xleftarrow{\$} \{0,1\}^{\ell}$ and sets $r_i = \mathsf{Enc}_{\mathrm{BMN}}(x^{(i)}) \in \mathbb{F}$ for every $i \in \{-\gamma,\dots,-1\}$. Client B samples random codeword $(r_{-\gamma},\dots,r_{-1},r_1,\dots,r_\eta) \xleftarrow{\$} C_j$ and sets $r = (r_1,\dots,r_\eta)$. Client B computes $m = r + x \in \mathbb{F}^{\eta}$ and sends (m,j) to Client A.

   (b) Client A samples $a^{(i)} \xleftarrow{\$} \{0,1\}^{\ell}$ and $b^{(i)} \xleftarrow{\$} \{0,1\}^{\ell}$ and sets $u_i = \mathsf{Enc}_{\mathrm{BMN}}(a^{(i)}) \in \mathbb{F}$ and $v_i = \mathsf{Enc}_{\mathrm{BMN}}(b^{(i)}) \in \mathbb{F}$ for every $i \in \{-\gamma,\dots,-1\}$. Client A samples random codewords $(u_{-\gamma},\dots,u_{-1},u_1,\dots,u_\eta) \xleftarrow{\$} C_j$ and $(v_{-\gamma},\dots,v_{-1},v_1,\dots,v_\eta) \xleftarrow{\$} (C_j * C_j)$. Client A sets $u = (u_1,\dots,u_\eta)$ and $v = (v_1,\dots,v_\eta)$ and computes

   $$\alpha = u - a \in \mathbb{F}^{\eta} \qquad\qquad \beta = (a * m) + b + v \in \mathbb{F}^{\eta}.$$

   Client A sends (α,β) to Client B.

   (c) Client B computes $t' = (\alpha * r') + \beta - z \in \mathbb{F}^{\eta}$. Client B performs erasure recovery on t' for code $C_j * C_j$ and obtains $t'' = (t_{-\gamma},\dots,t_{-1}) \in \mathbb{F}^{\gamma}$.

   (d) Client A outputs $\{(\mathsf{Dec}_{\mathrm{BMN}}(u_i), \mathsf{Dec}_{\mathrm{BMN}}(v_i))\}_{i\in\{-\gamma,\dots,-1\}}$ and Client B outputs $\{(\mathsf{Dec}_{\mathrm{BMN}}(r_i), \mathsf{Dec}_{\mathrm{BMN}}(t_i))\}_{i\in\{-\gamma,\dots,-1\}}$.

Figure 3.5. Our correlation extractor for $\mathsf{ROLE}(\mathbb{F})$.

The first step of our final correlation extractor will be, as described above, to use the efficient multiplication algorithms to (perfectly and securly) implement $(\mathsf{ROLE}(\mathbb{F})^{\eta})^{[t]}$ using $(\mathsf{ROLE}^{n})^{[t]}$, where here $|\mathbb{F}| = 2^k$ for some $k \in \mathbb{N}$. We start by giving a protocol for realizing a single $\mathsf{ROLE}(\mathbb{F})$ instance using ROLE^{ℓ}; i.e., ℓ independent samples of ROLE with no leakage applied. The protocol is given in Figure 3.6. The protocol of Figure 3.6 relies on a multiplication friendly code $\mathcal{D} \subseteq \{0,1\}^{\ell}$ with message space $\mathbb{F}$; in particular, $\mathcal{D}^{(2)} := \mathcal{D} * \mathcal{D} \subseteq \{0,1\}^{\ell}$ is also a code with message space $\mathbb{F}$.

The presented protocol of Figure 3.6 is a perfectly secure implementation of $\mathsf{ROLE}(\mathbb{F})$ in the ROLE^{ℓ}-hybrid against a semi-honest adversary; this follows from the fact that $\mathcal{D}$ is a

Let $\mathcal{D} \subseteq \{0,1\}^\ell$ be a multiplication friendly code with message space $\mathbb{F} = \mathbb{F}_{2^\alpha}$. Let $(\mathsf{Enc}_\mathcal{D}, \mathsf{Dec}_\mathcal{D})$ (resp., $\mathsf{Enc}_{\mathcal{D}^{(2)}}, \mathsf{Dec}_{\mathcal{D}^{(2)}}$) be encoding and decoding procedures for $\mathcal{D}$ (resp., $\mathcal{D}^{(2)}$).

Hybrid ROLE^ℓ: Client A receives (a, b) for $a, b \xleftarrow{\$} \{0,1\}^\ell$ and Client B receives (x, z) for $x \xleftarrow{\$} \{0,1\}^\ell$ such that $z_i = a_i \cdot x_i + b_i$ for every $i \in [\ell]$.

1. Client B samples $x_0 \xleftarrow{\$} \mathbb{F}$ and obtains random codeword $\tilde{x} = \mathsf{Enc}_\mathcal{D}(x_0) \in \{0,1\}^\ell$. Client B sets $m = x + \tilde{x}$ and sends m to Client A.
2. Client A samples $a_0, b_0 \xleftarrow{\$} \mathbb{F}$ and obtains random codewords $\tilde{a} = \mathsf{Enc}_\mathcal{D}(a_0) \in \{0,1\}^\ell$ and $b = \mathsf{Enc}_{\mathcal{D}^{(2)}}(b_0) \in \{0,1\}^\ell$. Client A sets
$$\alpha = a + \tilde{a} \qquad\qquad \beta = (a * m) + b + \tilde{b}$$
and sends (α, β) to Client B.
3. Client B computes $\tilde{z} = (\alpha * \tilde{x}) + \beta + z$. Client B computes $z_0 = \mathsf{Dec}_{\mathcal{D}^{(2)}}(\tilde{z})$.
4. Client A outputs (a_0, b_0) and Client B outputs (x_0, z_0).

Figure 3.6. Perfectly secure protocol for $\mathsf{ROLE}(\mathbb{F})$ in ROLE^ℓ-hybrid.

multiplication friendly code. To extend this guarantee to the t-leaky setting, we utilize the following lemma.

Lemma 3.7.4 ([158])**.** *Let Π be a perfectly secure (resp., ε-statistically secure) implementation of functionality f in the g-hybrid model, where Π makes a single call to the functionality g. Then Π is a perfectly secure (resp., ε-statistically secure) implementation of $f^{[t]}$ in the $g^{[t]}$-hybrid model.*

By Lemma 3.7.4, the protocol presented in Figure 3.6 is a prefect realization of $(\mathsf{ROLE}(\mathbb{F}))^{[t]}$ in the $(\mathsf{ROLE}^\ell)^{[t]}$-hybrid. Running the protocol of Figure 3.6 in parallel for η samples of the $\mathsf{ROLE}(\mathbb{F})$ correlation, we obtain a perfectly secure protocol for $(\mathsf{ROLE}(\mathbb{F})^\eta)^{[t]}$ in the $(\mathsf{ROLE}^{\eta\cdot\ell})^{[t]}$-hybrid.

Finally, we establish the message optimality of running the protocol of Figure 3.6 in parallel with our correlation extractor of Figure 3.5. The full protocol is presented in Figure 3.7. This realizes Theorem 3.2.1.

3.A Correlation Extractors: Deferred Proofs

We present proofs deferred from prior sections.

3.A.1 Proof of Theorem 3.6.2

We prove the erasure recovery property followed by the small-bias property.

Erasure recovery of γ indices in the Schur-product code. First we note that permuting or re-ordering the columns of a generator matrix does not change its distance, the distance of the Schur-product, or erasure recovery capability (as long as we know the mapping of new columns vis-à-vis old columns). Let $\mathcal{I}_\gamma = \{i_1, \ldots, i_\gamma\}$ be the indices of the erased coordinates of codeword in $C^{(2)}_{\pi,\lambda}$. Hence to show erasure recovery of the coordinates $\mathcal{I}_\gamma$ of a codeword of $C^{(2)}_{\pi,\lambda}$, it suffices to show erasure recovery of the γ erased coordinates $\mathcal{J}_\gamma = \{j_1, \ldots, j_\gamma\}$ of a codeword of $C^{(2)}_{\lambda}$, where C_λ is the uniform codespace generated by G_λ, and $\pi(j_k) = i_k$ for every $k \in [\gamma]$.

Note that since $\gamma < d^{(2)}$, the code $C^{(2)}$ supports erasure recovery of *any* γ coordinates. Thus it suffices to show that this implies that $C^{(2)}_\lambda$ also supports the erasure recovery of *any* γ coordinates. Note that since $\lambda \in (\mathbb{F}^*)^{\eta^*}$, multiplication of the columns of G according to λ does not change its distance or distance of the Schur-product code. Then we do the following to perform erasure recovery of γ coordinates in $C^{(2)}_\lambda$. Let $c^{(2)} \in C^{(2)}_\lambda$ be a codeword with erased coordinates $\mathcal{J}_\gamma = \{j_1, \ldots, j_\gamma\}$, and let $\mathcal{J}_\eta = \{j'_1, \ldots, j'_\eta\}$ be the coordinates of $c^{(2)}$ that have not been erased. For every $j \in \mathcal{J}_\eta$, compute $c_j = (\lambda_j^{-1})^2 \cdot c^{(2)}_j$. Then the vector $(c_j)_{j \in \mathcal{J}_\eta}$ is a codeword of $C^{(2)}$ with coordinates c_i erased for $i \in \mathcal{J}_\gamma$. Since $C^{(2)}$ has γ erasure recovery, we can recover the c_i for $i \in \mathcal{J}_\gamma$. Once recovered, for every $i \in \mathcal{J}_\gamma$, compute $c^{(2)}_i = \lambda_i^2 \cdot c_i$. This produces the γ erased coordinates of $c^{(2)}$ in $C^{(2)}_\lambda$. Finally, one can map the $c^{(2)}_i$ for $i \in \mathcal{J}_\gamma$ to the coordinates $\mathcal{I}_\gamma$ using π, recovering the erasures in $C^{(2)}_{\pi,\lambda}$.

$2^{-\delta}$-bias family of distributions. Let $C, C_\lambda, C_{\pi,\lambda}$ be the uniform distribution over the linear codes generated by $G, G_\lambda, G_{\pi,\lambda}$, respectively. Recall that $d^\perp$ is the dual distance for C.

Note that $C_\lambda, C_{\pi,\lambda}$ have dual-distance $d^\perp$ as well. Let $\eta^* = \eta + \gamma$. Since for every $S \in \mathbb{F}^{\eta^*}$ we have $\mathrm{Bias}_S(C_{\pi,\lambda}) = |\mathbb{F}|^{\eta^*}| \cdot |\widehat{C}_{\pi,\lambda}(S)|$, it suffices to show that

$$\mathop{\mathbb{E}}_{\pi,\lambda}\left[\widehat{C}_{\pi,\lambda}(S)^2\right] \leqslant \frac{1}{|\mathbb{F}|^{2\eta^*} \cdot 2^\delta}.$$

To begin, first recall the definition of $C_{\pi,\lambda}$:

$$C_{\pi,\lambda} := \{\pi(\lambda_1 \cdot c_1, \ldots, \lambda_{\eta^*} \cdot c_{\eta^*}) \mid (c_1, \ldots, c_{\eta^*}) \in C\}.$$

Next, given any $S \in \mathbb{F}^{\eta^*}$, define $\mathbb{S}(S) := \{\pi(\lambda_1 S_1, \ldots, \lambda_{\eta^*} S_{\eta^*}) \in \mathbb{F}^{\eta^*} \mid \forall \pi \in \mathcal{S}_{\eta^*} \wedge \lambda \in (\mathbb{F}^\times)^{\eta^*}\}$. Note that $\mathbb{S}(S)$ is equivalently characterized as

$$\mathbb{S}(S) = \{T = (T_1, \ldots, T_{\eta^*}) \in \mathbb{F}^{\eta^*} \mid \mathsf{wt}(T) = \mathsf{wt}(S)\}.$$

Notice that $|\mathbb{S}(S)| = \binom{\eta^*}{w_0} \cdot (q-1)^{\eta^* - w_0}$, where $w_0 = \eta^* - \mathsf{wt}(S)$; i.e., w_0 is the number of zeros in S. We prove the following claim.

Claim 3.A.1. *For any $S \in \mathbb{F}^{\eta^*}$, we have $\widehat{C}_{\pi,\lambda}(S) = \widehat{C}(\pi^{-1}(S) * \lambda)$.*

Proof. Notice that by definition for any $x \in C_{\pi,\lambda}$, we have $C_{\pi,\lambda}(x) = C(c)$ since $x = \pi(\lambda_1 \cdot c_1, \ldots, \lambda_{\eta^*} \cdot c_{\eta^*})$ for $c \in C$. This is equivalently stated as $C_{\pi,\lambda}(\pi(c * \lambda)) = C(c)$. For $x = \pi(\lambda_1 \cdot y_1, \ldots, \lambda_{\eta^*} \cdot y_{\eta^*}) \in \mathbb{F}^{\eta^*}$ and any $S \in \mathbb{F}^{\eta^*}$, we have

$$S \cdot x = \sum_{i=1}^{\eta^*} S_i \cdot x_i = \sum_{i=1}^{\eta^*} S_i \cdot (\lambda_{\pi(i)} \cdot y_{\pi(i)}) = \sum_{i=1}^{\eta^*} (S_{\pi^{-1}(i)}) \cdot \lambda_i \cdot y_i = (\pi^{-1}(S) * \lambda) \cdot y.$$

where $S \cdot x$ is the vector dot product. By definition of $\chi_S(x)$, this implies $\chi_S(x) = \chi_y(\pi^{-1}(S) * \lambda)$. Using these two facts and working directly from the definition of Fourier Transform, we have

$$\begin{aligned}
\widehat{C}_{\pi,\lambda}(S) &= \frac{1}{|\mathbb{F}|^{\eta^*}} \cdot \sum_{x \in \mathbb{F}^{\eta^*}} C_{\pi,\lambda}(x) \cdot \overline{\chi_S(x)} \\
&= \frac{1}{|\mathbb{F}|^{\eta^*}} \cdot \sum_{c \in \mathbb{F}^{\eta^*}} C_{\pi,\lambda}(\pi(\lambda_1 c_1, \ldots, \lambda_{\eta^*} c_{\eta^*})) \cdot \overline{\chi_S(\pi(\lambda_1 c_1, \ldots, \lambda_{\eta^*} c_{\eta^*}))} \\
&= \frac{1}{|\mathbb{F}|^{\eta^*}} \cdot \sum_{c \in \mathbb{F}^{\eta^*}} C(c) \cdot \overline{\chi_c(\pi^{-1}(S) * \lambda)} = \widehat{C}(\pi^{-1}(S) * \lambda).
\end{aligned}$$

□

Notice that $\mathsf{wt}(\pi^{-1}(S)*\lambda) = \mathsf{wt}(S)$ for any $S \in \mathbb{F}^{\eta^*}$ and $\lambda \in (\mathbb{F}^\times)^{\eta^*}$. Therefore $\pi^{-1}(S)*\lambda = T \in \mathbb{S}(S)$. From this observation and Claim 3.A.1, we prove the following claim.

Claim 3.A.2. *For any $S \in \mathbb{F}^n$, $\mathop{\mathbb{E}}_{\pi,\lambda}\left[\widehat{C}_{\pi,\lambda}(S)^2\right] = \mathop{\mathbb{E}}_{T \xleftarrow{\$} \mathbb{S}(S)}\left[\widehat{C}(T)^2\right]$.*

Proof. Suppose we have codeword $x \in C_{\pi,\lambda}$ such that $\pi(\lambda_1 \cdot c_1, \ldots, \lambda_\eta^* \cdot c_\eta^*) = x$, for some codeword $c \in C$. Let $\{i_1, \ldots, i_{w_0}\}$ be the set of indices of 0 in c; that is, $c_j = 0$ for all $j \in \{i_1, \ldots, i_{w_0}\}$. Then for any permutation π, the set $\{\pi(i_0), \ldots, \pi(i_{w_0})\}$ is the set of zero indices in x. Note also that for any index $j \notin \{\pi(i_0), \ldots, \pi(i_{w_0})\}$, we have $x_j \neq 0$. If this was not the case, then we have $x_j = c_{\pi^{-1}(j)} \cdot \lambda_{\pi^{-1}(j)} = 0$. Since $j \notin \{\pi(i_0), \ldots, \pi(i_{w_0})\}$, this implies $\pi^{-1}(j) \notin \{i_0, \ldots, i_{w_0}\}$, which further implies that $c_{\pi^{-1}(j)} \neq 0$. This is a contradiction since $\lambda \in (\mathbb{F}^\times)^{\eta^*}$. Thus any permutation π must map the zeros of S to the zeros of c, and there are $w_0! \cdot (\eta^* - w_0)!$ such permutations. Notice now that for any $c_k = 0$, λ_k can take any value in $\mathbb{F}^\times$, so we have $(q-1)^{w_0}$ such choices. Furthermore, if $c_k \neq 0$ and $\lambda_k \cdot c_k = x_{\pi^{-1}(k)} \neq 0$, then there is exactly one value $\lambda_k \in \mathbb{F}^\times$ which satisfies this equation. Putting it all together, we have

$$\begin{aligned}
\mathop{\mathbb{E}}_{\pi,\lambda}\left[\widehat{C_{\pi,\lambda}}(S)^2\right] &= \frac{1}{\eta^*! \cdot (q-1)^{\eta^*}} \cdot \sum_{\pi,\lambda} \widehat{C_{\pi,\lambda}}(S)^2 && (3.3)\\
&= \frac{1}{\eta^*! \cdot (q-1)^{\eta^*}} \cdot \sum_{\pi,\lambda} \widehat{C}\left(\pi^{-1}(S) * \lambda\right)^2 && (3.4)\\
&= \frac{(w_0! \cdot (\eta^* - w_0)! \cdot (q-1)^{w_0})}{\eta^*! \cdot (q-1)^{\eta^*}} \cdot \sum_{T \in \mathbb{S}(S)} \widehat{C}(T)^2 && (3.5)\\
&= \frac{w_0! \cdot (\eta^* - w_0)!}{\eta^*! \cdot (q-1)^{\eta^* - w_0}} \cdot \sum_{T \in \mathbb{S}(S)} \widehat{C}(T)^2 && (3.6)\\
&= \left(\binom{\eta^*}{w_0} \cdot (q-1)^{\eta^* - w_0}\right)^{-1} \cdot \sum_{T \in \mathbb{S}(S)} \widehat{C}(T)^2 = \mathop{\mathbb{E}}_{T \xleftarrow{\$} \mathbb{S}(S)}\left[\widehat{C}(T)^2\right], && (3.7)
\end{aligned}$$

where Equation (3.4) follows from Claim 3.A.1. □

Given Claim 3.A.2, we turn to finding δ such that for $S \in \mathbb{F}^{\eta^*} \setminus \{0^{\eta^s}\}$, we have

$$\mathop{\mathbb{E}}_{T \xleftarrow{\$} \mathbb{S}(S)}\left[\widehat{C}(T)^2\right] \leqslant \frac{1}{|\mathbb{F}|^{2\eta^*} \cdot 2^\delta}.$$

Since C is a linear code, it has non-zero Fourier coefficients only at codewords in the dual code $C^{\perp}$.

Fact 3.A.3. *For all* $S \in \mathbb{F}^{\eta^*}$, $\widehat{C}(S) = \begin{cases} \frac{1}{|\mathbb{F}|^{\eta^*}} & S \in C^{\perp} \\ 0 & \text{otherwise.} \end{cases}$

Let $A_w = |C^{\perp} \cap \mathbb{S}(S)|$, where $w = \eta^* - w_0 = \mathsf{wt}(S)$. Intuitively, A_w is the number of codewords in $C^{\perp}$ with weight w. Then from Fact 3.A.3, we have

$$\begin{aligned} \mathop{\mathbb{E}}_{T \xleftarrow{\$} \mathbb{S}(S)} \left[\widehat{C}(T)^2\right] &= \frac{|C^{\perp} \cap \mathbb{S}(S)|}{|\mathbb{F}|^{2\eta^*} \cdot \binom{\eta^*}{\eta^* - w_0} \cdot (q-1)^{\mathsf{wt}(S)}} = \frac{A_w}{|\mathbb{F}|^{2\eta^*} \cdot \binom{\eta^*}{w} \cdot (q-1)^{w}} \\ &= \frac{A_w}{|\mathbb{F}|^{2\eta^*} \cdot |\mathbb{S}(S)|}. \end{aligned} \tag{3.8}$$

Now, our goal is to upper bound A_w. Towards this goal, the weight enumerator for the code $C^{\perp}$ is defined as the polynomial

$$W_{C^{\perp}}(x) = \sum_{c \in C^{\perp}} x^{\eta^* - \mathsf{wt}(c)}.$$

This polynomial can equivalently be written as

$$W_{C^{\perp}}(x) = \sum_{w \in [\eta^*]_0} A_w \cdot x^{\eta^* - w}.$$

Define $a = \eta^* - d^{\perp}$.

Proposition 3.A.1 (Exercise 1.1.15 from [254])**.** *We have the following relation*

$$W_{C^{\perp}}(x) = x^{\eta^*} + \sum_{i=0}^{a} B.i(x-1)^i,$$

where

$$B_i = \sum_{j=\eta^*-a}^{\eta^*-i} \binom{\eta^* - j}{i} \cdot A_j \geqslant 0 \qquad A_i = \sum_{j=\eta^*-i}^{a} (-1)^{\eta^*+i+j} \binom{j}{\eta^* - i} \cdot B_j.$$

For weight $w \in \left\{d^{\perp}, \ldots, \eta^*\right\}$, we use the following expression to estimate A_w.

$$A_w = \binom{\eta^* - w}{\eta^* - w} \cdot B_{\eta^* - w} - \binom{\eta^* - w + 1}{\eta^* - w} \cdot B_{\eta^* - w + 1} + \cdots \pm \binom{\eta^* - d^{\perp}}{\eta^* - w} \cdot B_{\eta^* - d^{\perp}}$$

We are interested in the asymptotic behavior of A_w, and observe that asymptotically we have $\log(A_w) \sim \log(\Gamma(w))$, where

$$\Gamma(w) := \max_{\eta^* - w \leqslant j \leqslant \eta^* - d^{\perp}} \left\{ \binom{j}{\eta^* - w} \cdot B_j \right\} \tag{3.9}$$

Thus it suffices to compute $\Gamma(w)$ for every $w \in \{d^{\perp}, \ldots, \eta^*\}$.

We are interested in computing the maximum bias $2^{-\delta}$ such that

$$\mathop{\mathbb{E}}_{T \xleftarrow{\$} \mathbb{S}(S)} \left[\widehat{C}(T)^2\right] = \frac{A_w}{|\mathbb{F}|^{2\eta^*} \cdot |\mathbb{S}(S)|} \leqslant \frac{1}{|\mathbb{F}|^{2\eta^*} \cdot 2^{\delta}},$$

where $w = \mathsf{wt}(S)$ and the above equation is given by Equation (3.8). Recalling that $\log(A_w) \sim \log(\Gamma(w))$, by the above equation we are interested in the quantity

$$\delta \geqslant \min_{d^{\perp} \leqslant w \leqslant \eta^*} \log \left(\frac{|\mathbb{S}(w)|}{\Gamma(w)} \right). \tag{3.10}$$

Here we overload notation and let $\mathbb{S}(w) \subseteq \mathbb{F}^{\eta^*}$ be the set of all vectors of weight w. Note that for any $S \in \mathbb{F}^{\eta^*}$ of weight w, we have $\mathbb{S}(S) = \mathbb{S}(w)$. Computing the minimum value of δ in Equation (3.10) corresponds to an (asymptotic) upper bound on the bias of our family of linear code distributions. We defer the reader to the full version of [49] at [50] for complete details on the final derivation of δ that achieves Theorem 3.6.2.

3.A.2 Proof of Lemma 3.7.3

Lemma 3.7.3. *After the correlation generation and t-bit leakage phase of Figure 3.3, the protocol presented in Figure 3.4 has simulation error $\varepsilon \leqslant \sqrt{\frac{|\mathbb{F}|^{\gamma} \cdot 2^t}{2^{\delta}}}$, where γ and δ are the parameters for the family $\{C_j\}_{j \in \mathcal{J}}$ satisfying Property 3.6.1.*

We shall reduce the security of our protocol to the following unpredicatbility lemma.

Lemma 3.A.4 (Unpredictability Lemma [49, 54])**.** *Let $\{C_j\}_{j\in\mathcal{J}}$ be a $2^{-\delta}$-biased family of linear code distributions over $\mathbb{F}^{\eta^*}$ for $\eta^* = \gamma + \eta$. Consider the following game between an honest challenger $\mathcal{H}$ and an adversary $\mathcal{A}$:*

1. *$\mathcal{H}$ samples $m \xleftarrow{\$} \mathbb{F}^\eta$.*
2. *$\mathcal{A}$ sends leakage function $\mathcal{L}\colon \mathbb{F}^\eta \to \{0,1\}^t$ to $\mathcal{H}$.*
3. *$\mathcal{H}$ sends $\mathcal{L}(m)$ to $\mathcal{A}$.*
4. *$\mathcal{H}$ samples $j \xleftarrow{\$} \mathcal{J}$. $\mathcal{H}$ samples $(r_{-\gamma},\ldots,r_{-1},r_1,\ldots,r_\eta) \xleftarrow{\$} C_j$ and lets $r = (r_1,\ldots,r_\eta)$. $\mathcal{H}$ computes $y = r + m \in \mathbb{F}^\eta$. $\mathcal{H}$ samples $b \xleftarrow{\$} \{0,1\}$. If $b = 0$ then $\mathcal{H}$ sets $\mathsf{chal} = r'$ for $r' = (r_{-\gamma},\ldots,r_{-1})$; else if $b = 1$ then $\mathcal{H}$ sets $\mathsf{chal} = u \xleftarrow{\$} \mathbb{F}^\gamma$. $\mathcal{H}$ sends (y, j, chal) to $\mathcal{A}$.*
5. *$\mathcal{A}$ sends $\tilde{b} \in \{0,1\}$ to $\mathcal{H}$.*

The adversary $\mathcal{A}$ wins the game if $b = \tilde{b}$. Any adversary $\mathcal{A}$ wins the above game with advantage $\varepsilon \leqslant \frac{1}{2}\cdot\sqrt{\frac{|\mathbb{F}|^\gamma\cdot 2^t}{2^\delta}}$.

Given Lemma 3.A.4, we establish the (t,ε)-security of Figure 3.4 in the following lemma.

Lemma 3.A.5. *The simulation error of our protocol is $\varepsilon \leqslant \sqrt{\frac{|\mathbb{F}|^\gamma 2^t}{2^\delta}}$ after t bits of leakage, where γ and δ are the parameters for family of distributions $\mathcal{C}$ provided by Property 3.6.1.*

Proof. We proceed by proving the security of Bob followed by the security of Alice. For the security of Bob, assume that Alice is semi-honest. It suffices to show that Alice cannot distinguish between Bob's secret values $(r_{-\gamma},\ldots,r_{-1})$ and $(\xi_1,\ldots,\xi_\gamma) \xleftarrow{\$} \mathbb{F}^\gamma$ except with probability at most ε. Observe that the security of Bob directly reduces to Lemma 3.A.4 for the following values. Let $X_{[\eta]}$ be the random variable of Bob's initial input $x_{[\eta]}$. Then $X_{[\eta]} \equiv U_{\mathbb{F}^\eta}$. Before the protocol proceeds, adversary Alice obtains at most t bits of leakage $L = \mathcal{L}(X_{[\eta]}) \in \{0,1\}^*$. Then Bob samples $j \xleftarrow{\$} \mathcal{J}$ and $r = (r_{-\gamma},\ldots,r_{-1},r_1,\ldots,r_\eta) \xleftarrow{\$} C_j$, and sends $m_{[\eta]} = r_{[\eta]} + x_{[\eta]}$ to Alice. This is identical to the game between a semi-honest adversary $\mathcal{A}$ and an honest challenger $\mathcal{H}$ in Lemma 3.A.4. This implies that Alice cannot distinguish between $(r_{-\gamma},\ldots,r_{-1})$ and $(\xi_1,\ldots,\xi_\gamma) \xleftarrow{\$} \mathbb{F}^\gamma$ except with probability at most ε.

For the security of Alice, assume that Bob is semi-honest. We again reduce the security to Lemma 3.A.4 to show that Bob cannot distinguish between Alice's secret values $(u_{-\gamma},\ldots,u_{-1})$

and $(\xi_1, \ldots, \xi_\gamma) \xleftarrow{\$} \mathbb{F}^\gamma$ except with probability ε, however it is slightly more complicated in this case. Let $A_{[\eta]}$ be the random variable for Alice's initial input $a_{[\eta]}$. Without loss of generality, Bob receives at most t bits of leakage $L = \mathcal{L}(A_{[\eta]})$. Suppose by way of contradiction that Bob can distinguish between Alice's secret and the uniform distribution with probability greater than ε. Let $\mathcal{A}$ denote Bob. Then we construct an adversary $\mathcal{A}'$ using $\mathcal{A}$ that can win the game of Lemma 3.A.4 with probability greater than ε. Let $\mathcal{C} = \{C_j\}_{j \in \mathcal{J}}$ be our family of linear code distributions.

1. $\mathcal{H}$ samples $a_{[\eta]} \xleftarrow{\$} \mathbb{F}^\eta$.
2. $\mathcal{A}'$ samples $x_{[\eta]}, z_{[\eta]} \xleftarrow{\$} \mathbb{F}^\eta$ and forwards $(x_{[\eta]}, z_{[\eta]})$ to $\mathcal{A}$.
3. Upon receiving leakage function $\mathcal{L}$ from $\mathcal{A}$, $\mathcal{A}'$ forwards $\mathcal{L}$ to $\mathcal{H}$.
4. $\mathcal{H}$ sends $\ell = \mathcal{L}(a_{[\eta]})$ to $\mathcal{A}'$, and $\mathcal{A}'$ forwards ℓ to $\mathcal{A}$.
5. $\mathcal{H}$ samples and sends $j \xleftarrow{\$} \mathcal{J}$ to $\mathcal{A}'$, and $\mathcal{A}'$ forwards j to $\mathcal{A}$.
6. $\mathcal{A}$ responds with some message $m_{[\eta]}$ and $\mathcal{A}'$ receives this message but does not forward it to $\mathcal{H}$.
7. $\mathcal{H}$ samples $(u_{[-\gamma]}, u_{[\eta]}) \xleftarrow{\$} C_j$, computes $\alpha_{[\eta]} = u_{[\eta]} - a_{[\eta]}$ and sends $\alpha_{[\eta]}$ to $\mathcal{A}'$.
8. $\mathcal{A}'$ computes $r_{[\eta]} = m_{[\eta]} - x_{[\eta]}$, samples $(w_{[-\gamma]}, w_{[\eta]}) \xleftarrow{\$} C_j^{(2)}$, and computes $\beta_{[\eta]} = w_{[\eta]} - \alpha_{[\eta]} * r_{[\eta]} + z_{[\eta]}$. $\mathcal{A}'$ forwards $(\alpha_{[\eta]}, \beta_{[\eta]})$ to $\mathcal{A}$.
9. $\mathcal{H}$ samples $b \xleftarrow{\$} \{0,1\}$ and $y \xleftarrow{\$} \mathbb{F}^\gamma$, and sends $\mathsf{chal} = b \cdot y + (u_{-\gamma})$ to $\mathcal{A}'$. $\mathcal{A}'$ forwards chal to $\mathcal{A}$.
10. $\mathcal{A}$ replies with bit $\tilde{b} \in \{0,1\}$ and $\mathcal{A}'$ forwards $\tilde{b}$ to $\mathcal{H}$.

We highlight the differences between the above reduction and the actual protocol. In the reduction, the index j is sampled by the honest challenger instead of Bob; however, this is identical because Bob is semi-honest and will sample j honestly. Next, $\mathcal{A}'$ generates the value

$\beta_{[\eta]}$ differently than in the real protocol. However, by the correctness of the protocol we have that

$$\begin{aligned} t_{[\eta]} &= u_{[\eta]} * r_{[\eta]} + v_{[\eta]} \\ &= (\alpha_{[\eta]} * r_{[\eta]}) + \beta_{[\eta]} - z_{[\eta]}. \end{aligned}$$

This implies that

$$\begin{aligned} \beta_{[\eta]} &= ((u_{[\eta]} * r_{[\eta]}) + v_{[\eta]}) - (\alpha_{[\eta]} * r_{[\eta]}) + z_{[\eta]} \\ &= w_{[\eta]} - (\alpha_{[\eta]} * r_{[\eta]}) + z_{[\eta]}, \end{aligned}$$

where $(w_{[-\gamma]}, w_{[\eta]}) \xleftarrow{\$} C_j^{(2)}$. Note that in the real protocol $(v_{[-\gamma]}, v_{[\eta]}) \xleftarrow{\$} C_j^{(2)}$ and $(u_{[-\gamma]} * r_{[-\gamma]}, u_{[\eta]} * r_{[\eta]}) \in C_j^{(2)}$. This implies that $\beta_{[\eta]}$ generated in the reduction is identically distributed to $\beta_{[\eta]}$ generated in the real protocol. This implies that $\mathcal{A}'$ correctly predicts b with probability greater than ε, leading to our contradiction. □

Let $\{C_j\}_{j\in\mathcal{J}}$ be a family of linear code distributions over $\mathbb{F}^{\gamma+\eta}$ satisfying Property 3.6.1 for appropriate values of δ and γ, where $\mathbb{F}$ is a finite field of characteristic 2.
Let $\mathsf{Enc}_{\mathrm{BMN}}\colon \{0,1\}^{m/2} \to \mathbb{F}^{\gamma}$ and $\mathsf{Dec}_{\mathrm{BMN}}\colon \mathbb{F}^{\gamma} \to \{0,1\}^{m/2}$ be the encoding and decoding algorithms for the BMN Embedding [54] for appropriate $\ell \in \mathbb{N}$.

Let $\mathcal{D} \subseteq \{0,1\}^{n/2}$ be a multiplication friendly code with message space $\mathbb{F}^{\eta}$. Let $(\mathsf{Enc}_{\mathcal{D}}, \mathsf{Dec}_{\mathcal{D}})$ (resp., $\mathsf{Enc}_{\mathcal{D}^{(2)}}, \mathsf{Dec}_{\mathcal{D}^{(2)}}$) be encoding and decoding procedures for $\mathcal{D}$ (resp., $\mathcal{D}^{(2)}$).

Hybrid: Client A receives (a,b) for $a,b \xleftarrow{\$} \{0,1\}^{n/2}$ and Client B receives (x,z) for $x \xleftarrow{\$} \{0,1\}^{n/2}$ such that $z_i = a_i \cdot x_i + b_i$ for every $i \in [n/2]$.

1. *Code Generation.* Client B samples $j \xleftarrow{\$} \mathcal{J}$.
2. ROLE *Extraction Protocol.*

   (a) Client B samples $\tilde{x} \xleftarrow{\$} \{0,1\}^{m/2}$ and sets $R = \mathsf{Enc}_{\mathrm{BMN}}(\tilde{x}) \in \mathbb{F}^{\gamma}$. Client B samples random codeword $(R_{-\gamma}, \ldots, R_{-1}, R_1, \ldots, R_{\eta}) \xleftarrow{\$} C_j$ such that $R = (R_{-\gamma}, \ldots, R_{-1})$ and sets $R' = (R_1, \ldots, R_{\eta})$. Client B computes $r' = \mathsf{Enc}_{\mathcal{D}}(R') \in \{0,1\}^{n/2}$, computes $m = r' + x \in \{0,1\}^{n/2}$, and sends (m, j) to Client A.

   (b) Client A samples $\tilde{a} \xleftarrow{\$} \{0,1\}^{m/2}$ and $\tilde{b} \xleftarrow{\$} \{0,1\}^{m/2}$ and sets $U = \mathsf{Enc}_{\mathrm{BMN}}(\tilde{a}) \in \mathbb{F}^{\gamma}$ and $V = \mathsf{Enc}_{\mathrm{BMN}}(\tilde{b}) \in \mathbb{F}^{\gamma}$. Client A samples random codewords $(U_{-\gamma}, \ldots, U_{-1}, U_1, \ldots, U_{\eta}) \xleftarrow{\$} C_j$ and $(V_{-\gamma}, \ldots, V_{-1}, V_1, \ldots, V_{\eta}) \xleftarrow{\$} (C_j * C_j)$ such that $U = (U_{-\gamma}, \ldots, U_{-1})$ and $V = (V_{-\gamma}, \ldots, V_{-1})$. Client A sets $U' = (U_1, \ldots, U_{\eta})$ and $V' = (V_1, \ldots, V_{\eta})$, computes $u' = \mathsf{Enc}_{\mathcal{D}}(U') \in \{0,1\}^{n/2}$ and $v' = \mathsf{Enc}_{\mathcal{D}^{(2)}}(V') \in \{0,1\}^{n/2}$, and computes

   $$\alpha = u' - a \in \{0,1\}^{n/2} \qquad \beta = (a' * m) + b + v' \in \{0,1\}^{n/2}.$$

   Client A sends (α, β) to Client B.

   (c) Client B computes $t' = (\alpha * r') + \beta - z \in \{0,1\}^{n/2}$. Client B computes $T' = \mathsf{Dec}_{\mathcal{D}^{(2)}}(t') \in \mathbb{F}^{\eta}$, then computes $T \in \mathbb{F}^{\gamma}$ via the erasure recovery algorithm of $C_j * C_j$ such that $(T, T') \in C_j * C_j$. Client B then computes $\tilde{z} = \mathsf{Dec}_{\mathrm{BMN}}(T) \in \{0,1\}^{m/2}$.

   (d) Client A outputs $(\tilde{a}, \tilde{b})$ and Client B outputs $(\tilde{x}, \tilde{z})$.

Figure 3.7. Our correlation extractor for $\mathsf{ROLE}^{n/2}$.

4. SUCCINCT (NON-)INTERACTIVE ARGUMENTS OF KNOWLEDGE: USING CODING THEORY TECHNIQUES TO PROVE SOUNDNESS

A portion of this chapter appears in the 2020 Theory of Cryptography Conference, published by The International Association for Cryptologic Research and Springer-Verlag [51], available at https://doi.org/10.1007/978-3-030-64378-2_7. A portion of this chapter also appears in The International Association for Cryptologic Research Cryptology ePrint Archive [53], available https://ia.cr/2021/358. The article [53] is the full version of the article which appears in the proceedings of Advances in Cryptology–CRYPTO 2021, published by The International Association for Cryptologic Research and Springer-Verlag [52], available at https://doi.org/10.1007/978-3-030-84259-8_5.

In this chapter, we examine how coding theory has been composed with cryptography *implicitly* by examining cryptographic primitives with security guarantees that are provable by taking a coding theoretic approach to the security analysis. Such primitives are widespread, but we focus on *probabilistically checkable proofs*, or PCPs. The seminal PCP theorem [15, 16, 22, 114, 216] can be viewed (both implicitly and explicitly) as following from coding theoretic objects known as *locally decodable/testable codes*.[1] A PCP proof string attests the truth of a mathematical statement such that any verifier making a small number of queries to the string can be convinced of the truth with probability (arbitrarily close to) 1, and any false statement only convinces a verifier with probability less than half. The PCP theorem states that $\mathsf{NP} = \mathsf{PCP}[O(\log(n)), O(1)]$: for every language $L \in \mathsf{NP}$ and any $x \in \{0,1\}^n$, we have that $x \in L$ if and only if there exists a $\text{poly}(n)$-length proof string such that any verifier using $O(\log(n))$ bits of randomness and querying $O(1)$ bits of the proof accepts with probability 1, and $x \notin L$ if and only if after the above process, a verifier accepts with probability less than half. Intuitively, PCP proof of a false statement is guaranteed to contain many errors, a property that is analogous to error-correcting codes when encoding two distinct messages [104]. In some sense, a PCP proof string can be viewed as a *locally testable code*: an error-correcting code with codewords that can be queried at a few locations to determine if it is valid codeword.

[1] ↑In fact, we directly examine locally decodable codes in Part II.

The history of the PCP theorem has roots in the area of *interactive proofs* [132]. Interactive proofs are interactive protocols between a computationally unbounded prover P and a computationally weak (i.e., PPT) verifier V, where the prover P tries to convinces the verifier V of the truth of a mathematical statement. In the same vein as PCPs, the verifier V accepts any true statement with probability (arbitrarily close to) 1, and accepts any false statement with probability less than half.[2] The complexity class IP is the class of languages recognizable by interactive proofs with polynomially many rounds of interaction, and it is well-known that $\mathsf{IP} = \mathsf{PSPACE}$ [192, 236]; even here, methods used to prove this result (e.g., arithmetization) have strong connections to coding theory.

4.1 Succinct (Non-)Interactive Arguments for NP

In this dissertation, we examine *succinct (non-)interactive arguments* for NP [206]. Informally, an *interactive argument* is an interactive protocol between a computationally bounded (i.e., polynomial time) prover P and a weak (PPT) verifier V, where P tries to convince V of the validity of some NP statement. Such protocols are restrictions of interactive proofs [132], and have been of great interest in recent veins of research for efficiency concerns: naturally, interactive proofs are useful in many real-world contexts, but it is desirable to construct proofs using minimal resources (e.g., time and space). Recent results have focused on constructing arguments for the set of all NP languages that can be verified by a *random access machine (RAM)* M. A RAM M has the same semantics as a Turing machine, with the exception that during any time step of the computation, the machine head can jump to any location of the work tape with unit cost.[3] The machine M runs in time T and space S if on any input, (1) M runs in at most T steps then terminates; and (2) M uses at most S cells of the work tape during the entire execution of the machine. In this context, the argument is *succinct* if the communication complexity is sub-linear in T. Fixing T and S for the remainder of the chapter, let $\mathcal{L}_{\text{RAM}}$ denote the set of all NP languages verifiable by a time-T and space-S RAM, and let $\mathcal{R}_{\text{RAM}}$ denote the (NP) relation for $\mathcal{L}_{\text{RAM}}$.

[2] ↑More generally, we can define the probability of accepting a false statement to be some function $\varepsilon(\cdot)$.
[3] ↑This is an informal presentation of RAMs. For more formal presentations on the semantics, see [64, 157].

While there exist many ways to construct arguments for $\mathcal{R}_{\text{RAM}}$, the following general approach has found many success stories [31, 34, 42, 51, 52, 67, 72, 76, 124, 137, 255].

1. Given a RAM M for some input-witness pair $(x, w) \in \mathcal{R}_{\text{RAM}}$, transform M into an equivalent arithmetic circuit $C\colon \{0,1\}^* \to \mathbb{F}$, for some finite field $\mathbb{F}$, such that $M(x, w) = 1$ if and only if $C(x, w) = 1$ [33, 64]. In particular, C contains addition and multiplication gates over $\mathbb{F}$, and each gate has fan-in 2.

2. Let $s = O(\log(|C|))$. Then, given C and input (x, w), define the *wire transcript* $W\colon \{0,1\}^s \to \mathbb{F}$. That is, on input gate number $b \in \{0,1\}^s$, the function $W(b)$ is the value of the output wire of gate b of circuit C on input (x, w).

3. Given W, define the *multi-linear extension* $\widetilde{W}\colon \mathbb{F}^s \to \mathbb{F}$ of W;[4] that is, $\widetilde{W}$ is a polynomial of individual degree at most 1 such that $\widetilde{W}(x) = W(x)$ for all $x \in \{0,1\}^s$.

4. Define an auxiliary polynomial $F\colon \mathbb{F}^{3s} \to \mathbb{F}$ such that $F \equiv 0$ if and only if W is a correct wire transcript of $C(x, w)$; i.e., for every gate $g \in \{0,1\}^s$, the value of $W(g)$ is the output wire value of gate g in $C(x, w)$.

5. Commit to $\widetilde{W}$ using a *polynomial commitment scheme* [165]. Briefly, a polynomial commitment scheme is a commitment scheme that allows a sender to commit to a polynomial P with the additional feature that the receiver can query the polynomial at a point x, and the sender provides $P(x)$ and an additional proof π which certifies that $P(x)$ is consistent with the committed polynomial P. In particular, this proof π allows the sender to open at any point x *without revealing the entire polynomial* P.

6. Perform a *sum-check* protocol [192] over the polynomial F, which induces some claims about the polynomial $\widetilde{W}$, which are answered using the polynomial commitment scheme.

Ignoring the polynomial commitment scheme, the above transformation is used to construct interactive *proofs* for NP as well, where the polynomial $\widetilde{W}$ acts as a PCP proof string that a verifier can query at random points. In fact, $\widetilde{W}$ is a low-degree *Reed-Muller codeword*,

[4] ↑Many applications define $\widetilde{W}$ as some *low-degree extension*. For our purposes, a multi-linear extension suffices. See Section 4.5.2 for the formal definition.

which is crucial to the soundness (i.e., security) of many proof systems using the above transformation (e.g., see [64]). The addition of a polynomial commitment scheme yields a succinct argument with soundness that additionally relies on the cryptographic assumptions of the polynomial commitment.

Succinct arguments using the above general transformation have been pushed to nearly optimal parameters. Many schemes exist in which provers run in time $\widetilde{O}(T)$ [67, 72, 76, 255], verifiers run in time polylog(T) [51, 52, 76], and proofs have length $O(\log(T))$ or even $O(1)$ [67, 72, 76, 255].[5] However, very few results—even outside of this general transformation—have focused on the *space complexity* of the prover, with some notable exceptions [40, 41, 64, 68, 157, 251]. In particular, most results require a prover to store *the entire circuit* C: this circuit has size size $\Omega(T)$, which is (generally) *much larger* than the space S used by the underlying RAM M in applications; e.g., for typical applications we consider $S = \text{polylog}(T)$.

4.1.1 Towards Space Efficiency

As mentioned previously, with the few exceptions of [40, 41, 64, 68, 157, 251], space-efficiency has been largely overlooked; however, space, just like time, is an important resource to consider in the construction of (non-interactive, zero-knowledge) argument schemes. In fact, one may argue that it is more important: for example on a real computer, it is easy to allow a program to run for more time (simply let the program run longer). However, one cannot obtain more storage space as easily, and the hierarchical nature of computer storage (e.g., registers, cache, RAM, disk) have real implications on implementations of these schemes. Thus space-efficiency is well-motivated.

Our goal is to construct *complexity preserving* arguments [41]: arguments where the prover (roughly) runs in time T and space S. Given the above methodology, we identify two key bottlenecks to achieving a space-efficient prover. The first bottleneck is the transformation of M to an equivalent circuit C, as this circuit is directly manifested by the prover and is stored in its entirety. The second bottleneck is using polynomial $\widetilde{W}$ in the underlying polynomial commitment scheme. In *all* prior polynomial commitment schemes, the committer

[5] In all of the above complexities, poly(λ) factors are omitted, where λ is the security parameter.

is assumed to have the entire description of the polynomial in memory, and therefore the space complexity of this committer is proportional to the description size of the polynomial.

The work of Blumberg et al. [64] overcomes the first bottleneck by giving a complexity preserving *multi-prover interactive proof*, or an MIP, where the key technical contribution is providing a method for computing the RAM to circuit transformation in a *streaming* way, modifying the transformation of Ben-Sasson et al. [33]. By streaming, we mean that a prover can compute the circuit transcript gate-by-gate without writing down the entire circuit. This MIP is denoted as the Clover MIP, and an MIP is an interactive proof with multiple colluding provers with the following property: before the start of the protocol, all provers can interact with each other. At the start of the protocol, the verifier can interact with individual provers, and the colluding provers cannot communicate any further.

Let M be a RAM and C be its equivalent circuit. The Clover MIP gives a transformation that can evaluate C on inputs x and w in a gate-by-gate streaming manner by running the underlying RAM M. In more detail, this transformation can construct a wire transcript W of C in lexicographic order as a stream; i.e., first $W(0)$ is generated, followed by $W(1)$, all the way to $W(2^s - 1)$, where we interpret an integer in $\{0, \ldots, 2^s - 1\}$ as a binary number in the natural way. Ignoring the space required to store this wire transcript W for the time being, the generation of this transcript takes time $\widetilde{O}(T)$ and, more importantly, space $S \cdot \mathrm{polylog}(T)$. In relation to the transformation presented in Section 4.1, the Clover MIP transformation produces the transcript W in time $\widetilde{O}(T)$ and space $S \cdot \mathrm{polylog}(T)$. A restrictive property of this streaming construction of the wire transcript W is that if $W(b)$ has been computed for $b \in \{0,1\}^s$, then we can either continue to compute $W(b+1)$ using some small amount of time and space, *or* restart the stream. Notably, we cannot compute $W(b')$ for arbitrary $b' \in \{0,1\}^s$, unless we just computed $W(b'-1)$, and hence our streaming access is, in a sense, one-way and read-once (without re-computation). However, even given the above RAM to circuit transformation, it is not clear how to circumvent the second bottleneck.

4.2 Our Results

In a series of works, we overcome the second bottleneck by introducing the notion of a *streamable polynomial commitment scheme* [51, 52]. Such a polynomial commitment scheme assumes that a committer has multi-pass streaming access to the description of the polynomial (rather than hold the description in memory), and that the polynomial is efficiently computable in small space given this (streaming) description. For example, given a degree d univariate polynomial f over some finite field $\mathbb{F}$, a description of the polynomial can be $d+1$ evaluations $(f(0), \ldots, f(d))$ of the polynomial, and the polynomial can be evaluated at any point via Lagrange Interpolation. We stress that this multi-pass streaming access is the same type of streaming access given by the Clover MIP: if $(f(0), \ldots, f(d))$ is the description of the polynomial, then at any time step t we can either advance the stream by 1 (i.e., receive $f(t+1)$), or restart the stream (i.e., receive $f(0)$).

We give two different streamable polynomial commitment scheme for multi-linear polynomials, under different assumptions, and construct complexity preserving arguments for NP by composing their polynomial commitment scheme with the Clover MIP [51, 52]. Both works assume streaming access to the wire transcript W (given by the Clover MIP) and use this access to commit to the multi-linear extension $\widetilde{W}$ (since W uniquely define the extension $\widetilde{W}$; see Fact 4.3.1). We discuss both polynomial commitment schemes.

4.2.1 Streamable Polynomial Commitments from the Discrete-log Assumption in the Random Oracle Model

Our first result is constructing a streamable polynomial commitment scheme, in the random oracle model, assuming hardness of discrete logarithm. This result was the first to introduce and utilize the new model streamable polynomial commitment scheme, and is the *first* space-efficient polynomial commitment scheme. Using it we construct the *first* complexity preserving argument (and ZK-SNARK) under standard cryptographic assumptions (see [51, 52] for complete details).

Theorem 4.2.1 (Informal, see Theorem 4.6.4 [51])**.** *Let λ be the security parameter. Assume the existence of a prime-order group for which the discrete-log assumption holds. Then there*

exists a polynomial commitment scheme for multi-linear polynomials $Q\colon \mathbb{F}^n \to \mathbb{F}$ *over a prime-order field* $\mathbb{F}$ *(of size* $|\mathbb{F}| \leqslant 2^{\mathrm{poly}(n)}$*), in the random oracle model, with the following efficiency properties:*

1. *Commitments and evaluation proofs can be computed with* $2^n \cdot \mathrm{poly}(n)$ *queries to the random oracle, and in time* $2^n \cdot \mathrm{poly}(\lambda, n)$ *and space* $n \cdot \mathrm{poly}(\lambda)$ *given multi-pass streaming access to the evaluations of* Q *on the Boolean hypercube;*

2. *Verification time and query complexity are* $2^n \cdot \mathrm{poly}(\lambda, n)$ *and verification space is* $n \cdot \mathrm{poly}(\lambda)$*; and*

3. *The communication complexity is* $n \cdot \mathrm{poly}(\lambda)$.

Looking ahead, the polynomial commitment scheme of Theorem 4.2.1 is a (non-trivial) modification of the well-known Bulletproofs [67, 72] polynomial commitment scheme, which utilizes Pedersen commitments [218]. We discuss the main ideas and differences of the above polynomial commitment scheme in Section 4.3.2. We also mention that in Theorem 4.2.1, the random oracle model is not necessary for security, but rather for our construction we need the random oracle to maintain space-efficiency. See Section 4.3.2 for details.

4.2.2 Streamable Polynomial Commitments from the Hidden Order Assumption

Our second result is constructing a streamable polynomial commitment scheme under the *hidden order assumption*. Informally, the hidden order assumption for a group $\mathbb{G}$ states that for a random $g \xleftarrow{\$} \mathbb{G}$, it is computationally infeasible to find (any multiple of) the order of g. This streamable polynomial commitment scheme builds upon the ideas of Theorem 4.2.1, but is able to forgo the random oracle *and* have super-efficient verification (in addition to space-efficiency).

Theorem 4.2.2 (Informal, see Theorem 4.7.7 [52])**.** *Let* λ *be the security parameter. Assume the existence of a group for which the hidden order assumption holds. Then there exists a polynomial commitment scheme for multi-linear polynomials* $Q\colon \mathbb{F}^n \to \mathbb{F}$ *over a prime-order field* $\mathbb{F}$ *(of size* $|\mathbb{F}| \leqslant 2^{\mathrm{poly}(n)}$*) with the following efficiency properties:*

1. *Commitments and evaluation proofs can be computed in time* $2^n \cdot \text{poly}(n, \lambda)$ *and space* $n \cdot \text{poly}(\lambda)$*, given multi-pass streaming access to the evaluations of* Q *on the Boolean hypercube; and*

2. *The communication complexity and verification time are both* $n \cdot \text{poly}(\lambda)$*.*

Looking ahead, the polynomial commitment scheme of Theorem 4.2.2 is based off of the so-called DARK polynomial commitment scheme due to Bünz, Fisch, and Szepieniec [76]. Our polynomial commitment scheme is a highly non-trivial variant of the DARK polynomial commitment scheme and, in fact, circumvents a known gap in the security proof of the DARK scheme [52].[6] We discuss the main ideas and differences of the above polynomial commitment scheme in Section 4.3.3.

4.2.3 From Streamable Polynomial Commitments to Complexity Preserving Zero-Knowledge Arguments

We obtain complexity preserving arguments by compiling the streaming Clover MIP with both of our streamable polynomial commitment schemes. In the random oracle model and assuming the hardness of discrete-log, compiling the Clover MIP with Theorem 4.2.1 yields the following result.

Theorem 4.2.3 ([51])**.** *Let* λ *be the security parameter. Assume the existence of a group sampler of prime order for which the discrete-log assumption holds. Then there exists a public-coin zero-knowledge argument system in the random oracle model for any* NP *relation verifiable by a time-*T *space-*S *random access machine with the following complexity.*

1. *The protocol has perfect completeness and negligible soundness error;*

2. *The number of rounds is* $O(\log(T))$ *and the communication complexity complexity is* $\text{poly}(\lambda, \log(T))$*;*

3. *The prover runs in time* $T \cdot \text{poly}(\lambda, \log(T))$ *and space* $S \cdot \text{poly}(\lambda, \log(T))$*; and*

[6]↑We emphasize that the same gap was independently discovered by the authors of [76], and that we were informed of this gap by the authors as well [74].

4. *The verifier runs in time* $T \cdot \text{poly}(\lambda, \log(T))$ *and space* $\text{poly}(\lambda, \log(T))$.

Second, under the hidden order assumption, compiling the Clover MIP with Theorem 4.2.2 yields the following result.

Theorem 4.2.4 ([52])**.** *Assume the existence of a group sampler for which the hidden order assumption holds. Then there exists a public-coin zero-knowledge argument-system for any* NP *relation verifiable by a time-*T *space-*S *random access machine with the following complexity.*

1. *The protocol has perfect completeness and negligible soundness error;*
2. *The number of rounds is* $O(\log(T))$*;*
3. *The communication complexity is* $\text{poly}(\lambda, \log(T))$*;*
4. *The prover runs in time* $T \cdot \text{poly}(\lambda, \log(T))$ *and space* $S \cdot \text{poly}(\lambda, \log(T))$*; and*
5. *The verifier runs in time* $|x| \cdot \text{poly}(\lambda, \log(T))$*, for a given input* x*.*

In both Theorems 4.2.3 and 4.2.4, zero-knowledge is obtained via the standard "commit-and-prove" techniques due to Ben-Or et al. [29]. Further, both results are made non-interactive in the random oracle model via applying the Fiat-Shamir transformation [116].

4.3 Technical Overview

In this section, we discuss the key technical ideas behind our two streamable polynomial commitment schemes. We begin by discussing the steaming model for polynomial commitment schemes, then discuss our first polynomial commitment scheme based on the hardness of discrete-log, and finally discuss our second polynomial commitment scheme based on the hidden order assumption.

4.3.1 The Streaming Model for Polynomial Commitments

In prior works on polynomial commitments, a committer is assumed to have explicit access to the description of the polynomial it is committing to [30, 36, 67, 76, 166, 184, 232, 255, 266]. Here, a *description* of a polynomial is either a sequence of coefficients or

evaluations. For example, for a univariate polynomial $f \in \mathbb{F}[X]$ of degree at most d, the description of f could either be its $d+1$ coefficients or $d+1$ evaluations over $\mathbb{F}$. In the case of n-variate multilinear polynomials $Q \in \mathbb{F}[X_1, \ldots, X_n]$, we explicitly consider the sequence of evaluations $(Q(b))_{b \in \{0,1\}^n}$, where the sequence is given in lexicographic order. Prior works also work with polynomials in either the coefficient basis (in the case of univariate polynomials) or the evaluation basis (in the case of multilinear polynomials).[7] In either case, the space complexity of a committer is at least the amount of space required to store the description of the polynomial. In a void, this seems like a non-issue; however, when using polynomial commitments to compile a time-T space-S computation into succinct arguments, the description size of the polynomial is proportional to the time T of the computation. This leads to massive committer space overheads, and is undesirable for our goal of complexity preserving arguments.

Thus, our goal is to construct polynomial commitments where the committer uses space *sub-linear* in the description size of the polynomial in consideration. Since this is impossible when the committer explicitly stores the polynomial,[8] we instead consider an alternate access model for the description of the polynomial. We consider the *multi-pass streaming model* [51]: in this model, a committer is given multi-pass streaming access to the description of the polynomial in consideration. Focusing on the multilinear polynomial Q and its description $(Q(b))_{b \in \{0,1\}^n}$, the multi-pass streaming model has the following properties. The committer has streaming access to the description $(Q(b))_{b \in \{0,1\}^n}$ in lexicographic order. Initially, the stream begins at $Q(0)$, and the committer can access the description of Q in lexicographic order. Then, at any time step $b \in \{0,1\}^n$, the committer receives $Q(b)$ and can either

1. advance the stream to $Q(b+1)$; or
2. restart the stream to $Q(0)$.

In particular, the committer *cannot* move the stream backwards, and does not have random access to the description of Q (i.e., querying $Q(b)$ requires time proportional to b, rather than

[7] ↑We restrict our attention to multilinear polynomials as they suffice for our purposes.
[8] ↑Of course, in the case of a polynomial which is sufficiently sparse (e.g., $1 + X^d$), an arbitrary or random polynomial need not be sparse (and in fact is highly unlikely to be sparse).

$O(1)$ time). Further, the committer can restart and traverse the stream as many times as they wish (i.e., it is multi-pass, not read-once).

While this streaming model is indeed restrictive, it is sufficient for our purposes. Even in this restrictive model, we can evaluate the multilinear polynomial Q at any point, leveraging the following fact.

Fact 4.3.1 ([51, 213])**.** *An multilinear polynomial $f : \mathbb{F}^n \to \mathbb{F}$ (over a finite field $\mathbb{F}$) is uniquely defined by its evaluations over the Boolean hypercube. Moreover, for every $\zeta \in \mathbb{F}^n$ it holds that*

$$f(\zeta) = \sum_{b \in \{0,1\}^n} f(b) \cdot \prod_{i=1}^{n} \chi(b_i, \zeta_i), \tag{4.1}$$

where $\chi(b_i, \zeta_i) = b_i \cdot \zeta_i + (1 - b_i) \cdot (1 - \zeta_i)$.

Thus the following streaming algorithm efficiently computes $Q(\zeta)$ for any $\zeta \in \mathbb{F}^n$.

1. Initialize $y = 0 \in \mathbb{F}$.
2. For $b \in \{0,1\}^n$ in lexicographic order
   (a) Compute $\chi = \prod_i \chi(b_i, \zeta_i)$. Note that computing χ takes $O(n)$ field elements of storage and $O(n)$ field multiplications.
   (b) Compute $y = y + Q(b) \cdot \chi$.
   (c) Advance the stream by 1.
3. Output y.

Ignoring $\text{poly}(\lambda)$ factors, the above algorithm computes $Q(\zeta)$ in time $O(2^n \cdot n)$ and, more importantly, space $O(n)$, which is *exponentially smaller* than the description size of Q, which is 2^n. As we will see, this streaming access allows us to construct polynomial commitments space proportional to n.

Remark 4.3.2. *Given random access to the description of Q, or even allowing the committer to traverse the description of Q in either direction (e.g., similar to a Turing machine tape*

head) is a stronger model than what we propose here. However, when eventually compiling our polynomial commitments with the Clover MIP, the above model captures exactly how the Clover MIP generates the circuit wire transcript.

4.3.2 Overview of Theorem 4.2.1

Our first polynomial commitment scheme is based on the well-known Bulletproofs polynomial commitment scheme (and inner-product argument) [67, 72], which assumes the hardness of discrete logarithm for a group $\mathbb{G}$. We first give an overview of this scheme. Fix a multi-linear polynomial $Q\colon \mathbb{F}^n \to \mathbb{F}$ with evaluations $(Q(b))_{b\in\{0,1\}^n}$. For convenience, we (equivalently) write $Q \in \mathbb{F}^N$ as the evaluations $(Q(b))_{b\in\{0,1\}^n}$.

Commitment Phase

Bulletproofs commits to the polynomial Q using a Pedersen commitment: for $N = 2^n$ group generators $g = (g_b)_{b\in\{0,1\}^n} \in \mathbb{G}^N$, the commitment is computed as

$$C = \prod_{b\in\{0,1\}^n} g_b^{Q(b)}.$$

The committer computes and sends this commitment C to the receiver. This commitment C is a binding commitment to Q with respect to the generators g. We note that our commitment phase is identical to this, and that from the above equation it is easy to see how to construct a space-efficient streaming algorithm to compute the commitment C; see Section 4.6.4 for details.

Bulletproofs Evaluation Phase

Given an evaluation point $\zeta \in \mathbb{F}^n$ specified by the receiver, the committer computes and sends $y = Q(\zeta)$ and defines an auxiliary commitment $C_y = C \cdot h^y$, where $h \xleftarrow{\$} \mathbb{G}$ is chosen

by the receiver. The committer and receiver then engage in an interactive argument (of knowledge) for the following NP relation:

$$\mathcal{R}_{\mathsf{BP}}(n) = \left\{ (C_y, \zeta, g, h, z, y; Q) : \begin{array}{c} z, Q \in \mathbb{F}^N, \zeta \in \mathbb{F}^n, g \in \mathbb{G}^N, h \in \mathbb{G} \\ \wedge \ y = \langle Q, z \rangle \in \mathbb{F} \\ \wedge \ C_y = h^y \cdot \prod_b g_b^{Q(b)} \in \mathbb{G} \end{array} \right\}, \tag{4.2}$$

where $N = 2^n$, $g \in \mathbb{G}^N, h \in \mathbb{G}$ are given as above and $z := (\chi(b, \zeta))_{b \in \{0,1\}^n}$ and $\chi(b, \zeta)$ is defined as in Fact 4.3.1. Note that the inner-product $\langle Q, z) \rangle$ is identical to Equation (4.1) of Fact 4.3.1. This step allows the committer to prove knowledge of a decommitment Q of the commitment C_y such that $y = \langle Q, z \rangle$.

To prove the above inner-product argument, Bulletproofs employs a natural "split-and-fold" recursive argument. The core step for the above argument is a two-move randomized reduction step which allows the prover to decompose the size N statement (i.e., the inner-product of two N-sized vectors) (C_y, z, y) into two size $N/2$ statements, then using verifier randomness "fold" these into a single size $N/2$ statement. For example, consider a univariate polynomial $f(x) = 1 + x + x^2 + 2x^3$ with evaluation point $\zeta \in \mathbb{F}$, and let $y = f(\zeta)$. Ignoring any commitment, we can prove knowledge of f via the two-move reduction above.

1. The committer computes $y = f(\zeta)$. Additionally, the committer defines $f_L(x) = 1 + x$ and $f_R(x) = 1 + 2x$. Note that $f_L(x) + x^2 \cdot f_R(x) = f(x)$. The committer then computes $y_L = f_L(\zeta)$ and $y_R = f_R(\zeta)$ and sends (y, y_L, y_R) to the receiver.

2. The receiver samples $\alpha \xleftarrow{\$} \mathbb{F}$ and sends α to the committer.

3. The committer and receiver then compute $y' = \alpha \cdot y_L + y_R$ and the committer additionally computes $f'(x) = \alpha \cdot f_L(x) + f_R(x)$.

The committer and receiver recursively perform the above process until only a constant polynomial remains, and the committer sends this constant to the receiver, and a final check is performed (i.e., $y \stackrel{?}{=} f$).

Moving back to the relation $\mathcal{R}_{\mathsf{BP}}(n)$, fix $(C_y, g, h, z, y, Q) \in \mathcal{R}_{\mathsf{BP}}(n)$ and suppose both the committer and receiver are given (C_y, g, h, z, y) and the committer additionally receives

the vector Q. Then the two-move reduction of Bulletproofs modifies the above two-move reduction as follows:

1. The committer computes the cross-product $y_L = \langle Q_L, z_R \rangle$ between the left half of the vector Q and the right half of the vector z; similarly $y_R = \langle Q_R, z_L \rangle$ is computed. Furthermore, the committer computes a new commitment to the left vector Q_L and right vector Q_R as

$$C_L = h^{y_L} \cdot \prod_{b' \in \{0,1\}^{n-1}} g_{R,b'}^{Q_L(b')} \qquad\qquad C_R = h^{y_R} \cdot \prod_{b' \in \{0,1\}^{n-1}} g_{L,b'}^{Q_R(b')}.$$

   Here, g_L and g_R are the left and right halves of the vector g, respectively, and both lie in $\mathbb{G}^{N/2}$. The committer sends (y_L, y_R, C_L, C_R) to the receiver.

2. The receiver samples $\alpha \xleftarrow{\$} \mathbb{F}$ and sends α to the committer.

3. The committer and receiver both compute the folding

$$z' = \alpha^{-1} \cdot z_L + \alpha \cdot z_R \in \mathbb{F}^{N/2}$$
$$g' = (g_L)^{\alpha^{-1}} * (g_R)^{\alpha} \in \mathbb{G}^{N/2}$$
$$y' = \alpha^2 \cdot y_L + y + \alpha^{-2} \cdot y_R \in \mathbb{F}$$
$$C'_{y'} = C_L^{\alpha^2} \cdot C_y \cdot C_R^{\alpha^{-2}}.$$

   The committer additionally computes a new folded witness Q' as $Q' = \alpha \cdot Q_L + \alpha^{-1} \cdot Q_R$.

Leveraging the homomorphic properties of Pedersen commitments, one can show that if Q is a witness for the tuple (C_y, g, h, z, y), then Q' is a witness for the tuple $(C'_{y'}, g', h, z', y')$. This forms a recursion tree, and after $n = \log(N)$ steps, all the vectors consist of a single element, and the committer simply sends its (single element) witness to the receiver.

Obstacles to Space-Efficiency

As a reminder, our goal is to leverage the streaming model to obtain a space-efficient implementation, where space-efficiency here means space usage proportional to n (ignoring

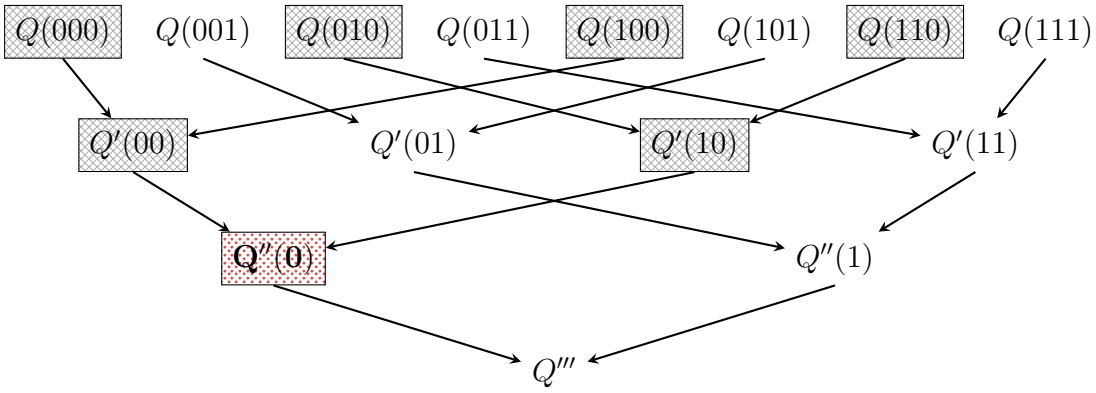

Figure 4.1. Example of the recursion tree induced by the Bulletproofs two-move reduction for $n = 3$, $N = 2^3$, and the polynomial $Q \in \mathbb{F}^N$. A child node is obtained by taking a linear combination of the parent nodes. In particular, left edges indicate multiplication of the parent by α and right edges indicate multiplication of the parent by α^{-1}, where α is the receiver challenge sent during the current round of recursion. For example, the value $Q''(0)$ depends on all nodes of the tree with a cross-pattern background.

poly(λ) factors). Unfortunately, directly applying our streaming model to the above Bulletproofs two-move reduction does not yield a space-efficient implementation. To see this, we examine the recursion tree induced by the above folding and first consider the folding of the polynomial Q. At any step of the recursion, the polynomial $Q \in \mathbb{F}^N$ is folded into a polynomial $Q' \in \mathbb{F}^{N/2}$; this folding is performed via a "left-right" folding. Formally, the polynomial Q' is defined as

$$\forall b' \in \{0,1\}^{n-1} \ : \ Q'(b') = \alpha \cdot Q(0 \circ b') + \alpha^{-1} \cdot Q(1 \circ b'),$$

where the folding is performed with respect to the most significant bit. This induces the recursion tree presented in Figure 4.1.

In our streaming model, we are only given multi-pass streaming access to the original polynomial Q, and not the subsequent recursive polynomials. Thus in order to achieve space-efficiency, we must compute the subsequent recursive streams (in their entirety) given only streaming access to the original stream Q. However, with the Bulletproofs two-move reduction, this is not possible in small-space. Let $Q^{(\ell)}$ denote the recursive stream at level

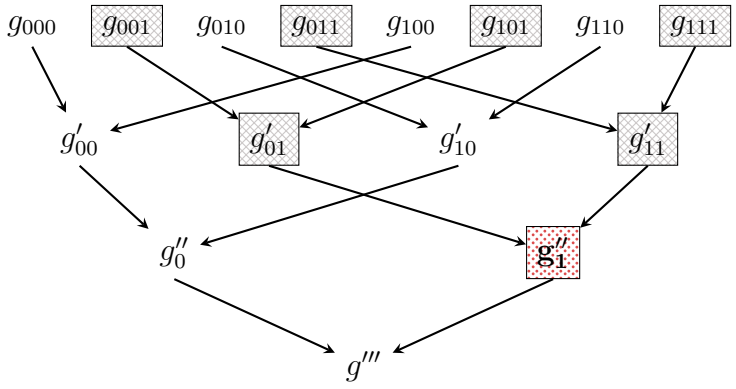

Figure 4.2. Example of the recursion tree induced by the Bulletproofs two-move reduction for $n = 3$, $N = 2^3$, and the generators $(g_1, \ldots, g_N)$. A child node is obtained by taking a linear combination of the parent nodes. In particular, left edges indicate exponentiation of the parent by α^{-1} and right edges indicate exponentiation of the parent by α, where α is the receiver challenge sent during the current round of recursion. For example, the value g''_1 depends on all nodes of the tree with a cross-pattern background.

$\ell \in \{0, 1, \ldots, n\}$ of the recursion (note that $Q^{(0)} = Q$). Then for any level $\ell \in \{0, 1, \ldots, n\}$ in the Bulletproofs recursion tree and any $b \in \{0,1\}^{n-\ell}$, the value $Q^{(\ell)}(b)$ depends on 2^ℓ values of the stream $Q^{(0)}$, and is specified by the following equation (ignoring the randomized challenges):

$$Q^{(\ell)}(b) = \sum_{c \in \{0,1\}^\ell} Q^{(0)}(c \circ b).$$

In particular, the value $Q^{(\ell)}(b)$ depends on all values of $Q^{(0)}$ with matching lower-order bits (i.e., $c \circ b \in \{0,1\}^n$). Thus the gap between any values of $Q^{(0)}$ needed to compute $Q^{(\ell)}(b)$ is exactly $2^{n-\ell}$; that is, the values of $Q^{(0)}$ needed to compute $Q^{(\ell)}(b)$ are not contiguous.

We remark that the space-efficiency issue arises in the computation of C_L and C_R, but not in computation of y_L, y_R.[9] The issue with the commitments is that each value $Q^{(\ell)}(b)$ is bound to a particular generator g_b (for the current level of recursion). For example, computing $Q''(0)$ in Figure 4.1 depends on generator g''_1 in Figure 4.2. Let $g_b^{(\ell)}$ denote the b^{th} folded generator

[9]↑There is a small-space algorithm to compute y_L, y_R at any level of recursion that leverages the fact that polynomial evaluations are a linear function.

at recursion level ℓ, where $\ell \in \{0, 1, \ldots, n\}$ and $b \in \{0,1\}^{n-\ell}$. Then for any recursion level $\ell \in \{0, 1, \ldots, n\}$ and any $b' \in \{0,1\}^{n-\ell-1}$, in the two-move reduction the generator $g^{(\ell)}_{1\circ b'}$ is raised to the power $Q^{(\ell)}(0 \circ b')$, and the generator $g^{(\ell)}_{0\circ b'}$ is raised to the power $Q^{(\ell)}(1 \circ b')$.

Let $C^{(\ell)}_L$ and $C^{(\ell)}_R$ denote the commitments computed in round ℓ of the recursion in the two-move reduction. Ignoring the space required to store the folded generators $g^{(\ell)} = (g^{(\ell)}_b)_{b\in\{0,1\}^{n-\ell}}$, we can compute these values in small-space given streaming access to Q as follows:

1. Initialize values $C^{(\ell)}_L = C^{(\ell)}_R = 1_{\mathbb{G}}$.
2. For $b = (b_1, \ldots, b_n) \in \{0,1\}^n$:
   (a) Let $b' = (b'_1, \ldots, b'_{n-\ell-1}) = (b_{\ell+2}, \ldots, b_n)$ and $c = b_{\ell+1}$
   (b) If $c = 0$ then compute $C^{(\ell)}_L = C^{(\ell)}_L \cdot (g^{(\ell)}_{1\circ b'})^{Q(b)}$. Else compute $C^{(\ell)}_R = C^{(\ell)}_R \cdot (g^{(\ell)}_{0\circ b'})^{Q(b)}$.

Notice that the above algorithm uses $O(1)$ group elements to compute the values $C^{(\ell)}_L, C^{(\ell)}_R$, where again we ignore the space required to store the generators $(g^{(\ell)}_b)_{b\in\{0,1\}^{n-\ell}}$.[10] Now during every round of recursion, the algorithm performs N group exponents, giving a total of $N \cdot \log(N)$ group exponents. This seems like we have our desired algorithm.

However, we now address the elephant in the room: the storage of generators $(g^{(\ell)}_b)_{b\in\{0,1\}^{n-\ell}}$. Clearly, storing these generators would violate our space-efficiency requirements, so we must rethink our access to these generators. Naturally, one might think to access these generators in a streaming manner, similar to our access to the polynomial Q. However, even given *random access* to the generators, a space-efficient implementation of the Bulletproofs two-move reduction would yield a *quadratic-time* committer via our methods.[11] This can be seen by examining the recursion tree in Figure 4.2: similar to the value $Q^{(\ell)}(b)$, the generator $g^{(\ell)}(b)$ depends on 2^ℓ generators at the top level (i.e., $(g_1, \ldots, g_N)$). Examining the above presented space-efficient algorithm, any generator $g^{(\ell)}_b$ needs to be recomputed 2^ℓ times in order to correctly construct the commitments $C^{(\ell)}_L, C^{(\ell)}_R$. This is due to the streaming access to Q: we cannot store too many $g^{(\ell)}_b$, and each $g^{(\ell)}_b$ is raised to the value $Q^{(\ell)}(b)$, which depends on 2^ℓ values of Q that appear in gaps of $2^{n-\ell}$ indices. Re-computation of these generators results

[10] ↑We also ignore the terms h^{y_L} and h^{y_R}.
[11] ↑We do not prove that it is impossible to obtain a space-efficient implementation with a nearly linear committer (i.e., $\widetilde{O}(N)$), but it is not clear how to do so with the Bulletproofs two-move reduction.

in a committer that runs in time (roughly) N^2 per round of recursion, which violates our desired goal of a nearly linear-time committer. We emphasize that this occurs *even given random access to the initial generators*. Thus, we must re-think the two-move reduction.

Our Evaluation Phase: Even-Odd Folding

Our solution is to slightly alter the two-move reduction phase to employ an "even-odd" folding, rather than the "left-right" folding of the Bulletproofs two-move reduction. In particular, Bulletproofs folds via the most significant bit, and we choose to fold via the least significant bit. More formally, again fix $(C_y, g, h, z, y, Q) \in \mathcal{R}_{\mathsf{BP}}(n)$ and suppose (C_y, g, h, z, y) are given to both the committer and receiver. Additionally, the committer receives Q. We describe our two-move reduction.

1. The committer computes the cross-product $y_e = \langle Q_e, z_o \rangle$, where $Q_e = (Q(b' \circ 0))_{b' \in \{0,1\}^{n-1}}$ and $z_o = (z_{b' \circ 1})_{b' \in \{0,1\}^{n-1}}$. That is, Q_e consists of elements of Q indexed by even indices, and z_o consists of elements of z indexed by odd indices. Similarly compute $y_o = \langle Q_o, z_e \rangle$, where Q_o consists of element of Q indexed by odd indices, and z_e consists of elements of z indexed by even indices. Furthermore, the committer computes a new commitment to the even vector Q_e and odd vector Q_o as

$$C_e = h^{y_e} \cdot \prod_{b' \in \{0,1\}^{n-1}} g_{o,b'}^{Q_e(b')} \qquad\qquad C_o = h^{y_o} \cdot \prod_{b' \in \{0,1\}^{n-1}} g_{e,b'}^{Q_o(b')}. \tag{4.3}$$

   Here, g_e and g_o are vectors defined by the even and odd indices of $g \in \mathbb{G}^N$, respectively, and both lie in $\mathbb{G}^{N/2}$. The committer sends (y_e, y_o, C_e, C_o).

2. The receiver samples $\alpha \xleftarrow{\$} \mathbb{F}$ and sends α to the committer.

3. The committer and receiver both compute the folding

$$\begin{aligned} z' &= \alpha^{-1} \cdot z_e + \alpha \cdot z_o \in \mathbb{F}^{N/2} \\ g' &= (g_e)^{\alpha^{-1}} * (g_o)^{\alpha} \in \mathbb{G}^{N/2} \\ y' &= \alpha^2 \cdot y_e + y + \alpha^{-2} \cdot y_o \in \mathbb{F} \\ C'_{y'} &= C_e^{\alpha^2} \cdot C_y \cdot C_o^{\alpha^{-2}} \in \mathbb{G}. \end{aligned}$$

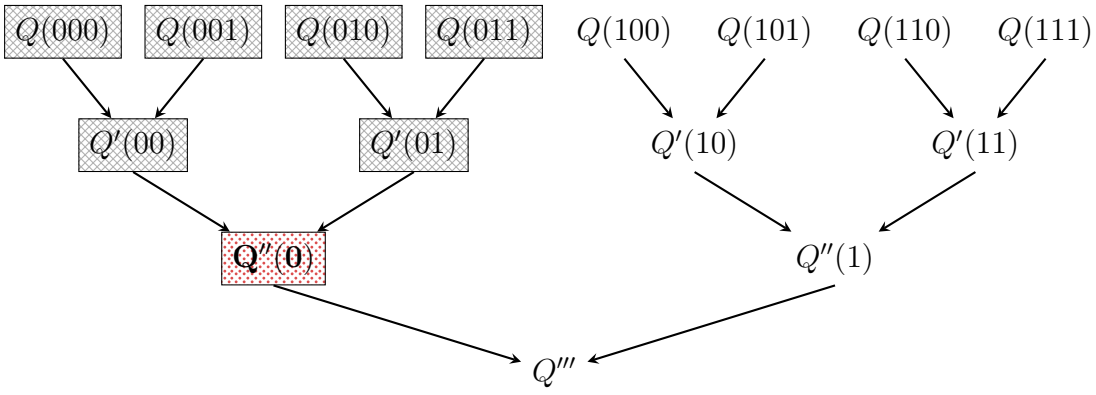

Figure 4.3. Example of the recursion tree induced by our two-move reduction for $n = 3$, $N = 2^3$, and the polynomial $Q \in \mathbb{F}^N$. A child node is obtained by taking a linear combination of the parent nodes. In particular, left edges indicate multiplication of the parent by α and right edges indicate multiplication of the parent by α^{-1}, where α is the receiver challenge sent during the current round of recursion. For example, the value $Q''(0)$ depends on all nodes of the tree with a cross-pattern background.

The committer additionally computes a new folded witness $Q' = \alpha \cdot Q_e + \alpha^{-1} \cdot Q_o$.

Again leveraging the homomorphic properties of Pedersen commitments, one can again show that if Q is a witness for (C_y, g, h, z, y) then Q' is a witness for the tuple $(C'_{y'}, g', h, z', y')$. This forms a new recursion tree which is amenable to streaming. We present the new recursion tree in Figure 4.3.

Space-Efficiency of Even-Odd Folding

As seen in Figure 4.3, the even-odd folding gives a recursion tree that is much more amenable to our streaming model. In particular, again let $Q^{(\ell)} \in \mathbb{F}^{2^{n-\ell}}$ denote the recursive stream at level ℓ of the recursion. Then, since we fold via the *least significant bit*, for any $b \in \{0,1\}^{n-\ell}$ we have that (again ignoring the verifier challenges):

$$Q^{(\ell)}(b) = \sum_{c \in \{0,1\}^{\ell}} Q^{(0)}(b \circ c).$$

In particular, the value $Q^{(\ell)}(b)$ depends on all values of $Q^{(0)}$ with matching *higher-order* bits. Thus the values of $Q^{(0)}$ necessary for computing $Q^{(\ell)}(b)$ appear contiguously in our stream of values. In fact, it is not difficult to see that we can compute the stream $Q^{(\ell)}$ in lexicographic order using a single pass over the stream $Q^{(0)}$ *and* by storing only a constant number of field elements. The following algorithm achieves such a time and space efficient implementation.

1. For $b \in \{0,1\}^{n-\ell}$:
   (a) Initialize value $Q^{(\ell)}(b) = 0 \in \mathbb{F}$.
   (b) For $c \in \{0,1\}^{\ell}$:
      i. Compute $Q^{(\ell)}(b) = Q^{(\ell)}(b) + Q^{(0)}(b \circ c)$.
   (c) Output $Q^{(\ell)}(b)$ and continue to the next iteration.

Now we are able to overcome the the hurdles to space-efficiency of the Bulletproofs two-move reduction. By Equation (4.3), the commitments $C_e^{(\ell)}, C_o^{(\ell)}$ are computed as:

$$C_e^{(\ell)} = h^{y_e} \cdot \prod_{b' \in \{0,1\}^{n-\ell}} (g_{o,b'}^{(\ell)})^{Q_e^{(\ell)}(b')} \qquad C_o^{(\ell)} = h^{y_o} \cdot \prod_{b' \in \{0,1\}^{n-\ell}} (g_{e,b'}^{(\ell)})^{Q_o^{(\ell)}(b')}.$$

Recall that the obstacle we faced for a time- and space-efficient implementation of the Bulletproofs two-move reduction was that in order to run in small-space, the prover would run in quadratic time to compute these commitments. This was due to the fact that the values that $Q_L^{(\ell)}(b')$ and $Q_R^{(\ell)}(b')$ depend on occur in gaps of $2^{n-\ell}$ in the stream, and thus we had to recompute the generators $g_{L,b'}^{(\ell)}$ and $g_{R,b'}^{(\ell)}$ from scratch every time. However, with our new even-odd folding, this is not the case: the values of the top-level stream that $Q_e^{(\ell)}(b')$ and $Q_o^{(\ell)}(b')$ depend on all occur in a contiguous block. This means that we only need to compute $g_{e,b'}^{(\ell)}$ and $g_{o,b'}^{(\ell)}$ *once*. The following algorithm highlights this change.

1. Initialize values $C_e^{(\ell)} = C_o^{(\ell)} = 1_{\mathbb{G}}$.
2. For $b = (b_1, \ldots, b_{n-\ell}) \in \{0,1\}^{n-\ell}$:
   (a) Initialize $Q^{(\ell)}(b) = 0 \in \mathbb{F}$.
   (b) For $c = (c_1, \ldots, c_\ell) \in \{0,1\}^{\ell}$:

i. Compute $Q^{(\ell)}(b) = Q^{(\ell)}(b) + Q^{(0)}(b \circ c)$.

(c) Let $b' = (b_1, \ldots, b_{n-\ell-1})$.

(d) If $b_{n-\ell} = 0$ then compute $C_e^{(\ell)} = C_e^{(\ell)} \cdot (g_{b' \circ 1}^{(\ell)})^{Q^{(\ell)}(b)}$. Else compute $C_o^{(\ell)} = C_o^{(\ell)} \cdot (g_{b' \circ 0}^{(\ell)})^{Q^{(\ell)}(b)}$.

Since we only need to compute the folded generators once, we can simply compute the necessary generator, update the current commitment value, then proceed to the next needed generator. In total, this gives a committer that performs $O(N)$ total operations per round of recursion, giving a $O(N \cdot \log(N))$ time committer.

We note here that we still need *random access* to the original generators $g \in \mathbb{G}^N$; in particular, we do not know how to obtain a space-efficient implementation assuming some from of streaming access to these generators.[12] For this reason, we choose to model the group $\mathbb{G}$ as a random oracle. In particular, we assume that for security parameter $\lambda > n$ we have a random oracle $H \colon \{0,1\}^\lambda \to \mathbb{G}$ and define $g_b = H(b)$ for $b \in \{0,1\}^n$, and randomly sample from group $\mathbb{G}$ by randomly sampling $r \xleftarrow{\$} \{0,1\}^\lambda$ and querying the oracle at $H(r)$.

This completes the overview of the ideas behind our space-efficient polynomial commitment satisfying Theorem 4.2.1. We discuss the formal details and theorem in Section 4.6.4.

4.3.3 Overview of Theorem 4.2.2

Our second polynomial commitment scheme is based on the recent so-called DARK polynomial commitment scheme due to Bünz, Fisch, and Szepieniec [76]. We directly describe our modified commitment scheme based on the DARK polynomial commitment. Our scheme is tailored for multilinear polynomials, and is designed to overcome a gap in the security proof of the original DARK scheme. We defer the reader to [52] for details on this gap, as this is not the focus of this dissertation. Again throughout we fix a multilinear polynomial $Q \colon \mathbb{F}^n \to \mathbb{F}$ with evaluations $(Q(b))_{b \in \{0,1\}^n}$ and equivalently write $Q \in \mathbb{F}^N$ for $N = 2^n$.

[12] ↑Though we conjecture that such an implementation exists, eliminating the need for a random oracle.

Commitment Phase

Our commitment phase is identical to the DARK commitment phase. In particular, we commit to the evaluations $Q \in \mathbb{F}^N$ via encoding Q as a large integer. Let $\mathbb{F}$ be a finite field of prime order p. Commitment to Q is computed by first constructing a large integer $\mathcal{Z}(Q)$ such that the base-q digits of $\mathcal{Z}(Q)$ correspond to the values of Q (modulo p). That is, $\mathcal{Z}(Q) := \sum_{b \in \{0,1\}^n} q^b \cdot Q(b)$, where $q \gg p$ is a large integer and we interpret q^b as exponentiation by the integer uniquely represented by b in the natural way.

Now committing to Q is simply $C = g^{\mathcal{Z}(Q)}$, where $g \in \mathbb{G}$ is a random element of the group $\mathbb{G}$ for which the hidden order assumption holds, and is additionally specified during the generation of public parameters. For arbitrary integer $Z \in \mathbb{Z}$, we say that Z is *consistent* with our polynomial Q if the base-q representation of Z, after reducing each digit modulo p, results in the original sequence $Q \in \mathbb{F}^N$. Since $q \gg p$, there are many integers Z which are consistent with Q, including our constructed integer $\mathcal{Z}(Q)$. However, the commitment C is a binding commitment since finding another integer $Z \neq \mathcal{Z}(Q)$ that is consistent with Q *and* the commitment C (e.g., $g^{\mathcal{Z}(Q)} = g^Z$) reveals (a multiple of) the order of g, which breaks the hidden order assumption.

Crucial to our evaluation phase is that this commitment is homomorphic in the following sense. Given two integers Z_1, Z_2 consistent with two multilinear polynomials Q_1, Q_2, respectively, that have *sufficiently small digits* in their base-q representation, it holds that $Z_1 + Z_2$ is consistent with the polynomial $Q_1 + Q_2$ (modulo p). This homomorphism is similarly true for multiplications by scalars: if α is a sufficiently small integer, then $\alpha \cdot Z_1$ is consistent with the polynomial $\alpha \cdot Q_1$ (modulo p). In our evaluation proofs, we make extensive use of the first homomorphic property.

Evaluation Proofs

Similar to the evaluation proof of our previous polynomial commitment scheme, our evaluation proof is performed again via a two-move reduction, reducing a statement of size N to a statement of size $N/2$. Given an evaluation point $\zeta \in \mathbb{F}^n$ specified by the receiver,

the committer computes and sends $y = Q(\zeta)$. The committer and receiver then engage in an interactive argument (of knowledge) for the following NP relation:

$$\mathcal{R}(n) = \left\{ (C, \zeta, g, q, Z, y; Q) : \begin{array}{c} Z \in \mathbb{F}^N, Q \in \mathbb{Z}^N, \zeta \in \mathbb{F}^n, g \in \mathbb{G} \\ \wedge \ y = \langle Q \ (\mathrm{mod}\, p), Z \rangle \in \mathbb{F} \\ \wedge \ C = g^{\mathcal{Z}(Q)} \in \mathbb{G} \\ \wedge \ \mathcal{Z}(Q) = \sum_{b \in \{0,1\}^n} q^b \cdot Q(b) \in \mathbb{Z} \end{array} \right\}, \tag{4.4}$$

where $N = 2^n$ and $g \in \mathbb{G}$ are given as above and $Z := (\overline{\chi}(b, \zeta))_{b \in \{0,1\}^n}$ for $\overline{\chi}(b, \zeta) = \prod_i \chi(b_i, \zeta_i)$. In Equation (4.4), we quantify with respect to *integer sequences* $Q \in \mathbb{Z}^N$ rather than sequences over the field $\mathbb{F}$; looking ahead, this is to ensure during our evaluation proofs we are able to keep commitments consistent.

We again employ a natural "spit-and-fold" recursive argument to construct an argument system for the above relation. In fact, rather than perform a 1-2-1 split-and-fold reduction, we employ a λ_{s}-$2\lambda_{\mathsf{s}}$-λ_{s}, where $\lambda_{\mathsf{s}} \in \mathbb{N}$ is a *statistical security parameter*. Suppose we have λ_{s} statements $(C_i, \zeta, g, q, Z_i, y_i; Q_i) \in \mathcal{R}(n)$ for $i \in [\lambda_{\mathsf{s}}]$.[13] Then for each $i \in [\lambda_{\mathsf{s}}]$, the committer computes a simple left-right split of the evaluations of Q_i on point ζ. That is, the committer computes

$$y_{i,0} = \sum_{b \in \{0,1\}^{n-1}} (Q_i(0 \circ b) \bmod p) \cdot \prod_{j=1}^{n-1} \chi(b_j, \zeta_j)$$
$$y_{i,1} = \sum_{b \in \{0,1\}^{n-1}} (Q_i(1 \circ b) \bmod p) \cdot \prod_{j=1}^{n-1} \chi(b_j, \zeta_j),$$

where we interpret $b \in \{0,1\}^n$ as $b = (b_n, b_{n-1}, \ldots, b_1)$ and $\zeta \in \mathbb{F}^n$ as $\zeta = (\zeta_n, \zeta_{n-1}, \ldots, \zeta_1)$. The committer then computes the left-right split of the commitments to each polynomial Q_i as

$$C_{i,0} = g^{\ell_i} \qquad\qquad \ell_i = \sum_{b \in \{0,1\}^{n-1}} q^b \cdot Q_i(0 \circ b)$$

[13]↑We can, in fact, prove λ_{s} independent statements (all with the same evaluation point ζ), or prove a single statement by copying it λ_{s} times and performing the same protocol.

$$C_{i,1} = g^{r_i} \qquad\qquad r_i = \sum_{b\in\{0,1\}^{n-1}} q^b \cdot Q_i(1 \circ b).$$

Given the values $y_{i,0}, y_{i,1}$ and $C_{i,0}, C_{i_1}$, we observe that given the original statement $(C_i, \zeta, g, q, Z_i, y_i; Q_i)$, we have that $y_i = y_{i,0} \cdot (1 - \zeta_n) + y_{i,1} \cdot \zeta_n$ and $C_i = C_{i,0} \cdot C_{i,1}^{q^{N/2}}$ where $N = 2^n$. Thus the committer sends the values $y_{i,0}, y_{i,1}$ and $C_{i,0}, C_{i_1}$ to the committer, and the verifier performs the above checks. Then we perform a *binary* folding of the $2\lambda_{\mathsf{s}}$ statements, inspired by LT Codes [191]. For every $i \in [\lambda_{\mathsf{s}}]$, the receiver samples vectors $u_{i,0}, u_{i,1} \xleftarrow{\$} \{0,1\}^{\lambda}$ and sends these vectors to the committer. Then for every $i \in [\lambda_{\mathsf{s}}]$, both the committer and receiver compute new folded values

$$y'_i = \langle u_{i,0}, (y_{1,0}, \ldots, y_{\lambda_{\mathsf{s}},0})\rangle + \langle u_{i,1}, (y_{1,1}, \ldots, y_{\lambda_{\mathsf{s}},1})\rangle \in \mathbb{F} \quad C'_i = \prod_{j\in[\lambda_{\mathsf{s}}]} C_{j,0}^{u_{i,0}(j)} \cdot C_{j,1}^{u_{i,1}(j)} \in \mathbb{G};$$

further, the committer additionally computes for every $i \in [\lambda_{\mathsf{s}}]$ and every $b \in \{0,1\}^{n-1}$

$$Q'_i(b) = \sum_{j\in[\lambda_{\mathsf{s}}]} u_{i,0}(j) \cdot Q_j(0 \circ b) + u_{i,1}(j) \cdot Q_j(1 \circ b).$$

The committer defines $Q'_i := (Q'_i(b))_{b\in\{0,1\}^{n-1}}$. The committer and receiver then recurse on the λ_{s} new statements $(C'_i, \zeta' = \zeta \setminus \{\zeta_n\}, g, q, Z'_i, y'_i; Q'_i)$ where $Z'_i = (\overline{\chi}(b', \zeta'))_{b'\in\{0,1\}^{n-1}}$. The recursion continues for $n + 1$ steps.

Space-Efficient Implementation

The above evaluation phase admits a straight-forward space-efficient implementation in the streaming model. This is due to the fact that all values computed by the prover are simple linear functions; that is, the order in which we compute the values does not matter since the function being computed is a linear function. During each round of the recursion, computing $y_{i,0}, y_{i,1}$ and $C_{i,0}, C_{i,1}$ can be done in a single pass over the initial set of streams from the 0^{th} level of recursion. We give formal details in Section 4.7.3.

Verifier Efficiency

The verifier in the above protocol does not meet our efficiency target. This is because the verifier computes $q^{N/2}$ and performs expensive group exponents. Even with repeated squaring, this results in a verifier time roughly $O(N)$, which is undesirable; moreover, this is less efficient than the verifier in the scheme presented in Theorem 4.6.4. Thus we offload this computational complexity of the verifier to the prover via a proof of exponent protocol. We introduce a *statistically secure* variant of Pietrzak's proof of exponent protocol [219] which is secure over *any* group, and use this protocol to achieve greater verifier efficiency with no cost to the asymptotic complexity of the prover. Moreover, we implement this proof of exponent in a space-efficient manner, which results in a final verifier complexity of polylog(n) time.

4.3.4 Obtaining Space-Efficient Succinct Arguments

Recall that in Section 4.1 we gave a general approach to constructing arguments for the relation $\mathcal{R}_{\text{RAM}}$. We adopt that approach to obtain space-efficient succinct arguments for $\mathcal{R}_{\text{RAM}}$ by combining either of our polynomial commitment schemes with the Clover MIP. We discuss this transformation in detail in Section 4.8 and give a high-level overview here.

Given any time-T and space-S RAM M, we leverage the Clover MIP's streaming algorithm to stream the wire transcript $W\colon \{0,1\}^s \to \mathbb{F}$ in lexicographic order. Streaming this transcript takes time $T \cdot \text{polylog}(T)$ and space $S \cdot \text{polylog}(T)$ [51, 64]. We compose this stream with either of our streamable polynomial commitments, i.e., Theorems 4.2.3 and 4.2.4 to commit to the wire transcript. Then, we follow the Clover MIP which constructs additional polynomials to encode wire constraints (i.e., addition and multiplication constraints), which culminates in some auxiliary polynomial $F\colon \mathbb{F}^{3s} \to \mathbb{F}$ such that $F \equiv 0$ if and only if W is correct. The prover and verifier then engage in the sum-check protocol [192] for the polynomial F, which induces $O(1)$ evaluations for the polynomial $\widetilde{W}$. Finally, using the evaluation protocols of the polynomial commitment schemes, the verifier obtains these evaluations of $\widetilde{W}$ and finishes off the sum-check protocol, accepting or rejecting appropriately.

Implicit Use of Coding Theory

Relating back to our thesis statement, the final time- and space-efficient arguments obtained by composing the Clover MIP with our polynomial commitments has soundness which relies on two key components. First is the (knowledge) soundness of the polynomial commitment schemes, which relies on the particular cryptographic assumptions we make for these protocols. Second is the soundness of the sum-check protocol and the Clover MIP, which reduces to performing low-degree tests on Reed-Muller codewords. These low-degree tests rely on the definition and guarantees of Reed-Muller codewords (i.e., guarantees about polynomials via theorems like the Schwartz-Zippel Lemma). So the final soundness guarantee reduces to a coding theoretic argument about correctness of a codeword and distance guarantees given by Reed-Muller codewords.

4.4 Additional Related Work

4.4.1 Polynomial Commitments

Polynomial commitment schemes were introduced by Kate, Zaverucha, and Goldberg [165]. As discussed above, such commitments allow one to commit to a polynomial and later answer evaluation queries while proving consistency with the commitment without revealing the entire polynomial.

There are several variants of polynomial commitments include privately verifiable schemes [165, 217], publicly-verifiable schemes with trusted setup [76], and zero-knowledge schemes [258]. More recently, much focus has been on obtaining publicly-verifiable schemes without a trusted setup [30, 36, 67, 76, 166, 184, 232, 255, 266]. In all prior work, the space complexity of the committer is proportional to the description size of the polynomial. Recently, [68] adapt the polynomial commitment scheme of [165] to the streaming setting and construct what they call "Elastic SNAKRs". Such SNARKs have two modes: a time-optimal mode and a space-efficient mode, allowing the prover to swap between either mode as necessary.

Lastly, we mention that classical works on low degree testing (à la [228]) as well as more recent works [30, 32, 36] can be used to construct polynomial-commitments by Merkle hashing the entire truth table of the polynomial (and using a self-correction procedure or protocol).

4.4.2 Privately Verifiable Proofs

The question of constructing proof systems in which the prover is efficient both in terms of time *and* space was first raised by Bitansky and Chiesa [41], who constructed a time- and space-efficient (or in their terminology *complexity preserving*) interactive argument for any problem in NP based on fully homomorphic encryption. Holmgren and Rothblum [157] constructed *non-interactive* time- and space-efficient arguments for P based on the (sub-exponential) learning with errors assumption. The protocols of [41, 157] are *privately verifiable*, meaning that only a designated verifier (who knows the randomness used to sampled the verifier messages) is able to verify the proof.

4.4.3 Proofs by Recursive Composition

An alternative approach to *publicly verifiable* time- and space-efficient arguments is by recursively composing SNARKs for NP [40, 251]. Recursive composition requires both the prover and verifier to make non-black-box usage of an "inner" verifier for a different SNARK, leading to large computational overhead. Several recent works [69, 73, 91] attempt to solve the inefficiency problems with recursive composition, but at additional expense to the underlying cryptographic assumptions. These works rely on hash functions that are modeled as random oracles in the security proof despite being used in a *non-black-box* way by the honest parties. Security thus cannot be reduced to a simple computational hardness assumption, even in the random oracle model. Moreover, the practicality of the schemes crucially requires usage of a novel hash function (e.g., Rescue [11]) with algebraic structure designed to maximize the efficiency of non-black-box operations. Such hash functions have endured far less scrutiny than standard SHA hash functions, and the algebraic structure could potentially lead to a security vulnerability (though no such vulnerabilities are known at the time of this writing to the best of our knowledge). We also mention a recent work of Ephraim et al. [110] which uses

recursive composition to address the related question of implementing the prover in small *depth* (i.e., parallel time).

4.4.4 Multi-Prover Proofs

Bitansky and Chiesa [41], as well as Blumberg et al. [64], construct time- and space-efficient *multi-prover* interactive proofs; that is, soundness only holds under the assumption that the provers do not collude. Justifying this assumption in practice seems difficult and indeed multi-prover interactive proofs are usually only used as building blocks toward more complex systems.

4.5 Preliminaries

We state preliminaries which common to both of our polynomial commitment schemes. Unfortunately, due to the differences in models between our polynomial commitment schemes, some definitions (e.g., definition of a polynomial commitment scheme) differ. Thus rather than try to unify the definitions, we simply present appropriate definitions for the scheme being discussed later in the text. See Sections 4.6.1 and 4.7.1.

4.5.1 Notation

We let λ denote the security parameter, let $n \in \mathbb{N}$ and $N = 2^n$. We let $\mathsf{Primes}(1^\lambda)$ denote the set of all λ-bit primes. For bit string $b \in \{0,1\}^n$, we write $b = (b_n, \ldots, b_1)$, where b_n is the most significant bit and b_1 is the least significant bit. Similarly, for vectors of field elements $\zeta \in \mathbb{F}^n$, we write $\zeta = (\zeta_n, \ldots, \zeta_1)$.

4.5.2 Multilinear Polynomials

An n-variate polynomial $f \colon \mathbb{F}^n \to \mathbb{F}$ is *multilinear* if the individual degree of each variable in f is at most 1. We recall Fact 4.3.1 here.

Fact 4.3.1 ([51, 213])**.** *An multilinear polynomial $f : \mathbb{F}^n \to \mathbb{F}$ (over a finite field $\mathbb{F}$) is uniquely defined by its evaluations over the Boolean hypercube. Moreover, for every $\zeta \in \mathbb{F}^n$ it holds that*

$$f(\zeta) = \sum_{b \in \{0,1\}^n} f(b) \cdot \prod_{i=1}^{n} \chi(b_i, \zeta_i), \tag{4.1}$$

where $\chi(b_i, \zeta_i) = b_i \cdot \zeta_i + (1 - b_i) \cdot (1 - \zeta_i)$.

For ease of presentation, when $|b| = |\zeta| = n$ we let $\overline{\chi}(b, \zeta) := \prod_i \chi(b_i, \zeta_i)$.

Multilinear Polynomial Notation

For an n-variate multilinear polynomial f over field $\mathbb{F}$, we represent f as the unique N-sized sequence $Y \in \mathbb{F}^N$ for $N = 2^n$ such that $Y_b = f(b)$ for all $b \in \{0,1\}^n$. Similarly, for a sequence $Y \in \mathbb{F}^N$, we denote the evaluation of a multilinear polynomial defined by Y at a point $\zeta \in \mathbb{F}^n$ as $\mathsf{ML}(Y, \zeta) := \sum_b Y_b \cdot \overline{\chi}(b, \zeta)$. We also consider evaluating a multilinear polynomial defined by some integer sequence $Z \in \mathbb{Z}^N$. For prime p such that $|\mathbb{F}| = p$ and any $\zeta \in \mathbb{F}^n$, we define $\mathsf{ML}(Z, \zeta) := \sum_b (Z_b \bmod p) \cdot \overline{\chi}(b, \zeta)$.

Streaming Model for Multilinear Polynomials

For our commitment scheme, we assume that the committer will have *multi-pass streaming* access to the function table Y of f (which defines the multi-linear polynomial) in the lexicographic ordering. Specifically, the committer will be given access to a read-only tape that is pre-initialized with the sequence $Y = \left(Y_b = f(b) : b \in \{0,1\}^n\right)$. At every time-step the committer is allowed to either move the machine head to the right one position, or to restart its position to 0.

With the above notation, we can now view $\mathsf{ML}(Y, \zeta \in \mathbb{F}^n)$ as an inner-product between Y and $Z = (z_b = \overline{\chi}(b, \zeta) : b \in \{0,1\}^n)$ where computing z_b requires $O(n = \log(N))$ field multiplications for fixed $\zeta \in \mathbb{F}^n$ any $b \in \{0,1\}^n$.

4.5.3 Assumptions on Groups

Definition 4.5.1 (Group Sampler)**.** *A PPT algorithm $\mathcal{G}$ is a* group sampler *if for every $\lambda \in \mathbb{N}$, on input 1^λ the algorithm $\mathcal{G}$ samples a group description*[14] *$\mathbb{G}$ of a group of size at most 2^λ. As a shorthand, we denote this random process by $\mathbb{G} \leftarrow \mathcal{G}(1^\lambda)$, and by $g \xleftarrow{\$} \mathbb{G}$ denote the process of sampling a random group element g from $\mathbb{G}$. Furthermore, we say that $\mathcal{G}$ is* public-coin *if the output of $\mathcal{G}$ (i.e., the group description) is a uniformly random string.*

We focus on group samplers $\mathcal{G}$ for which either the Discrete-log Assumption or the Hidden Order Assumption holds. We discuss the discrete-log assumption first. Informally, the discrete-log assumption states that given a generator of a group g and a random α, it is computationally difficult to compute α given g^α. In the case of discrete-log, the group sampler additionally outputs a λ-bit prime p (the order of $\mathbb{G}$) and a generator g of $\mathbb{G}$.

Assumption 4.5.1 (Discrete-log Assumption)**.** *Let $\mathcal{G}$ be a group sampler such that $(\mathbb{G}, p, g) \leftarrow \mathcal{G}(1^\lambda)$, where $\mathbb{G}$ is the description of a group of prime order p, p is a λ-bit prime, and g is a generator of $\mathbb{G}$. The* discrete-log assumption *holds for $\mathcal{G}$ if for every polynomial-sized family of circuits $\mathcal{A} = \{\mathcal{A}_\lambda\}_{\lambda\in\mathbb{N}}$, there exists a negligible function μ such that:*

$$\Pr\left[\alpha = \alpha' : \begin{array}{c} (\mathbb{G}, p, g) \leftarrow \mathcal{G}(1^\lambda); \alpha \xleftarrow{\$} \mathbb{Z}_p \\ \alpha' \leftarrow \mathcal{A}_\lambda(\mathbb{G}, p, g, g^\alpha) \end{array}\right] \leqslant \mu(\lambda).$$

Since one of our polynomial commitment schemes is based on Pedersen commitments, we use the following variant of the discrete-log assumption.

Assumption 4.5.2 (Discrete-log Relation Assumption [67])**.** *The* discrete-log relation assumption *holds for group sampler $\mathcal{G}$ if for every polynomial-sized family of circuits $\mathcal{A} = \{\mathcal{A}_\lambda\}_{\lambda\in\mathbb{N}}$ and all $n \geqslant 2$, there exists a negligible function μ such that:*

$$\Pr\left[\exists \alpha_i \neq 0 \ \wedge \ \prod_{i=1}^{n} g_i^{\alpha_i} = 1 : \begin{array}{c} (\mathbb{G}, p, g) \leftarrow \mathcal{G}(1^\lambda); g_1, \ldots, g_n \xleftarrow{\$} \mathbb{G}; \\ \mathbb{Z}_p^n \ni (\alpha_1, \ldots, \alpha_n) \leftarrow \mathcal{A}_\lambda(\mathbb{G}, p, g, \{g_i\}_{i\in[n]}) \end{array}\right] \leqslant \mu(\lambda).$$

[14] The group description includes a $\mathrm{poly}(\lambda)$ description of the identity element, and $\mathrm{poly}(\lambda)$ size circuits checking membership in the group, equality, performing the group operation and generating a random element in the group.

We say that $\prod_{i=1}^{n} g_i^{\alpha_i} = 1$ is a non-trivial discrete-log relation between $g_1, \ldots, g_n$ when not all α_i are zero. Thus Assumption 4.5.2 says an adversary cannot find non-trivial discrete-log relations between random group elements.

We now turn to the Hidden Order Assumption. Informally, the hidden order assumption states that it is computationally hard to find (any multiple of) the order of a random group element g of $\mathbb{G} \leftarrow \mathcal{G}(1^\lambda)$.

Assumption 4.5.3 (Hidden Order Assumption)**.** *The* hidden order assumption *holds for $\mathcal{G}$ if for every polynomial-sized family of circuits $\mathcal{A} = \{\mathcal{A}_\lambda\}_{\lambda \in \mathbb{N}}$, there exists a negligible function μ such that:*

$$\Pr\left[g^a = 1 \quad \wedge \quad a \neq 0 \; : \; \begin{array}{c} \mathbb{G} \leftarrow \mathcal{G}(1^\lambda), g \leftarrow \mathbb{G} \\ a \leftarrow \mathcal{A}_\lambda(\mathbb{G}, g) \end{array} \right] \leqslant \mu(\lambda).$$

4.6 Streamable Polynomial Commitment Scheme for Multilinear Polynomials from Discrete-log in the Random Oracle Model

In this section, we give our polynomial commitment scheme which realizes Theorem 4.2.3. Section 4.6.1 gives relevant preliminaries for this section. Section 4.6.2 gives the polynomial commitment scheme. Section 4.6.3 proves the completeness and security of the scheme. Section 4.6.4 proves the efficiency of the scheme.

4.6.1 Preliminaries

Random Oracles

We let $\mathcal{U}(\lambda)$ denote the set of all functions that map $\{0,1\}^*$ to $\{0,1\}^\lambda$. A random oracle with security parameter λ is a function $\rho : \{0,1\}^* \to \{0,1\}^\lambda$ sampled uniformly at random from $\mathcal{U}(\lambda)$.

Interactive Arguments of Knowledge in the Random Oracle Model

Definition 4.6.1 (Witness Relation Ensemble)**.** *A* witness relation ensemble *or relation ensemble is a ternary relation $\mathcal{R}_L$ that is polynomially bounded, polynomial time recognizable*

and defines a language $\mathcal{L} = \{(pp, x) : \exists w \text{ s.t. } (pp, x, w) \in \mathcal{R}_{\mathcal{L}}\}$. *We omit pp when considering languages recognized by binary relations.*

Definition 4.6.2 (Interactive Arguments [132])**.** *Let* $\mathcal{R}$ *be some relation ensemble. Let* (P, V) *denote a pair of PPT interactive algorithms and* Setup *denote a non-interactive setup algorithm that outputs public parameters pp given security parameter* 1^λ. *Let* $\langle P(pp, x, w), V(pp, x)\rangle$ *denote the output of* V*'s interaction with* P *on common inputs public parameter pp and statement* x *where additionally* P *has the witness* w. *The triple* (Setup, P, V) *is an* argument for $\mathcal{R}$ *in the random oracle model (ROM) if*

1. *Perfect Completeness. For any adversary* A

$$\Pr\left[(x, w) \notin \mathcal{R} \text{ or } \langle P^\rho(pp, x, w), V^\rho(pp, x)\rangle = 1\right] = 1 \ ,$$

where probability is taken over $\rho \xleftarrow{\$} \mathcal{U}(\lambda), pp \xleftarrow{\$} \mathsf{Setup}^\rho(1^\lambda), (x, w) \xleftarrow{\$} A^\rho(pp)$.

2. *Computational Soundness. For any non-uniform PPT adversary* A

$$\Pr\left[\forall w \ (x, w) \notin \mathcal{R} \text{ and } \langle A^\rho(pp, x, st), V^\rho(pp, x)\rangle = 1\right] \leq \mathsf{negl}(\lambda) \ ,$$

where probability is taken over $\rho \xleftarrow{\$} \mathcal{U}(\lambda), pp \xleftarrow{\$} \mathsf{Setup}^\rho(1^\lambda), (x, st) \xleftarrow{\$} A^\rho(pp)$.

Remark 4.6.1. *Usually completeness is required to hold for all* $(x, w) \in \mathcal{R}$. *However, for the argument systems used in this work, statements* x *depends on pp output by* Setup *and the random oracle* ρ. *We model this by asking for completeness to hold for statements sampled by an adversary* A, *that is, for* $(x, w) \xleftarrow{\$} A(pp)$.

For our applications, we will need (Setup, P, V) to be an argument of knowledge. Informally, in an argument of knowledge for $\mathcal{R}$, the prover convinces the verifier that it "knows" a witness w for x such that $(x, w) \in \mathcal{R}$. In this paper, knowledge means that the argument has *witness-extended emulation* [138, 186].

Definition 4.6.3 (Witness-extended Emulation)**.** *Given a public-coin interactive argument tuple* (Setup, P, V) *and some arbitrary prover algorithm* P^*, *let* $\mathsf{Record}(P^*, pp, x, st)$ *denote*

the message transcript between P^* and V on shared input x, initial prover state st, and pp generated by Setup. *Furthermore, let* $\mathcal{E}^{\mathsf{Record}(P^*,pp,x,st)}$ *denote a machine* $\mathcal{E}$ *with a transcript oracle for this interaction that can be rewound to any round and run again on fresh verifier randomness. The tuple* (Setup, P, V) *has witness-extended emulation if for every deterministic polynomial-time* P^* *there exists an expected polynomial-time emulator* $\mathcal{E}$ *such that for all non-uniform polynomial-time adversaries* A *the following holds:*

$$\Pr\left[A^{\rho}(tr) = 1 : \begin{array}{c} \rho \xleftarrow{\$} \mathcal{U}(\lambda),\ pp \xleftarrow{\$} \mathsf{Setup}^{\rho}(1^{\lambda}), \\ (x, st) \xleftarrow{\$} A^{\rho}(pp),\ tr \xleftarrow{\$} \mathsf{Record}^{\rho}(P^*, pp, x, st) \end{array}\right] \approx$$

$$\Pr\left[\begin{array}{c} A^{\rho}(tr) = 1 \text{ and} \\ tr \text{ accepting} \implies (x,w) \in \mathcal{R} \end{array} : \begin{array}{c} \rho \xleftarrow{\$} \mathcal{U}(\lambda),\ pp \xleftarrow{\$} \mathsf{Setup}^{\rho}(1^{\lambda}), \\ (x, st) \xleftarrow{\$} A^{\rho}(pp), \\ (tr, w) \xleftarrow{\$} \mathcal{E}^{\rho, \mathsf{Record}^{\rho}(P^*,pp,x,st)}(pp, x) \end{array}\right]$$

It was shown in [67, 76] that witness-extended emulation is implied by an extractor that can extract the witness given a tree of accepting transcripts. For completeness we state this—dubbed Generalized Forking Lemma—more formally below but refer to [67, 76] for the proof.

Definition 4.6.4 (Tree of Accepting Transcripts)**.** *An* $(n_1, \ldots, n_r)$*-tree of accepting transcripts for an interactive argument on input* x *is defined as follows: The root of the tree is labeled with the statement* x*. The tree has* r *depth. Each node at depth* $i < r$ *has* n_i *children, and each child is labeled with a distinct value for the* i*-th challenge. An edge from a parent node to a child node is labeled with a message from* P *to* V*. Every path from the root to a leaf corresponds to an accepting transcript, hence there are* $\prod_{i=1}^{r} n_i$ *distinct accepting transcripts overall.*

Lemma 4.6.2 (Generalized Forking Lemma [67, 76])**.** *Let* (Setup, P, V) *be an* r*-round public-coin interactive argument system for a relation* $\mathcal{R}$*. Let* T *be a tree-finder algorithm that, given access to a* $\mathsf{Record}(\cdot)$ *oracle with rewinding capability, runs in polynomial time and outputs an* $(n_1, \ldots, n_r)$*-tree of accepting transcripts with overwhelming probability. Let* Ext *be a deterministic polynomial-time extractor algorithm that, given access to* T*'s output, outputs*

a witness w for the statement x with overwhelming probability over the coins of T. Then, (P, V) has witness-extended emulation.

Definition 4.6.5 (Public-coin)**.** *An argument of knowledge is called* public-coin *if all messages sent from the verifier to the prover are chosen uniformly at random and independently of the prover's messages, i.e., the challenges correspond to the verifier's randomness ρ.*

Zero-Knowledge

We also need our argument of knowledge to be zero-knowledge, that is, to not leak partial information about w apart from what can be deduced from $(x, w) \in \mathcal{R}$.

Definition 4.6.6 (Zero-knowledge Arguments)**.** *Let (Setup, P, V) be an public-coin interactive argument system for witness relation ensemble $\mathcal{R}$. Then, (Setup, P, V) has computational zero-knowledge with respect to an auxiliary input if for every PPT interactive machine V^*, there exists a PPT algorithm S, called the simulator, running in time polynomial in the length of its first input, such that for every $(x, w) \in \mathcal{R}$ and any $z \in \{0,1\}^*$:*

$$\mathrm{View}(\langle P(w), V^*(z)\rangle(x)) \approx_c S(x, z) \ ,$$

where $\mathrm{View}(\langle P(w), V^*(z)\rangle(x))$ denotes the distribution of the transcript of interaction between P and V^*, and $\approx_c$ denotes that the two quantities are computationally indistinguishable. If the statistical distance between the two distributions is negligible then the interactive argument is said to be statistical zero-knowledge. If the simulator is allowed to abort with probability at most $1/2$, but the distribution of its output conditioned on not aborting is identically distributed to $\mathrm{View}(\langle P(w), V^*(z)\rangle(x))$, then the interactive argument is called perfect zero-knowledge.

Multilinear Polynomial Commitment Scheme in the Random Oracle Model

In defining the syntax of various protocols, we use the following convention for any list of arguments or returned tuple $(a, b, c; d, e)$: variables listed before semicolon are known both to the prover and verifier whereas the ones after are only known to the prover; e.g., here a, b, c are public and d, e are secret. The semicolon is omitted when there is no secret information.

Definition 4.6.7 (Commitment to Multilinear Polynomials)**.** *A polynomial commitment to multilinear polynomials is a tuple of protocols* $(\mathsf{Setup}, \mathsf{Com}, \mathsf{Open}, \mathsf{Eval})$*:*

1. $pp \leftarrow \mathsf{Setup}^{\rho}(1^{\lambda}, 1^{N})$ *takes as input the unary representations of security parameter* $\lambda \in \mathbb{N}$ *and size parameter* $N = 2^{n}$ *corresponding to* $n \in \mathbb{N}$*, and produces public parameter* pp*. We allow* pp *to contain the description of the field* $\mathbb{F}$ *over which the multi-linear polynomials will be defined.*

2. $(C; d) \leftarrow \mathsf{Com}^{\rho}(pp, Y)$ *takes as input public parameter* pp *and sequence* $Y = (y_b : b \in \{0,1\}^{n}) \in \mathbb{F}^{N}$ *that defines the multi-linear polynomial to be committed, and outputs public commitment* C *and secret decommitment* d*.*

3. $b = \mathsf{Open}^{\rho}(pp, C, Y, d)$ *takes as input* pp*, a commitment* C*, sequence committed* Y *and a decommitment* d *and returns a decision bit* $b \in \{0,1\}$*.*

4. $\mathsf{Eval}^{\rho}(pp, C, \zeta, \gamma; Y, d)$ *is a public-coin interactive protocol between a prover* P *and a verifier* V *with common inputs—public parameter* pp*, commitment* C*, evaluation point* $\zeta \in \mathbb{F}^{n}$ *and claimed evaluation* $\gamma \in \mathbb{F}$*, and prover has secret inputs* Y *and* d*. The prover then engages with the verifier in an interactive argument system for the relation*

$$\mathcal{R}_{\mathsf{mle}}(pp) = \{(C, \zeta, \gamma; Y, d) : \mathsf{Open}^{\rho}(pp, C, Y, d) = 1 \wedge \gamma = \mathsf{ML}(Y, \zeta)\}\,. \tag{4.5}$$

   The output of V *is the output of* Eval *protocol.*

Furthermore, we require the following three properties.

1. *Computational Binding. For all PPT adversaries* A *and* $n \in \mathbb{N}$

$$\Pr\left[b_0 = b_1 \neq 0 \wedge Y_0 \neq Y_1 : \begin{array}{c} \rho \xleftarrow{\$} \mathcal{U}(\lambda),\ pp \xleftarrow{\$} \mathsf{Setup}^{\rho}(1^{\lambda}, 1^{N}) \\ (C, Y_0, Y_1, d_0, d_1) \xleftarrow{\$} A^{\rho}(pp) \\ b_0 \leftarrow \mathsf{Open}^{\rho}(pp, C, Y_0, d_0) \\ b_1 \leftarrow \mathsf{Open}^{\rho}(pp, C, Y_1, d_1) \end{array}\right] \leq \mathsf{negl}(\lambda)\ .$$

2. *Perfect Correctness.* *For all* $n, \lambda \in \mathbb{N}$ *and all* $Y \in \mathbb{F}^N$ *and* $\zeta \in \mathbb{F}^n$,

$$\Pr\left[1 = \mathsf{Eval}^{\rho}(pp, C, Z, \gamma; Y, d) : \begin{array}{c} \rho \xleftarrow{\$} \mathcal{U}(\lambda),\ pp \xleftarrow{\$} \mathsf{Setup}^{\rho}(1^{\lambda}, 1^{N}), \\ (C; d) \xleftarrow{\$} \mathsf{Com}^{\rho}(pp, Y),\ \gamma = \mathsf{ML}(Y, \zeta) \end{array}\right] = 1 \ .$$

3. *Witness-extended Emulation.* *We say that the polynomial commitment scheme has witness-extended emulation if* Eval *has a witness-extended emulation as an interactive argument for the relation ensemble* $\{\mathcal{R}_{\mathsf{mle}}(pp)\}_{pp}$ *(Equation (4.5)) except with negligible probability over the choice of* ρ *and coins of* $pp \xleftarrow{\$} \mathsf{Setup}^{\rho}(1^{\lambda}, 1^{N})$.

4.6.2 Space-Efficient Commitment for Multilinear Polynomials

In this section we describe our polynomial commitment scheme for multilinear polynomials, a high level overview of which was provided in Section 4.3.2. We dedicate the remainder of the section to constructing our polynomial commitment scheme.

Commitment Scheme

We describe our commitment scheme $(\mathsf{Setup}, \mathsf{Com}, \mathsf{Open}, \mathsf{Eval})$ to multilinear polynomials.

1. $\mathsf{Setup}^{\rho}(1^{\lambda}, 1^{N})$: On inputs security parameter 1^{λ} and size parameter $N = 2^n$ and access to ρ, Setup samples $(\mathbb{G}, p, h) \leftarrow \mathcal{G}(1^{\lambda})$, sets $\mathbb{F} = \mathbb{F}_p$ and returns $pp = (\mathbb{G}, \mathbb{F}, N, p)$. Furthermore, it implicitly defines a sequence of generators $g = (g_b = \rho(b) : b \in \{0,1\}^n)$.

2. $\mathsf{Com}^{\rho}(pp, Y)$ returns $C \in \mathbb{G}$ as the commitment and Y as the decommitment where

$$C \leftarrow \prod_{b \in \{0,1\}^n} (g_b)^{y_b} \ .$$

3. $\mathsf{Open}^{\rho}(pp, C, Y)$ returns 1 iff $C = \mathsf{Com}^{\rho}(pp, Y)$.

4. $\mathsf{Eval}^{\rho}(pp, C, \zeta, \gamma; Y)$ is an interactive protocol $\langle P, V \rangle$ that begins with V sending a random $h \xleftarrow{\$} \mathbb{G}$. Then, both P and V compute the commitment $C_\gamma \leftarrow C \cdot h^{\gamma}$ to additionally bind the claimed evaluation γ. Then, P and V engage in an interactive

protocol $\mathsf{EvalReduce}$ on input $(C_\gamma, Z, g, h, \gamma; Y)$ where the prover proves knowledge of Y such that

$$C_\gamma = \mathsf{Com}(g, Y) \cdot h^\gamma \wedge \langle Y, Z \rangle = \gamma \ ,$$

where $Z = (z_b = \bar{\chi}(b, \zeta) : b \in \{0,1\}^n)$. We define the protocol in Figure 4.4.

Remark 4.6.3. *In fact, our scheme readily extends to proving any linear relation $\alpha \in \mathbb{F}^N$ about a committed sequence Y (i.e., the value $\langle \alpha, Y \rangle$), as long as each element of α can be generated in poly-logarithmic time.*

The described commitment scheme is characterized by the following theorem.

Theorem 4.6.4. *Let $\mathcal{G}$ be a generator of obliviously sampleable,*[15] *prime-order groups. Assuming the hardness of discrete logarithm problem for $\mathcal{G}$, the scheme* $(\mathsf{Setup}, \mathsf{Com}, \mathsf{Open}, \mathsf{Eval})$ *defined above is a polynomial commitment scheme to multilinear polynomials with witness-extended emulation in the random oracle model. Furthermore, for every $N \in \mathbb{N}$ and sequence $Y \in \mathbb{F}^N$, the committer/prover has multi-pass streaming access to Y and*

1. Com *performs $O(N \cdot \log(p))$ group operations, stores $O(1)$ field and group elements, requires one pass over Y, makes N queries to the random oracle, and outputs a single group element. Evaluating $\mathsf{ML}(Y, \cdot)$ requires $O(N)$ field operations, storing $O(1)$ field elements and requires one pass over Y.*

2. Eval *is public-coin and has $O(\log(N))$ rounds with $O(1)$ group elements sent in every round. Furthermore,*

   - *Prover performs $O(N \cdot (\log^2(N)) \cdot \log(p))$ field and group operations, $O(N \cdot \log(N))$ queries to the random oracle, requires $O(\log(N))$ passes over Y and stores $O(\log N)$ field and group elements.*
   - *Verifier performs $O(N \cdot (\log(N)) \cdot \log(p))$ field and group operations, $O(N)$ queries to the random oracle, and stores $O(\log(N))$ field and group elements.*

[15] By *obliviously sampleable* we mean that there exist algorithms S and S^{-1} such that on input random coins r, the algorithm S samples a uniformly random group element g, whereas on input g, the algorithm S^{-1} samples random coins r that are consistent with the choice of g. In other words, if S uses ℓ random bits then the joint distributions $(U_\ell, S(U_\ell))$ and $(S^{-1}(S(U_\ell)), S(U_\ell))$ are identically distributed, where U_ℓ denotes the uniform distribution on ℓ bit strings.

Eval $(pp, C, \zeta, \gamma; Y)$

Input : Public parameters pp, $C \in \mathbb{G}$, $\zeta \in \mathbb{F}^n$, $\gamma \in \mathbb{F}$, $Y \in \mathbb{F}^N$

Output : Accept or Reject

V samples and sends $h \xleftarrow{\$} \mathbb{G}$ to P.

P and V define $C_\gamma = C \cdot h^\gamma$.

P and V define $Z = (z_b := \prod_{i=1}^n \chi(b_i, \zeta_i))_{b \in \{0,1\}^n}$.

P and V engage in EvalReduce$(C_\gamma, Z, \gamma, g, h; Y)$.

EvalReduce $(C_\gamma, Z, \gamma, g, h; Y)$

Input : $C_\gamma \in \mathbb{G}, Z \in \mathbb{F}^N, \gamma \in \mathbb{F}, g \in \mathbb{G}^N, h \in \mathbb{G}, Y \in \mathbb{F}^N$

Output : Accept or Reject

Set $N = |Z|$, $n = \log(N)$.

if $N = 1$ **then**

Let $g = (g') \in \mathbb{G}$, $Z = (z) \in \mathbb{F}$, and $Y = (y) \in \mathbb{F}$. P sends y to V.

V accepts if and only if $C_\gamma = (g')^y \cdot h^{y \cdot z}$.

else

P computes γ_0, γ_1 where

$$\gamma_0 = \sum_{b \in \{0,1\}^{n-1}} Y(b \circ 0) \cdot Z(b \circ 1) \qquad \gamma_1 = \sum_{b \in \{0,1\}^{n-1}} Y(b \circ 1) \cdot Z(b \circ 0).$$

P computes and sends C_0, C_1 to V, where

$$C_0 = h^{\gamma_0} \cdot \prod_{b \in \{0,1\}^{n-1}} (g_{b \circ 1})^{Y(b \circ 0)} \qquad C_1 = h^{\gamma_1} \cdot \prod_{b \in \{0,1\}^{n-1}} (g_{b \circ 0})^{Y(b \circ 1)}.$$

V samples $\alpha \xleftarrow{\$} \mathbb{F}$ and sends α to P.

P and V both compute

$$C'_{\gamma'} = (C_0)^{\alpha^2} \cdot C_\gamma \cdot (C_1)^{\alpha^{-2}}$$
$$Z' = (z'_b = \alpha^{-1} \cdot z_{b \circ 0} + \alpha \cdot z_{b \circ 1})_{b \in \{0,1\}^{n-1}}$$
$$g' = ((g_{b \circ 0})^{\alpha^{-1}} \cdot g_{b \circ 1}^{\alpha})_{b \in \{0,1\}^{n-1}}.$$

P computes $Y' = (y'_b = \alpha \cdot y_{b \circ 0} + \alpha^{-1} \cdot y_{b \circ 1})_{b \in \{0,1\}^{n-1}}$.

return EvalReduce$(C'_{\gamma'}, Z', \gamma', g', h; Y')$.

Figure 4.4. Eval protocol for the commitment scheme from Section 4.6.2.

4.6.3 Correctness and Security

Lemma 4.6.5. *The scheme from Section 4.6.2 is perfectly correct, computationally binding and* Eval *has witness-extended emulation under the hardness of the discrete logarithm problem for groups sampled by $\mathcal{G}$ in the random oracle model.*

The perfect correctness of the scheme follows from the correctness of EvalReduce protocol, which we prove in Lemma 4.6.6, computationally binding follows from that of Pedersen multi-commitments which follows from the hardness of discrete-log (in the random oracle model). The witness-extended emulation of Eval follows from the witness-extended emulation of the inner-product protocol in [72]. At a high level, we make two changes to their inner-product protocol: (1) sample the generators using the random oracle ρ, (2) perform the 2-move reduction step using the lsb-based folding approach (see Section 4.3.2 for a discussion). At a high level, given a witness Y for the inner-product statement (C_γ, g, Z, γ), one can compute a witness for the permuted statement $(C_\gamma, \pi(g), \pi(Z), \gamma)$ for any efficiently computable/invertible public permutation π. Choosing π as the permutation that reverses its input allows us, in principle, to base the extractability of our scheme (lsb-based folding) to the original scheme of [72]. Due to (1) our scheme enjoys security only in the random-oracle model.

Lemma 4.6.6. *Let $(C_\gamma, Z, \gamma, g, h; Y)$ be inputs to* EvalReduce *and let $(C'_{\gamma'}, Z', \gamma', g', h; Y')$ be generated as in Figure 4.4. Then,*

$$\begin{array}{ccc} C_\gamma = \mathsf{Com}(g, Y) \cdot h^\gamma & & C'_{\gamma'} = \mathsf{Com}(g', Y') \cdot h^{\gamma'} \\ \wedge & \implies & \wedge \\ \langle Y, Z \rangle = \gamma & & \langle Y', Z' \rangle = \gamma' \end{array} .$$

Proof. Let $N = |Z|$ and let $n = \log N$. Then,

1. To show $\gamma' = \langle Y', Z' \rangle$:

$$\begin{aligned}
\langle Y', Z' \rangle &= \sum_{b \in \{0,1\}^{n-1}} y'_b \cdot z'_b, \\
&= \sum_{b \in \{0,1\}^{n-1}} (\alpha \cdot y_{b\circ 0} + \alpha^{-1} \cdot y_{b\circ 1}) \cdot (\alpha^{-1} \cdot z_{b\circ 0} + \alpha \cdot z_{b\circ 1}), \\
&= \sum_{b \in \{0,1\}^{n-1}} y_{b\circ 0} \cdot z_{b\circ 0} + \alpha^2 \cdot y_{b\circ 0} \cdot z_{b\circ 1} + y_{b\circ 1} \cdot z_{b\circ 1} + \alpha^{-2} \cdot y_{b\circ 1} \cdot z_{b\circ 1}, \\
&= \gamma + \alpha^2 \cdot \gamma_0 + \alpha^{-2} \cdot \gamma_1 = \gamma'.
\end{aligned}$$

2. $C'_{\gamma'} = \mathsf{Com}(g', Y') \cdot g^{\gamma'}$:

$$\begin{aligned}
\mathsf{Com}(g', Y') &= \prod_{b \in \{0,1\}^{n-1}} (g'_b)^{y'_b}, = \prod_{b \in \{0,1\}^{n-1}} \left(g_{b\circ 0}^{\alpha^{-1}} \cdot g_{b\circ 1}^{\alpha} \right)^{\alpha \cdot y_{b\circ 0} + \alpha^{-1} \cdot y_{b\circ 1}}, \\
&= \prod_{b \in \{0,1\}^{n-1}} \left(g_{b\circ 0}^{y_{b\circ 0}} \cdot g_{b\circ 0}^{\alpha^{-2} \cdot y_{b\circ 1}} \cdot g_{b\circ 1}^{\alpha^2 \cdot y_{b\circ 0}} \cdot g_{b\circ 1}^{y_{b\circ 1}} \right), \\
&= \prod_{b \in \{0,1\}^{n-1}} (g_{b\circ 0}^{y_{b\circ 1}})^{\alpha^{-2}} \cdot g_{b\circ 0}^{y_{b\circ 0}} \cdot g_{b\circ 1}^{y_{b\circ 1}} \cdot (g_{b\circ 1}^{y_{b\circ 0}})^{\alpha^2}.
\end{aligned}$$

Then, above with the definition of γ' implies that $C'_{\gamma'} = \mathsf{Com}(g', Y') \cdot h^{\gamma'}$. □

4.6.4 Efficiency

In this section we discuss the efficiency aspects of each of the protocols defined in Section 4.6.2 with respect to four complexity measures: (1) queries to the random oracle ρ, (2) field/group operations performed, (3) field/group elements stored and (4) number of passes over the stream Y.

For the rest of this section, we fix $n, N = 2^n, \rho, \mathbb{G}, \mathbb{F}, \zeta \in \mathbb{F}^n$ and furthermore fix $Y = (y_b : b \in \{0,1\}^n)$, $g = (g_b = \rho(b) : b \in \{0,1\}^n)$ and $Z = (z_b = \bar{\chi}(b, \zeta) : b \in \{0,1\}^n)$. Note given ζ, any z_b can be computed by performing $O(n)$ field operations.

First, consider the prover P of Eval protocol (Figure 4.4). Given the inputs $(C, Z, \gamma, g, h; Y)$, P and V call the recursive protocol $\mathsf{EvalReduce}$ on the N sized statement $(C_\gamma, Z, \gamma, g, h; Y)$ where $C_\gamma = C \cdot h^\gamma$. The prover's computation in this call to $\mathsf{EvalReduce}$ is dictated by

computing (a) γ_0, γ_1 (line 6), (2) C_0, C_1 (line 7) and (c) inputs for the next recursive call on $\mathsf{EvalReduce}$ with $N/2$ sized statement $(C'_{\gamma'}, Z', \gamma', g', h; Y')$ (line 9,11). The rest of its computation requires $O(1)$ number of operations. The recursion ends on the n-th call with statement of size 1. For $k \in \{0, \ldots, n\}$, the inputs at the k-th depth of the recursion be denoted with superscript k, that is, $C^{(k)}, \gamma^{(k)}, Z^{(k)}, g^{(k)}, Y^{(k)}$. For example, $Z^{(0)} = Z$, $Y^{(0)} = Y$ denote the initial inputs (at depth 0) where prover computes $\gamma_0^{(0)}, \gamma_1^{(0)}, C_0^{(0)}, C_1^{(0)}$ with verifier challenge $\alpha^{(0)}$. The sequences $Z^{(k)}, Y^{(k)}$ and $g^{(k)}$ are of size 2^{n-k}.

At a high level, we ask prover to never explicitly compute the sequences $g^{(k)}, Z^{(k)}, Y^{(k)}$ (item (c) above) but instead compute elements $g_b^{(k)}, z_b^{(k)}, y_b^{(k)}$, of the respective sequences, on demand, which then can be used to compute $\gamma_0^{(k)}, \gamma_1^{(k)}, C_0^{(k)}, C_1^{(k)}$ in required time and space. For this, first it will be useful to see how the elements of sequences $Z^{(k)}, Y^{(k)}, g^{(k)}$ depend on the initial (i.e., depth-0) sequence $Z^{(0)}, Y^{(0)}, g^{(0)}$.

Relating $Y^{(k)}$ with $Y^{(0)}$.

First, we consider $Y^{(k)} = (y_b^{(k)} : b \in \{0,1\}^{n-k})$ at depth $k \in \{0, \ldots, n\}$. Let $\alpha^{(i)}$ denote the verifier's challenge sent during a prior round $i \in \{0, 1, \ldots, k-1\}$.

Lemma 4.6.7 (Streaming of $Y^{(k)}$)**.** *For every $b \in \{0,1\}^{n-k}$,*

$$y_b^{(k)} = \sum_{c \in \{0,1\}^k} \left(\prod_{j=1}^{k} \mathsf{coeff}(\alpha^{(j-1)}, c_j) \right) \cdot y_{b \circ c} \ , \tag{4.6}$$

where $\mathsf{coeff}(\alpha, c) = \alpha \cdot (1-c) + \alpha^{-1} \cdot c$.

The proof follows by induction on depth k. Lemma 4.6.7 allows us to simulate the stream $Y^{(k)}$ with **one** pass over the initial sequence Y, additionally performing $O(N \cdot k)$ multiplications to compute appropriate coeff functions.

Relating $Z^{(k)}$ with $Z^{(0)}$.

Next, consider $Z^{(k)} = (z_b^{(k)} : b \in \{0,1\}^{n-k})$ at depth $k \in \{0, \ldots, n\}$.

$\mathsf{Computez}(k, c, \zeta, \alpha)$

1. $z_c^{(k)} = 0 \in \mathbb{F}$
2. for each $a \in \{0,1\}^k$:
   (a) $\mathsf{temp} = 1 \in \mathbb{F}$
   (b) for each $j \in [k]$:
      i. $\mathsf{temp} = \mathsf{temp} \cdot \mathsf{coeff}(\alpha^{(j-1)}, a_j)$
   (c) $z_c^{(k)} = z_c^{(k)} + \mathsf{temp} \cdot \bar{\chi}(c \circ a, \zeta)$
3. **return** $z_c^{(k)}$

$\mathsf{Computeg}^{\rho}(k, c, \alpha)$

1. $g_c^{(k)} = 1 \in \mathbb{G}$
2. for each $a \in \{0,1\}^k$:
   (a) $\mathsf{temp} = 1 \in \mathbb{F}$
   (b) for each $j \in [k]$:
      i. $\mathsf{temp} = \mathsf{temp} \cdot \mathsf{coeff}(\alpha^{(j-1)}, a)$
   $g_c^{(k)} = g_c^{(k)} \cdot \rho(c \circ a)^{\mathsf{temp}}$
3. **return** $g_c^{(k)}$

Figure 4.5. Algorithms for computing $z_b^{(k)}$ and $g_b^{(k)}$. In both algorithms $c \in \{0,1\}^{n-k}$ and $\alpha = (\alpha^{(0)}, \ldots, \alpha^{(k-1)})$, where $\bar{\chi}(b, \zeta) = \prod_{i=1}^{n} \chi(b_i, \zeta_i)$ for $b = c \circ a$, and $\mathsf{coeff}(\alpha, c) = \alpha \cdot c + \alpha^{-1} \cdot (1 - c)$.

Lemma 4.6.8 (Computing $z_b^{(k)}$)**.** *For every $b \in \{0,1\}^{n-k}$,*

$$z_b^{(k)} = \sum_{c \in \{0,1\}^k} \left(\prod_{j=1}^{k} \mathsf{coeff}(\alpha^{(j-1)}, c_j) \right) \cdot z_{b \circ c} \ , \tag{4.7}$$

where $\mathsf{coeff}(\alpha, c) = \alpha \cdot c + \alpha^{-1} \cdot (1 - c)$. Furthermore, computing $z_b^{(k)}$ requires $O(2^k \cdot n)$ field multiplications and storing $O(n)$ elements (see algorithm Computez *in Figure 4.5).*

Relating $g^{(k)}$ with $g^{(0)}$.

Finally, consider $g^{(k)} = (g_b^{(k)} : b \in \{0,1\}^{n-k})$ at depth $k \in \{0, \ldots, n\}$.

Lemma 4.6.9 (Computing $g_b^{(k)}$)**.** *For every $b \in \{0,1\}^{n-k}$,*

$$g_b^{(k)} = \prod_{c \in \{0,1\}^k} g_{b \circ c}^{\mathsf{coeff}(\alpha, \boldsymbol{c})} \ ; \ \mathsf{coeff}(\alpha, \boldsymbol{c}) = \prod_{i=1}^{k} \alpha^{(j-1)} \cdot c_j + (\alpha^{(j-1)})^{-1} \cdot (1 - c_j) \ . \tag{4.8}$$

Furthermore, computing $g_b^{(k)}$ requires $2^k \cdot k$ field multiplications, 2^k queries to ρ, 2^k group multiplications and exponentiations, and storing $O(k)$ elements (see algorithm Computeg *in Figure 4.5).*

We now discuss the efficiency of the commitment scheme.

Commitment Phase

We first note that Com^{ρ} on input pp and given streaming access to Y can compute the commitment $C = \prod_b (\rho(b))^{y_b}$ for $b \in \{0,1\}^n$ making N queries to ρ, performing N group exponentiations and a single pass over Y. Furthermore, requires storing only a single group element. Note that a single group exponentiation h^{α} can be emulated while performing $O(\log p)$ group multiplications while storing $O(1)$ group and field elements. Since, $\mathbb{G}, \mathbb{F}$ are of order p, field and group operations can, furthermore, be performed in polylog$(p(\lambda))$ time.

Evaluating $\mathsf{ML}(Y, \zeta)$

The honest prover (when used in higher level protocols) needs to evaluate $\mathsf{ML}(Y, \zeta)$ which requires performing $O(N \log N)$ field operations overall and a single pass over stream Y.

Prover Efficiency

For every depth-k of the recursion, it is sufficient to discuss the efficiency of computing $\gamma_0^{(k)}, \gamma_1^{(k)}, C_0^{(k)}$, and $C_1^{(k)}$. We argue the complexity of computing $\gamma_0^{(k)}$ and $C_0^{(k)}$ and the analysis for the remaining is similar. We give a formal algorithm Prover in Figure 4.6.

Computing $\gamma_0^{(k)}$

Recall that $\gamma_0^{(k)} = \sum_b y_{b\circ 0}^{(k)} \cdot z_{b\circ 1}^{(k)}$ for $b \in \{0,1\}^{n-k-1}$. To compute $\gamma_0^{(k)}$ we stream the initial N-sized sequence Y and generate elements of the sequence $(y_{b\circ 0}^{(k)} : b \in \{0,1\}^{n-k-1})$ in a streaming manner. Since each $y_{b\circ 0}^{(k)}$ depends on a contiguous block of 2^k elements in the initial stream Y, we can compute $y_{b\circ 0}^{(k)}$ by performing $2^k \cdot k$ field operations (lines 2-7 in Figure 4.6). For every $b \in \{0,1\}^{n-k-1}$, after computing $y_{b\circ 0}^{(k)}$, we leverage "random access" to Z and compute $z_{b\circ 1}^{(k)}$ (Lemma 4.6.8) which requires $O(2^k \cdot k)$ field operations. Overall, $\gamma_0^{(k)}$ can be computed in $O(N \cdot k)$ field operations and a single pass over Y.

Prover$^{\rho}$ $(pp, k, Y, \zeta, g, \alpha^{(0)}, \ldots, \alpha^{(k-1)})$

Input : Public parameters pp, integer $k \in \{0, 1, \ldots, n\}$, $Y \in \mathbb{F}^N$, $\zeta \in \mathbb{F}^n$, $\alpha^{(i)} \in \mathbb{F}$ for $i \in \{0, \ldots, k-1\}$.

Output : Values $\gamma_0, \gamma_1 \in \mathbb{F}$ and $C_0, C_1 \in \mathbb{G}$.

Set $\gamma_0 = \gamma_1 = y^{(k)} = 0 \in \mathbb{F}$, $g^{(k)} = C_0 = C_1 = 1 \in \mathbb{G}$, $count = 0$.
foreach $b = (b_n, \ldots, b_1) \in \{0,1\}^n$ **do**
    $\mathsf{temp} = 1 \in \mathbb{F}$
    foreach $j \in [k]$ **do**
        $\mathsf{temp} = \mathsf{temp} \cdot \mathsf{coeff}(\alpha^{(j)}, b_j)$
    $y^{(k)} = y^{(k)} + \mathsf{temp} \cdot y_b$
    $count = count + 1$
    if $count = 2^k$ **then**
        $z^{(k)} = \mathsf{Computez}(k, (b_n, \ldots, b_{n-k+1}, 1 - b_{n-k}), \zeta, \alpha^{(0)}, \ldots, \alpha^{(k-1)})$
        $g^{(k)} = \mathsf{Computeg}^{\rho}(k, (b_n, \ldots, b_{n-k+1}, 1 - b_{n-k}), \alpha^{(0)}, \ldots, \alpha^{(k-1)})$
        if $b_{n-k} = 0$ **then**
            $\gamma_0 = \gamma_0 + z^{(k)} \cdot y^{(k)}$
            $C_0 = C_0 \cdot (g^{(k)})^{y^{(k)}}$
        else
            $\gamma_1 = \gamma_1 + z^{(k)} \cdot y^{(k)}$
            $C_1 = C_1 \cdot (g^{(k)})^{y^{(k)}}$
        $y^{(k)} = 0$; $g^{(k)} = 1$; $count = 0$
$C_0 = C_0 \cdot h^{\gamma_0}$; $C_1 = C_1 \cdot h^{\gamma_1}$
return $(\gamma_0, \gamma_1, C_0, C_1)$

Figure 4.6. Space-Efficient Prover implementation.

Computing $C_0^{(k)}$

The two differences in computing $C_0^{(k)}$ (see Figure 4.4 for the definition) is that (a) we need to compute $g_{b\circ 1}^{(k)}$ instead of computing $z_{b\circ 1}^{(k)}$ and (b) perform group exponentiations, that is, $g_{b\circ 1}^{(k)\, y_{b\circ 0}^{(k)}}$ as opposed to group multiplications as in the computation of $\gamma_0^{(k)}$. Both steps overall can be implemented in $O(N \cdot k \cdot \log p)$ field and group operations and N queries to ρ (Lemma 4.6.9). Overall, at depth k the prover (1) makes $O(N)$ queries to ρ, (2) performs $O(N \cdot k \cdot \log(p))$ field and group operations and (3) requires a single pass over Y.

Therefore, the entire prover computation (over all calls to $\mathsf{EvalReduce}$) requires $O(\log N)$ passes over Y, makes $O(N \log N)$ queries to ρ and performs $O(N \cdot \log^2 N \cdot \log p)$ field/group operations. Furthermore, this requires storing only $O(\log N)$ field and group elements.

Verifier Efficiency

V only needs to compute folded sequence $Z^{(n)}$ and folded generators $g^{(n)}$ at depth-n of the recursion. These can computed by invoking Computez and Computeg (Figure 4.5) with $k = n$ and require $O(N \cdot \log(N, p))$ field and group operations, $O(N)$ queries to ρ and storing $O(\log N)$ field and group elements.

Lemma 4.6.10. *The time and space efficiency of each of the phases of the protocols are listed below:*

Computation	*ρ queries*	*Y passes*	$\mathbb{F}/\mathbb{G}$ *ops*[16]	$\mathbb{G}/\mathbb{F}$ *elements*
Com	N	1	$O(N)$	$O(1)$
$\mathsf{ML}(Y, \zeta)$	0	1	$O(N \log N)$	$O(1)$
P *(in* Eval*)*	$O(N \log N)$	$O(\log N)$	$O(N \log^2 N)$	$O(\log N)$
V *(in* Eval*)*	$O(N)$	0	$O(N \log N)$	$O(\log N)$

Finally, Theorem 4.6.4 follows directly from Lemma 4.6.5 and Lemma 4.6.10.

4.7 Streamable Polynomial Commitment Scheme for Multilinear Polynomials from Groups of Unknown Order

In this section, we give our polynomial commitment scheme which realizes Theorem 4.2.4. Section 4.7.1 gives relevant preliminaries for this section.

4.7.1 Preliminaries

Notation

We let $\mathbb{F}_p$ denote a finite field of prime order p, and often use lower-case Greek letters to denote elements of $\mathbb{F}$, e.g., $\alpha \in \mathbb{F}$. For bit strings $b \in \{0,1\}^n$, we naturally associate b with integers in the set $\{0, 1, \ldots, 2^n - 1\}$; i.e., $b \equiv \sum_{i=1}^{n} b_i \cdot 2^{i-1}$. We assume that $b = (b_n, \ldots, b_1)$, where b_n is the most significant bit and b_1 is the least significant bit. For bit string $b \in \{0,1\}^n$ and $\sigma \in \{0,1\}$ we let σb (resp., $b\sigma$) denote the string $(\sigma \circ b) \in \{0,1\}^{n+1}$

[16] $\uparrow \log(p)$ factors are omitted.

(resp., $(b \circ \sigma) \in \{0,1\}^{n+1}$). For $(\alpha_n, \ldots, \alpha_1) = \alpha \in \mathbb{F}^n$, we refer to α_n as the most significant field element and α_1 as the least significant field element. For two equal length vectors u, v, we let $u \odot v$ denote the coordinate-wise product of u and v. We let uppercase calligraphic letters denote sequences and let corresponding lowercase letters to denote its elements, e.g., $\mathcal{Y} = (y_b)_{b \in \{0,1\}^n} \in \mathbb{F}^N$ is a sequence of N elements in $\mathbb{F}$. Often, for $b \in \{0,1\}^n$, we let $\mathcal{Y}_b$ denote the value y_b.

We use upper case letters to denote matrices, e.g., $M \in \mathbb{Z}^{m \times n}$. For a matrix M of dimension $m \times n$, we let $M(i, *)$ and $M(*, j)$ denote the i^{th} row and j^{th} column of M, respectively. For row vector u of length m and column vector v of length n, we let $u \cdot M$ and $M \cdot v$ denote the standard matrix-vector product.

Non-standard Notation

We are also interested in matrix-vector "exponents". Let $\mathbb{G}$ be some group, $M \in \mathbb{Z}^{m \times n}$, $u = (u_1, \ldots, u_m) \in \mathbb{G}^{1 \times m}$, and $v = (v_1, \ldots, v_n)^\top \in \mathbb{G}^{n \times 1}$. We let $u \star M$ and $M \star v$ denote a matrix-vector exponent, defined as

$$(u \star M)_j = \prod_{i=1}^{m} u_i^{M(i,j)} \qquad\qquad (M \star v)_{i'} = \prod_{j'=1}^{n} v_{j'}^{M(i',j')},$$

for every $j \in [n]$ and every $i' \in [m]$. Note that $u \star M \in \mathbb{G}^{1 \times n}$ and $M \star v \in \mathbb{G}^{m \times 1}$.

For vector $x \in \mathbb{Z}^n$ and group element $g \in \mathbb{G}$, we abuse notation and let $g^x := (g^{x_1}, \ldots, g^{x_n})$. Finally, for $k \in \mathbb{Z}$ and vector $u \in \mathbb{G}^n$, we let u^k denote the vector $(u_1^k, \ldots, u_n^k) \in \mathbb{G}^n$.

Interactive Games and Proof Systems

Definition 4.7.1 (Merlin-Arthur Games)**.** *Let $r \in \mathbb{Z}^+$. A* $\mathsf{MA}[2r]$ game *(or just a* MA *game*[17] *if r is unspecified) is a tuple $G = (1^r, 1^\ell, W)$, where $\ell \in \mathbb{Z}^+$ and $W \subseteq \{0,1\}^*$ is a set, called the* win predicate*, that is represented as a Boolean circuit. The integer r is called the* number of rounds *of G and $\{0,1\}^\ell$ is called the* challenge space*.*

[17] ↑MA stands for Merlin-Arthur proofs [23], differing from Arthur-Merlin proofs in that the prover (Merlin) sends the first message.

If $G = (1^r, 1^\ell, W)$ *is a* $\mathsf{MA}[2r]$ *game and* $P\colon \{0,1\}^* \to \{0,1\}^*$ *is a function, then the* value of G with respect to P *is denoted and defined as*

$$v[P](G) := \Pr\left[(\alpha_1, \beta_1, \ldots, \alpha_r, \beta_r) \in W\right],$$

where the probability is taken over $\beta_1, \ldots, \beta_r \xleftarrow{\$} \{0,1\}^\ell$ *and* $\alpha_i := P(\beta_1, \ldots, \beta_{i-1})$*. The* value of G*, denoted by* $v(G)$*, is defined as* $\sup_P\{v[P](G)\}$*.*

Definition 4.7.2 (Game Transcripts)**.** *If* $G = (1^r, 1^\ell, W)$ *is a* $\mathsf{MA}[2r]$ *game, then a* transcript for G *is a tuple* $\tau = (\alpha_1, \beta_1, \ldots, \alpha_r, \beta_r)$ *with each* $\beta_i \in \{0,1\}^\ell$ *and* $\alpha_i \in \{0,1\}^*$*. If* $\tau \in W$ *then we say it is an* accepting transcript for G*. If for function* $P\colon \{0,1\}^* \to \{0,1\}^*$ *we have* $\alpha_i = P(\beta_1, \ldots, \beta_{i-1})$ *for every* $i \in [r]$*, then* τ *is said to be* consistent with P*. If* τ *is both an accepting transcript for* G *and consistent with* P*, we say that* τ *is an* accepting transcript for (P, G)*.*

Definition 4.7.3 (MA Verifiers)**.** *For a function* $r\colon \mathbb{Z}^+ \to \mathbb{Z}^+$ *and a language* $\mathcal{L}$*, a* $\mathsf{MA}[2r]$ verifier for $\mathcal{L}$ *is a polynomial-time algorithm* V *such that*

1. V *maps any string* $x \in \{0,1\}^*$ *to a* $\mathsf{MA}[2r(|x|)]$ *game.*[18]*.*
2. *The* completeness *of* V *is a function* $c\colon \mathbb{Z}^+ \to [0,1]$*, defined as*

$$c(n) := \min_{x \in \mathcal{L} \cap \{0,1\}^n} v(V(x)).$$

3. *The* soundness error of V *is a function* $s\colon \mathbb{Z}^+ \to [0,1]$*, defined as*

$$s(n) := \max_{x \in \{0,1\}^n \setminus \mathcal{L}} v(V(x)).$$

Definition 4.7.4 (Witness-Extended Emulation (cf., [138, 186]))**.** *A* MA *verifier* V *has* (statistical) witness-extended $\varepsilon(\cdot)$-emulation *with respect to relation* $\mathcal{R}$ *if there exists an expected polynomial-time oracle algorithm* $\mathcal{E}$ *such that for all* $P\colon \{0,1\}^* \to \{0,1\}^*$ *and all* $x \in \{0,1\}^*$*, for sample* $(\tau, w) \leftarrow \mathcal{E}^P(x)$ *we have*

[18] ↑This definition implies that there is a polynomial in n that bounds the length of any accepting transcript for $V(x)$ when $x \in \{0,1\}^n$.

1. *τ is distributed uniformly at random on the set of all possible transcripts between $V(x)$ and P; and*

2. *with all but $\varepsilon(|x|)$ probability, if τ is an accepting transcript for $V(x)$ then $(x, w) \in \mathcal{R}$.*

$$\Pr\left[\tau \text{ is accepting } \wedge \ (x, w) \notin \mathcal{R}\right] \leqslant \varepsilon.$$

A MA *verifier V has* statistical witness-extended emulation *with respect to relation $\mathcal{R}$ if it has statistical witness-extended ε-emulation for some negligible function ε.*

Multilinear Polynomial Commitment Scheme

Definition 4.7.5 (Multilinear Polynomial Commitment Scheme)**.** *A* multilinear polynomial commitment scheme *is a tuple of protocols* $(\mathsf{Setup}, \mathsf{Com}, \mathsf{Open}, \mathsf{Eval})$ *such that*

1. *$pp \leftarrow \mathsf{Setup}(1^\lambda, p, 1^n)$: takes as input the security parameter $\lambda \in \mathbb{N}$ and outputs public parameter pp that allows to support n-variate multilinear polynomials over $\mathbb{F} = \mathbb{F}_p$ for some prime p.*

2. *$(C; d) \leftarrow \mathsf{Com}(pp, \mathcal{Y})$: takes as input public parameters pp and a description of a multilinear polynomial $\mathcal{Y} = (y_b)_{b \in \{0,1\}^n}$ and outputs a commitment C and a (secret) decommitment d.*

3. *$b \leftarrow \mathsf{Open}(pp, C, \mathcal{Y}, d)$: takes as input pp, a commitment C, a description of the multilinear polynomial $\mathcal{Y}$ and a decommitment d, and returns a decision bit $b \in \{0, 1\}$.*

4. *$\mathsf{Eval}(pp, C, \zeta, \gamma; \mathcal{Y}, d)$: is a public-coin interactive proof system (P, V) for the relation:*

$$\mathcal{R}_{\mathsf{ml}} = \left\{(pp, C, \zeta, \gamma; \mathcal{Y}, d) : \mathsf{Open}(pp, C, \mathcal{Y}, d) = 1 \wedge \gamma = \mathsf{ML}(\mathcal{Y}, \zeta)\right\}, \tag{4.9}$$

where V is an MA *verifier (as per Definition 4.7.3) where P is the honest strategy for V.*

Note that the verifier in this proof-system gets as input the public parameters pp, commitment C, evaluation point $\zeta \in \mathbb{F}^n$ and claimed evaluation $\gamma \in \mathbb{F}$, and the prover additionally receives the full description of the polynomial $\mathcal{Y}$ and the decommitment d.

We require the following three properties from the scheme $(\mathsf{Setup}, \mathsf{Com}, \mathsf{Open}, \mathsf{Eval})$*:*

1. ***Perfect Correctness:*** *for all primes p, $\lambda \in \mathbb{N}$, $n \in \mathbb{N}$ and all $\mathcal{Y} \in \mathbb{F}_p^{\leqslant 2^n}$ and $\zeta \in \mathbb{F}_p^n$,*

$$\Pr\left[1 = \mathsf{Eval}(pp, C, \mathcal{Z}, \gamma; \mathcal{Y}, d) : \begin{array}{c} pp \leftarrow \mathsf{Setup}(1^\lambda, p, 1^n), \\ (C; d) \leftarrow \mathsf{Com}(pp, \mathcal{Y}),\ \ \gamma = \mathsf{ML}(\mathcal{Y}, \zeta) \end{array}\right] = 1 \ .$$

2. ***Computational Binding:*** *for every polynomial-sized family of circuits $A = \{A_\lambda\}_{\lambda \in \mathbb{N}}$ the following holds*

$$\Pr\left[\Big(b_0 = 1\Big) \wedge \Big(b_1 = 1\Big) \wedge \Big(\mathcal{Y}_0 \neq \mathcal{Y}_1\Big) : \begin{array}{c} pp \leftarrow \mathsf{Setup}(1^\lambda, p, 1^N) \\ (C, \mathcal{Y}_0, \mathcal{Y}_1, d_0, d_1) \leftarrow A_\lambda(pp) \\ b_0 \leftarrow \mathsf{Open}(pp, C, \mathcal{Y}_0, d_0) \\ b_1 \leftarrow \mathsf{Open}(pp, C, \mathcal{Y}_1, d_1) \end{array}\right] \leq \mathsf{negl}(\lambda) \ .$$

3. ***Witness-Extended Emulation:*** *For $\mathsf{Eval} = (P, V)$, V has (statistical) witness-extended emulation for the relation $\mathcal{R}_{\mathsf{ml}}$ (defined in Equation (4.9)).*

Remark 4.7.1. *We note that this definition of polynomial commitment scheme is stronger than the ones used in the literature (see, e.g., [30, 36, 51, 76, 166, 184, 232, 255, 266]), in that we require* Eval *to have* statistical *soundness (rather than computational). As a result we show soundness for every pair (x, pp).*

A key ingredient in our efficient argument-systems is polynomial commitments that can be generated in a time *and space* efficient way. We call such polynomial commitments *streamable*.

Definition 4.7.6 (Streamable Multilinear Polynomial Commitment Scheme)**.** *A* streamable multilinear polynomial commitment scheme *is a multilinear polynomial commitment scheme (as per Definition 4.7.5) with the following efficiency properties for n-variate multilinear polynomials over $\mathbb{F}_p$ for some prime $p \leqslant 2^\lambda$:*

1. *The commitment output by* Com *is of size* $n \cdot \mathrm{poly}(\lambda)$*, and assuming multi-pass streaming access to the description of the polynomial, the commitment can be implemented in time* $2^n \cdot \mathrm{poly}(n, \lambda)$ *and space* $\mathrm{poly}(n, \lambda)$*.*

2. *The communication complexity of the* Eval *protocol is* $n \cdot \mathrm{poly}(\lambda)$ *and the receiver of* Eval *runs in time* $\mathrm{poly}(n, \lambda)$*. Assuming multi-pass streaming access to the description of the polynomial, the committer of* Eval *can be implemented in time* $2^n \cdot \mathrm{poly}(n, \lambda)$ *and space* $\mathrm{poly}(n, \lambda)$*.*

4.7.2 Multilinear Polynomial Commitment Scheme in Hidden Order Groups

We describe our commitment scheme $(\mathsf{Setup}, \mathsf{Com}, \mathsf{Open}, \mathsf{Eval})$ for multilinear polynomials $f \colon \mathbb{F}^n \to \mathbb{F}$ over some field $\mathbb{F}$ of prime-order p which is specified as an input to Setup. Throughout the section, we work with the description $\mathcal{Y} := (f(b))_{b \in \{0,1\}^n} \in \mathbb{F}^{2^n}$ of the multilinear polynomial f.

We first describe how to encode $\mathcal{Y}$ as an integer. Then we describe our polynomial commitment scheme.

Encoding Multilinear Polynomials as an Integer

One key portion of our polynomial commitment scheme is encoding the sequence $\mathcal{Y}$, which defines our multilinear polynomial, as an integer. We do so by using a technique first introduced by [76]. Towards this, we first describe an encoding scheme for *integer* sequences. For any $N = 2^n$ and an odd integer $q \in \mathbb{N}$, let $\mathsf{Enc}_q \colon \mathbb{Z}^N \to \mathbb{Z}$ be the function that encodes a sequence of integers $\mathcal{Z} \in \mathbb{Z}^N$ as[19]

$$\mathsf{Enc}_q(\mathcal{Z}) := \sum_{\boldsymbol{b} \in \{0,1\}^n} q^{\boldsymbol{b}} \cdot \mathcal{Z}_b,$$

where $q^{\boldsymbol{b}}$ interprets $\boldsymbol{b}$ (an n-bit string) as the naturally corresponding integer in the set $\{0, 1, \ldots, N-1\}$. To decode an integer $v \in \mathbb{Z}$, we output its base-q representation where, for

[19] ↑This encoding is valid for sequences of arbitrary length, but we restrict to powers of two for convenience.

convenience, the base-q digits of v are integers in the range $[-q/2, q/2)$. We refer to the decoding function as Dec_q.

Our Enc_q scheme has two homomorphic properties which we leverage to design our polynomial commitment. First, $\mathsf{Enc}_q(\cdot)$ is a linear homomorphism over $\mathbb{Z}$; that is, for any $\mathcal{Z}, \mathcal{Z}' \in \mathbb{Z}^N$ and $\alpha, \beta \in \mathbb{Z}$, it holds that $\alpha \cdot \mathsf{Enc}_q(\mathcal{Z}) + \beta \cdot \mathsf{Enc}_q(\mathcal{Z}') = \mathsf{Enc}_q(\alpha \cdot \mathcal{Z} + \beta \cdot \mathcal{Z}')$. Second, $\mathsf{Enc}_q(\cdot)$ satisfies a restricted form of multiplicative homomorphism; that is, for any $d \in \mathbb{N}$, we have $q^d \cdot \mathsf{Enc}_q(\mathcal{Z}) = \mathsf{Enc}_q((0^d, \mathcal{Z}))$.

Encoding Bounded Integer Sequences

In fact, looking ahead, we are interested in encoding only sequences of bounded integers. For some $B \in \mathbb{R}_{\geqslant 1}$, we let $\mathbb{Z}(B) := \{z \in \mathbb{Z} \colon -B \leqslant z < B\}$ be the set of integers whose absolute value is bounded by B. Then, to encode integer sequences in $\mathbb{Z}(B)^N$, we consider the restriction of Enc_q to the set $\mathbb{Z}(B)^N$. Notice that by definition, for any $\mathcal{Z} \in \mathbb{Z}(B)^N$, we have that $\mathsf{Enc}_q(\mathcal{Z}) \in \mathbb{Z}(B \cdot (q^N - 1)/(q-1))$. We remark that while Enc_q is not injective over all integer sequences (as integer sequences $(1+q, 0)$ and $(1, 1)$ both encode to the integer $1+q$), the restriction of Enc_q to the set $\mathbb{Z}(q/2)^N$ *is* injective. We capture this in the following fact:

Fact 4.7.2 ([76, Fact 1])**.** *Let q be any odd integer and let $N \in \mathbb{N}$. For any $v \in \mathbb{Z}(q^N/2)$, there exists a unique sequence $\mathcal{Z} \in \mathbb{Z}(q/2)^N$ such that $v = \mathsf{Enc}_q(\mathcal{Z})$. Furthermore, $\mathcal{Z} = \mathsf{Dec}_q(v)$.*

Proof. For any sequence $\mathcal{Z} \in \mathbb{Z}(q/2)^N$, by definition of Dec_q we observe that $\mathsf{Dec}_q(\mathsf{Enc}_q(\mathcal{Z})) = \mathcal{Z}$. This implies that (restriction of) Enc_q (to $\mathbb{Z}(q/2)^N$) is injective. Furthermore, the cardinality of sets $\mathbb{Z}(q^N/2)$ and $\mathbb{Z}(q/2)^N$ are equal. Therefore, for every $v \in \mathbb{Z}(q^N/2)$, $\mathsf{Dec}_q(v)$ is the unique sequence in $\mathbb{Z}(q/2)^N$ that encodes to v. □

Similar to Enc_q, the function Dec_q also satisfies some homomorphic properties: for integers z_1, z_2, we have that $\mathsf{Dec}_q(z_1 + z_2) = \mathsf{Dec}_q(z_1) + \mathsf{Dec}_q(z_2)$ as long as z_1, z_2 encode sequences whose elements are bounded by $q/4$. For our security proof, it will be more convenient to use the following more general statement.

Claim 4.7.3. *Let $\ell, q, N \in \mathbb{N}$ such that q is odd, and let $B_1, B_2 \geq 1$ be such that $B_1 \cdot B_2 \leq q/(2\ell)$. Then, for every $\alpha_1, \ldots, \alpha_\ell \in \mathbb{Z}(B_1)$, and integers $z_1, \ldots, z_\ell \in \mathbb{Z}(q^N/2)$ such that $\mathsf{Dec}_q(z_i) \in \mathbb{Z}(B_2)^N$,*

$$\mathsf{Dec}_q\Big(\sum_{i\in[\ell]} \alpha_i \cdot z_i\Big) = \sum_{i\in[\ell]} \alpha_i \cdot \mathsf{Dec}_q(z_i). \tag{4.10}$$

Remark 4.7.4. *Looking ahead, the correctness of our extractor (to show security for our polynomial commitment scheme) relies crucially on Claim 4.7.3.The main issue with [76]'s extractor is that their extractor relies on a variant of Claim 4.7.3 (formulated below) which is false. Lemma 8 in the full version [75] of [76] uses the following claim to argue correctness of the extracted integer decommitments f_L and f_R.*

Claim 4.7.5 (False claim implicit in [75, Lemma 8])**.** *For $p, q, N \in \mathbb{N}$ such that $2 \leq p \leq q$ where q is odd. For every $\alpha \in \mathbb{Z}(p)$ and $z \in \mathbb{Z}(q^N/2)$ such that $\alpha \mid z$,*

$$\mathsf{Dec}_q(z/\alpha) = \mathsf{Dec}_q(z)/\alpha\ . \tag{4.11}$$

We note that $z, z/\alpha \in \mathbb{Z}(q^N/2)$, by Fact 4.7.2 $\mathsf{Dec}_q(z), \mathsf{Dec}_q(z/\alpha) \in \mathbb{Z}(q/2)^N$. But, $\mathsf{Dec}_q(z)/\alpha$ may not be an integer sequence. Counter-example: for $z = 1 + q, \alpha = 2$, we have $\mathsf{Dec}_q(z) = (1, 1)$ but $\mathsf{Dec}_q(z)/2$ is not an integer sequence even though $\alpha \mid z$.

Encoding $\mathcal{Y}$

Given the integer encoding function Enc_q, we now describe how to encode the sequence of evaluations $\mathcal{Y} \in \mathbb{F}^N$. Recall that $\mathbb{F}$ is a field of prime-order p. To encode $\mathcal{Y}$, we first define a lifting function $[\![\cdot]\!] \colon \mathbb{F} \to \mathbb{Z}(p/2)$ in the natural way. That is, for any $\alpha \in \mathbb{F}$, we define $[\![\alpha]\!]$ to be the unique integer in $\mathbb{Z}(p/2)$ such that $[\![\alpha]\!] \equiv \alpha \bmod p$. We then define $\mathsf{Enc}_q(\mathcal{Y})$ as

$$\mathsf{Enc}_q(\mathcal{Y}) := \sum_{\boldsymbol{b}\in\{0,1\}^n} q^{\boldsymbol{b}} \cdot [\![\mathcal{Y}_b]\!]. \tag{4.12}$$

Scheme

Our polynomial commitment scheme is parameterized by three components: (a) the encoding scheme $(\mathsf{Enc}_q, \mathsf{Dec}_q)$ defined in Section 4.7.2, (b) A group sampler $\mathcal{G}$ for which the Hidden Order Assumption holds, and (c) a perfectly correct, statistically sound PoE protocol (we present one such protocol over arbitrary groups in Section 4.7.6). We now present all algorithms $(\mathsf{Setup}, \mathsf{Com}, \mathsf{Open}, \mathsf{Eval})$ for the polynomial commitment scheme.

$\mathsf{Setup}(1^\lambda, p, 1^n)$

On input security parameter 1^λ, a prime p, and the number of polynomial variables 1^n, expressed in unary, the algorithm Setup samples group description $\mathbb{G} \leftarrow \mathcal{G}(1^\lambda)$, samples $g \leftarrow \mathbb{G}$, sets $q := q(n, p, \lambda) \in \mathbb{N}$, and outputs public parameters $pp = (q, g, \mathbb{G})$. We require that q be odd such that $q > p \cdot 2^{n \cdot \mathrm{poly}(\lambda)}$ (see [52] for details on this choice of q).

$\mathsf{Com}(pp, \mathcal{Y})$

On input $pp = (q, g, \mathbb{G})$ output by Setup and sequence $\mathcal{Y}$, the algorithm Com computes a commitment to the sequence $\mathcal{Y}$ as $C = g^{\mathsf{Enc}_q(\mathcal{Y})}$. The output of Com is the commitment C and secret decommitment $\mathcal{Z} = (\llbracket \mathcal{Y}_b \rrbracket)_{b \in \{0,1\}^n} \in \mathbb{Z}(p/2)^N$.

$\mathsf{Open}(pp, C, \mathcal{Y}, \mathcal{Z})$

On inputs $pp = (q, g, \mathbb{G})$, C output by Com, committed sequence $\mathcal{Y} \in \mathbb{F}^N$ and decommitment $\mathcal{Z} \in \mathbb{Z}^N$ for $N = 2^n$, the algorithm Open outputs a decision bit. Open outputs 1 if and only if (1) $\mathcal{Z} \subseteq \mathbb{Z}(q/2)^N$; (2) $\mathcal{Y} \equiv \mathcal{Z} \bmod p$; and (3) $C = g^{\mathsf{Enc}_q(\mathcal{Z})}$. Otherwise, Open outputs 0.

$\mathsf{Eval}(pp, C, \zeta, \gamma; \mathcal{Y}, \mathcal{Z})$

On input $pp = (q, g, \mathbb{G})$, $C \in \mathbb{G}$, $\zeta \in \mathbb{F}^n$, $\gamma \in \mathbb{F}$, $\mathcal{Y} \in \mathbb{F}^N$ and $\mathcal{Z} \in \mathbb{Z}^N$ for $N = 2^n$, the Eval algorithm is an interactive protocol (P, V) for the relation,

$$\mathcal{R}_{\mathsf{ml}} = \Big\{ (pp, C, \zeta, \gamma; \mathcal{Y}, \mathcal{Z}) : \mathsf{Open}(pp, C, \mathcal{Y}, \mathcal{Z}) = 1 \wedge \gamma = \mathsf{ML}(\mathcal{Y}, \zeta) \Big\}, \tag{4.13}$$

EvalReduce $(C, k, \zeta, \gamma; Z)$

Input : $C \in \mathbb{G}^\lambda$, $k \in \mathbb{N}$, $\zeta \in \mathbb{F}^n$, $\gamma \in \mathbb{F}^\lambda$, and $Z \in \mathbb{Z}^{\lambda \times 2^{n-k}}$.

Output : Accept or reject.

if $k = n$ **then**

  P sends $Z \in \mathbb{Z}^\lambda$ to V.

  V outputs accept if and only if $\left\|Z\right\|_\infty \leq p(2\lambda)^n$, $\gamma \equiv Z \bmod p$, and $C = g^Z$.

else

  P computes

$$\gamma_0 = \sum_{b \in \{0,1\}^{n-k-1}} (Z(*, 0b) \bmod p) \cdot \prod_{j=1}^{n-k-1} \chi(b_j, \zeta_{j+k+1})$$

$$\gamma_1 = \sum_{b \in \{0,1\}^{n-k-1}} (Z(*, 1b) \bmod p) \cdot \prod_{j=1}^{n-k-1} \chi(b_j, \zeta_{j+k+1})$$

  P computes

$$C_0 = g^\ell \qquad \text{where } \ell = \sum_{b \in \{0,1\}^{n-k-1}} q^b \cdot Z(*, 0b)$$

$$C_1 = g^r \qquad \text{where } r = \sum_{b \in \{0,1\}^{n-k-1}} q^b \cdot Z(*, 1b)$$

  P sends (γ_0, γ_1) and (C_0, C_1) to V.

  V checks $\gamma \stackrel{?}{=} \gamma_0 \cdot (1 - \zeta_{k+1}) + \gamma_1 \cdot \zeta_{k+1}$.

  P and V run $\mathsf{PoE}(C_1, C/C_0, q, n-k-1, \lambda)$ which is a proof showing $C_1(i)^{q^{2^{n-k-1}}} = C(i)/C_0(i)$ for every $i \in [\lambda]$ (see Section 4.7.6). Here, C/C_0 denotes coordinate-wise division of the elements of C by the elements of C_0.

  V samples $U = [U_0 \| U_1] \xleftarrow{\$} \{0,1\}^{\lambda \times 2\lambda}$ and sends U to P, where $U_0, U_1 \in \{0,1\}^{\lambda \times \lambda}$.

  P and V compute

$$\gamma' = U_0 \cdot \gamma_0 + U_1 \cdot \gamma_1 \qquad C' = (U_0 \star C_0) \odot (U_1 \star C_1)$$

  For $Z_0, Z_1 \in \{0,1\}^{\lambda \times 2^{n-k-1}}$ such that $Z = [Z_0 \| Z_1]$, P computes

$$Z' = U_0 \cdot Z_0 + U_1 \cdot Z_1.$$

  return $\mathsf{EvalReduce}(C', k+1, \zeta, \gamma'; Z')$

Figure 4.7. Description of EvalReduce.

where on common input (pp, C, ζ, γ), P tries to convince V that it knows a committed sequence $\mathcal{Y} \in \mathbb{F}^N$ and an integer sequence $\mathcal{Z} \in \mathbb{Z}^N$ such that $\mathsf{Open}(pp, C, \mathcal{Y}, \mathcal{Z}) = 1$ and γ is the evaluation of the multilinear polynomial defined by $\mathcal{Y}$ at evaluation point $\zeta = (\zeta_n, \ldots, \zeta_1)$; that is, $\gamma \stackrel{?}{=} \mathsf{ML}(\mathcal{Y}, \zeta)$. More specifically, both the committer and receiver in Eval first make λ many copies of the statement $(C, \zeta, \gamma; \mathcal{Z})$ as $(C, \zeta, \gamma; Z)$, where $C = (C, \ldots, C) \in \mathbb{G}^\lambda$, $\gamma = (\gamma, \ldots, \gamma) \in \mathbb{F}^\lambda$, and $Z \in \mathbb{Z}^{\lambda \times N}$ is a matrix such that $Z(i, b) := \mathcal{Z}_b$ for every $i \in [\lambda]$ and $b \in \{0,1\}^n$. The committer and receiver then run the subroutine $\mathsf{EvalReduce}$, presented in Figure 4.7.

$\mathsf{EvalReduce}$ is a recursive protocol which given the statement $(C, \zeta, \gamma; Z)$ proves that $\gamma_i = \mathsf{ML}(Z(i, *), \zeta)$ and $C_i = \mathsf{Com}(Z(i, *))$ for every $i \in [\lambda]$, where $Z(i, *) \in \mathbb{Z}^{1 \times N}$ is the i^{th} row of Z. This is done via a divide and conquer approach. Let $P_i \colon \mathbb{F}^n \to \mathbb{F}$ be the multilinear polynomial defined by row i of matrix Z for every $i \in [\lambda]$. For presentation, we focus on the polynomial P_1. To prove that $\gamma_1 = P_1(\zeta)$ and $C_1 = \mathsf{Com}(P_1) = g^{\mathsf{Enc}_q(P_1)}$, the committer first splits P_1 into it's "left" and "right" halves, defined by $P_{1,0}(\cdot) = P_1(\cdot, 0)$ and $P_{1,1}(\cdot, 1)$. Then it computes evaluations of these polynomials at the point $\zeta' = (\zeta_n, \ldots, \zeta_2)$ to obtain $\gamma_{1,0} = P_{1,0}(\zeta')$ and $\gamma_{1,1} = P_{1,1}(\zeta')$ (Line 5). Similarly, the committer also computes commitments $C_{1,0} = g^{\mathsf{Enc}_q(P_{1,0})}$ and $C_{1,1} = g^{\mathsf{Enc}_q(P_{1,1})}$ (Line 6). The claims $(\gamma_{1,0}, \gamma_{1,1})$ and $(C_{1,0}, C_{1,1})$ are then sent to the receiver. If indeed the committer defined $P_{1,0}$ and $P_{1,1}$ correctly, then $\gamma_1 = \gamma_{1,0} \cdot (1 - \zeta_1) + \gamma_{1,1} \cdot \zeta_1$ (Line 8) and $C_{1,0} \cdot C_{1,1}^{q^T} = C_1$ for $T = 2^{n-1}$. Since checking $C_{1,0} \cdot C_{1,1}^{q^T} = C_1$ directly is too costly to the receiver, the committer and prover run a *proof of exponent* protocol PoE to prove that equality holds (Line 9). The committer does simultaneously this for all polynomials P_i. The receiver then specifies random linear combinations $U \xleftarrow{\$} \{0,1\}^{\lambda \times 2\lambda}$ (Line 10). The committer and receiver then obtain a set of λ new evaluations $\gamma'_i = \sum_{j \in [\lambda]} U(i,j) \cdot \gamma_{j,0} + U(i, 2j) \cdot \gamma_{j,1}$ and λ new commitments $C'_i = \prod_{j \in [\lambda]} (C_{j,0})^{U(i,j)} \cdot (C_{j,1})^{U(i,2j)}$ (Line 11). This also defines new matrix $Z' = U_0 \cdot Z_0 + U_1 \cdot Z_1$ (Line 12) for $U = [U_0 \| U_1]$ and $Z = [Z_0 \| Z_1]$. If the committer is honest, then the polynomial P'_1 defined by the row $Z'(1, *)$ satisfies $\gamma'_1 = P'_1(\zeta')$ and $C'_1 = g^{\mathsf{Enc}_q(P'_1)}$ (and similarly for all other polynomials P'_i defined by row $Z'(i, *)$). The committer and receiver recurse via the above λ-to-2λ-to-λ reduction until the matrix Z is a single column; at this point, Z is sent to

the receiver. The receiver checks if the entries of Z are appropriately bounded, if the final vector $\gamma \equiv Z (\mathrm{mod}\, p)$, and if $C = g^Z = (g^{Z_1}, \ldots, g^{Z_\lambda})$ (Line 3).

Remark 4.7.6. *For simplicity of presentation, we let the (computational) security parameter λ_c given as input to* Setup *to be equal to the statistical security parameter λ_s given to* Eval*. However, they may be set differently: λ_c needs to be set so that $2^{\lambda_c^\epsilon}$ is larger than the running time of the adversary (generally, $\lambda_c = 2048$ for RSA groups to have security against 2^{80} time adversaries). However, λ_s needs to set so that the success probability of the adversary (we want to tolerate) is upper-bounded by $2^{-\Omega(\lambda_s)}$, in fact, even relatively small values of λ_s would be sufficient for security, and offer qualitatively more efficient implementations.*

We discuss the efficiency of our polynomial commitment scheme in Section 4.7.3. We defer the correctness and security proofs to our paper [52]. Finally, we present our new proof of exponent protocol in Section 4.7.6.

4.7.3 Space-Efficient Multilinear Polynomial Commitment Scheme in the Streaming Model

In this section, we show that given streaming access to the sequence $\mathcal{Y} \in \mathbb{F}^N$ of evaluations of a multilinear polynomial on the Boolean hypercube, our multilinear polynomial commitment scheme of Section 4.7.2 is a streamable multilinear polynomial commitment scheme.

Theorem 4.7.7. *The multilinear polynomial commitment scheme of Section 4.7.2 has the following efficiencies.*

- Com *outputs a commitment that is a single group element of size* $\mathrm{poly}(\lambda)$ *bits, runs in time* $N \cdot \mathrm{poly}(\log(N), \log(p), \lambda)$ *and space* $n + \mathrm{poly}(\lambda)$ *bits, and uses a single pass over the sequence of evaluations $\mathcal{Y}$.*
- *The committer in the* Eval *protocol runs in time* $N \cdot \mathrm{poly}(\log(N), \log(p), \lambda)$*, space* $\log(N) \cdot \mathrm{poly}(\log(p), \lambda)$ *bits, and uses* $O(\log(N))$ *passes over $\mathcal{Y}$.*
- *The receiver in the* Eval *protocol runs in time* $\mathrm{poly}(\log(N), \log(p), \lambda)$ *and space* $\log(N) \cdot \mathrm{poly}(\log(p), \lambda)$ *bits.*

- *The communication complexity of* Eval *is* $\text{poly}(\log(N) \cdot \log(p), \lambda)$ *bits and has* $O(\log(N))$ *rounds of communication.*

4.7.4 Space-Efficient Implementation Overview

Our goal is to implement the committer algorithm of our polynomial commitment scheme in small-space. The committer is assumed to have multi-pass streaming access to the evaluations $\mathcal{Y} \in \mathbb{F}^N$ of a multilinear polynomial over the Boolean hypercube. Given this streaming access, we need to implement the following computations in small-space: (1) computation of $\mathsf{Com}(\mathcal{Y})$; (2) computation of $\mathsf{ML}(\mathcal{Y}, \zeta)$ for $\zeta \in \mathbb{F}^n$ specified by the receiver; and (3) computation of all committer messages in the Eval protocol. The main technical challenge is implementing the committer algorithm of Eval in small-space. Recall that Eval is an interactive protocol where on common input (C, ζ, γ) the committer tries to convince a receiver that it knows $\mathcal{Y} \in \mathbb{F}^N$ and $\mathcal{Z} \in \mathbb{Z}^N$ such that To do so, the committer and receiver construct the statement (C, ζ, γ) where $C = (C, \ldots, C) \in \mathbb{G}^\lambda$, $\gamma = (\gamma, \ldots, \gamma) \in \mathbb{F}^\lambda$. The committer then defines matrix $Z \in \mathbb{Z}^{\lambda \times N}$ where $Z(i, b) = \mathcal{Z}_b$ for every $i \in [\lambda]$ and $b \in \{0,1\}^n$. The committer and receiver then run the protocol $\mathsf{EvalReduce}(C, 0, \zeta, \gamma; Z)$.

$\mathsf{EvalReduce}$ is a recursive protocol between the committer and receiver which for input $(C, k, \zeta, \gamma; Z)$ proves the statement

$$\text{“}\gamma_i = \mathsf{ML}(Z(i, *), \zeta) \quad \wedge \quad C_i = \mathsf{Com}(Z(i, *)) \text{ for every } i \in [\lambda],\text{”} \tag{4.14}$$

where $C \in \mathbb{G}^\lambda$, $\zeta \in \mathbb{F}^n$, $\gamma \in \mathbb{F}^\lambda$, and $Z \in \mathbb{Z}^{\lambda \times 2^n}$. To prove Equation (4.14), the protocol reduces the above λ claims to λ new claims about some matrix $Z' \in \mathbb{Z}^{\lambda \times 2^{n-1}}$. This reduction is performed as follows. Let $Z = [Z_0 \| Z_1]$ for $Z_0, Z_1 \in \mathbb{Z}^{\lambda \times 2^{n-1}}$. First the prover constructs “left-and-right” evaluations $\gamma_0, \gamma_1 \in \mathbb{F}^\lambda$ and “left-and-right” commitments $C_0, C_1 \in \mathbb{G}^\lambda$ defined as

$$(\gamma_0)_i = \mathsf{ML}(Z_0(i, *), (\zeta_n, \ldots, \zeta_2)) \qquad (\gamma_1)_i = \mathsf{ML}(Z_1(i, *), (\zeta_n, \ldots, \zeta_2)) \tag{4.15}$$

$$(C_0)_i = \mathsf{Com}(Z_0(i, *)) \qquad (C_1)_i = \mathsf{Com}(Z_1(i, *)), \tag{4.16}$$

for every $i \in [\lambda]$. The verifier then sends challenge matrix $U = [U_0 \| U_1] \xleftarrow{\$} \{0,1\}^{\lambda \times 2\lambda}$, and the committer and receiver define values $\gamma' \in \mathbb{F}^\lambda$ and $C' \in \mathbb{G}^\lambda$ as[20]

$$\gamma' = U_0 \cdot \gamma_0 + U_1 \cdot \gamma_1 \qquad\qquad C' = (U_0 \star C_0) \odot (U_1 \star C_1).$$

The committer then defines matrix $Z' = U_0 \cdot Z_0 + U_1 \cdot Z_1 \in \mathbb{Z}^{\lambda \in 2^{n-1}}$, and the committer and receiver recurse on the statement $(C', k+1, \zeta, \gamma'; Z')$. The recursion continues for n rounds: at the end, the committer sends over the matrix Z, which has been reduced to a single column of λ integers, and the receiver performs a variety of checks and accepts or rejects.

Fixing notation, for any $k \in \{0, 1, \ldots, n\}$, let $(C^{(k)}, \zeta, \gamma^{(k)}; Z^{(k)})$ be the input to the k^{th} round of $\mathsf{EvalReduce}$, where $Z^{(k)} = [Z_0^{(k)} \| Z_1^{(k)}] \in \mathbb{Z}^{\lambda \times 2^{n-k}}$. Further let $\gamma_0^{(k)}$ and $\gamma_1^{(k)}$ denote the left-and-right evaluations and let $C_0^{(k)}$ and $C_1^{(k)}$ denote the left-and-right commitments computed by the committer in round k, and let $U_0^{(k)}$ and $U_1^{(k)}$ denote the receiver challenges. Our goal is to compute the left-and-right evaluations and commitments in small-space, for any round k. Observe that by Equations (4.15) and (4.16) the values $\gamma_0^{(k)}$, $\gamma_1^{(k)}$, $C_0^{(k)}$, and $C_1^{(k)}$ are linear combinations of the columns of the matrix $Z^{(k)}$.

For space-efficiency, the committer cannot store the matrix $Z^{(k)}$, as this would use $\Omega(N)$ bits (for most $k \in \{0, 1, \ldots, n\}$), so the committer does not have direct access to $Z^{(k)}$. Instead, the committer has multi-pass streaming access to the integer sequence $\mathcal{Z}$, which implies that it has the same streaming access to the columns of the matrix $Z^{(0)} = Z$ in lexicographic (i.e., left-to-right) order. Further by construction we have that $Z^{(k)} = U_0^{(k-1)} \cdot Z_0^{(k-1)} + U_1^{(k-1)} \cdot Z_1^{(k-1)}$ for every $k \in [n]$. This implies that the columns of $Z^{(k)}$ are linear combinations of the columns of $Z^{(0)}$. Leveraging this observation, we have that the left-and-right evaluations and commitments are linear combinations of the columns of $Z^{(0)}$, where the weights of these combinations depend on the receiver challenges $U_0^{(j)}, U_1^{(j)}$ for $j \in \{0, 1, \ldots, k-1\}$ and the evaluation point ζ. Thus so long as these weights time- and space-efficient to compute, we can construct streaming algorithms for the committer messages time- and space-efficiently.

The remainder of this section is dedicated to proving Theorem 4.7.7. In the next three sub-sub-sections, we discuss implementing Com in small space, followed by a discussion on

[20]↑Recall that for $M \in \mathbb{Z}^{m \times n}$ and vector $g \in \mathbb{G}^n$, $(M \star g)_i = \prod_j g_j^{M(i,j)}$. See Section 4.7.1.

the efficiency of computing a multilinear extension in the streaming model, and conclude by discussing how to implement the committer of Eval ins small space. After that, in Section 4.7.5 we discuss the efficiency of the receiver of Eval, and finally in Section 4.7.5 we prove Theorem 4.7.7.

Space-Efficient Implementation of Com

We begin by showing $\mathsf{Com}(\mathcal{Y})$ is computable in small space.

Lemma 4.7.8. *The algorithm* Com *of the polynomial commitment scheme of Section 4.7.2 is implementable in time* $N \cdot (\log(q) + \log(p)) \cdot \mathrm{poly}(\lambda)$ *and space* $n + \mathrm{poly}(\lambda)$ *bits, using a single pass over the sequence* $\mathcal{Y}$.

Proof. Recall that $\mathsf{Com}(\mathcal{Y}) = g^{\mathsf{Enc}_q(\mathcal{Y})}$, where $\mathsf{Enc}_q(\mathcal{Y}) = \sum_{b \in \{0,1\}^n} q^b \cdot [\![\mathcal{Y}_b]\!]$. Let $v := \mathsf{Enc}_q(\mathcal{Y})$. Then

$$g^v = g^{\sum_b q^b \cdot [\![\mathcal{Y}_b]\!]} = \prod_{b \in \{0,1\}^n} (g^{q^b})^{[\![\mathcal{Y}_b]\!]}.$$

We can implement $\mathsf{Com}(pp, \mathcal{Y})$ in a streaming manner by iterating through b in lexicographic order as follows. First, set two values $C = 1_{\mathbb{G}}$ and $D = g$; at the start of the b^{th} iteration, C will be the value $g^{v'}$ for $v' = \sum_{b' < b} q^{b'} \cdot [\![\mathcal{Y}_{b'}]\!]$, and D will be the value g^{q^b}. Now to update C and D from their b^{th} values to their $(b+1)^{th}$ values, we set $C = C \cdot D^{[\![\mathcal{Y}_b]\!]}$, followed by $D = D^q$. Once iteration over b is complete, we output C.

Note that the above algorithm makes a single pass over the stream $\mathcal{Y}$. In the b^{th} iteration, observe that $D^{[\![\mathcal{Y}_b]\!]} = g^{q^b \cdot [\![\mathcal{Y}_b]\!]}$. By the homomorphism $v \mapsto g^v$, it holds that $C \cdot D^{[\![\mathcal{Y}_b]\!]} = g^{v' + q^b \cdot [\![\mathcal{Y}_b]\!]}$ for $v' = \sum_{b' < b} q^{b'} \cdot [\![\mathcal{Y}_{b'}]\!]$. Further, $v' + q^b \cdot [\![\mathcal{Y}_b]\!] = \sum_{b' \leqslant b} q^{b'} \cdot [\![\mathcal{Y}_{b'}]\!]$. This implies that the value C output by the above algorithm satisfies $C = g^{\mathsf{Enc}_q(\mathcal{Y})}$. The described algorithm uses $O(1)$ group elements of storage, which is $\mathrm{poly}(\lambda)$ bits of storage, and we use an additional n bits of storage for the counter b. Further, the algorithm is dominated by $O(N)$ group exponents of size $O(q)$, $O(N)$ exponents of size $O(p)$, and $O(N)$ group multiplications. This gives an overall runtime of $N \cdot (\log(q) + \log(p)) \cdot \mathrm{poly}(\lambda)$. □

Computing $\mathsf{ML}(\mathcal{Y}, \zeta)$

Next we show that $\mathsf{ML}(\mathcal{Y}, \zeta)$ is computable in small space.

Lemma 4.7.9. *For $\zeta \in \mathbb{F}^n$ and $\mathcal{Y} \in \mathbb{F}^N$ for $N = 2^n$, the value $\mathsf{ML}(\mathcal{Y}, \zeta)$ is computable in $N \cdot \log(N) \cdot \text{polylog}(p)$ time and $O(\log(N) \cdot \log(p))$ bits of space, using a single pass over $\mathcal{Y}$.*

Proof. By definition

$$\mathsf{ML}(\mathcal{Y}, \zeta) = \sum_{b \in \{0,1\}^n} \mathcal{Y}_b \cdot \overline{\chi}(b, \zeta),$$

where $\overline{\chi}(b, \zeta) = \prod_{i=1}^{n} \chi(b_i, \zeta_i)$. We can compute $\mathsf{ML}(\mathcal{Y}, \zeta)$ in a streaming manner as follows. First, store an accumulator $\gamma = 0 \in \mathbb{F}$. Then, iterating over $b \in \{0,1\}^n$ in lexicographic order, compute $\gamma = \gamma + \mathcal{Y}_b \cdot \overline{\chi}$, and output γ. Thus we compute $\mathsf{ML}(\mathcal{Y}, \zeta)$ using a single pass over $\mathcal{Y}$. The main complexity of this algorithm is computing $\overline{\chi}$ for every b, which is computable in $O(n) = O(\log(N))$ field multiplications and thus $\log(N) \cdot \text{polylog}(p)$ time. So the overall time complexity of computing $\mathsf{ML}(\mathcal{Y}, \zeta)$ is $N \cdot \log(N) \cdot \text{polylog}(p)$. For space, $\overline{\chi}(b, \zeta)$ is computable using $O(\log(N))$ field elements, and the algorithm above uses an additional $O(1)$ field elements of storage. This gives space complexity $O(\log(N) \cdot \log(p))$ bits. □

Space-Efficient Implementation of Eval

We begin with a lemma which relates the matrix $Z^{(k)}$ to the matrix $Z^{(0)}$ in $\mathsf{EvalReduce}$.

Lemma 4.7.10. *Let $k \in \{0, 1, \ldots, n\}$ denote the k^{th} depth of recursion of algorithm $\mathsf{EvalReduce}$, let $Z^{(k)} \in \mathbb{Z}^{\lambda \times 2^{n-k}}$ be the integer matrix given as input to $\mathsf{EvalReduce}$ during depth k, and let $(U_0^{(j)}, U_1^{(j)})_{j \in \{0,1,\ldots,k-1\}}$ be the receiver challenges in each recursion level $j \in \{0, 1, \ldots, k-1\}$. Then for any $b \in \{0,1\}^{n-k}$, it holds that*

$$Z^{(k)}(*, b) = \sum_{c \in \{0,1\}^k} \left(\prod_{j=0}^{k-1} U_0^{(j)} \cdot (1 - c_{k-j}) + U_1^{(j)} \cdot c_{k-j} \right) \cdot Z^{(0)}(*, c \circ b), \tag{4.17}$$

gammaStreamGen $(k, \zeta, (U^{(i)})_{i\in\{0,1,\ldots,k-1\}})$

Input : Recurion level $k \in \{0, 1, \ldots, n-1\}$, evaluation point $\zeta \in \mathbb{F}^n$, and receiver challenges $U^{(i)} = [U_0^{(i)} \| U_1^{(i)}] \in \{0,1\}^{\lambda\times 2\lambda}$ for every $i \in \{0, 1, \ldots, k-1\}$.

Given : Streaming access to the columns of $Z^{(0)}$ in lexicographic order.

Output : A tuple $(\gamma'_0, \gamma'_1) \in \mathbb{F}^\lambda \times \mathbb{F}^\lambda$.

Set $\gamma'_0 = \gamma'_1 = \mathbf{0}^\lambda \in \mathbb{F}^\lambda$.

foreach $c \in \{0,1\}^k$ *(in lexicographic order)* **do**

  Set $\gamma''_0 = \gamma''_1 = \mathbf{0}^\lambda \in \mathbb{F}^\lambda$.

  foreach $(a \circ b) \in \{0,1\} \times \{0,1\}^{n-k-1}$ *(in lexicographic order)* **do**

    Compute $\chi = \prod_{j=1}^{n-k-1} \chi(b_j, \zeta_{j+k+1})$.

    Set $\hat{c} = c \circ a \circ b \in \{0,1\}^n$.

    if $a = 0$ **then**

      Compute $\gamma''_0 = \gamma''_0 + (Z^{(0)}(*, \hat{c}) \bmod p) \cdot \chi$.

    else

      Compute $\gamma''_1 = \gamma''_1 + (Z^{(0)}(*, \hat{c}) \bmod p) \cdot \chi$.

  Compute

$$\gamma'_0 = \gamma'_0 + M_c \cdot \gamma''_0 \qquad\qquad \gamma'_1 = \gamma'_1 + M_c \cdot \gamma''_1,$$

  where $M_c = \prod_{i=0}^{k-1} \left(U_0^{(i)} \cdot (1 - c_{k-i}) + U_1^{(i)} \cdot c_{k-i}\right)$.

return (γ'_0, γ'_1)

Figure 4.8. Space-Efficient computation of recursive values γ'_0, γ'_1.

where $Z^{(k)}(*, b)$ denotes the b^{th} column of the matrix $Z^{(k)}$ and

$$\prod_{j=0}^{k-1} U_0^{(j)} \cdot (1 - c_{k-j}) + U_1^{(i)} \cdot c_{k-j} =$$
$$(U_0^{(0)} \cdot (1 - c_k) + U_1^{(0)} \cdot c_k) \cdots (U_0^{(k-1)} \cdot (1 - c_1) + U_1^{(k-1)} \cdot c_1).$$

The above lemma holds by induction on k, see [52] for the proof. Given Lemma 4.7.10, we show that $\gamma_0^{(k)}$ and $\gamma_1^{(k)}$ are computable in small space.

Lemma 4.7.11. *Let $\gamma_0^{(k)}, \gamma_1^{(k)} \in \mathbb{F}_p^\lambda$ be the left and right evaluations computed by the committer in recursion level $k \in \{0, 1, \ldots, n-1\}$, and let $(U_0^{(j)}, U_1^{(j)})_{j\in\{0,1,\ldots,k-1\}}$ be the receiver*

challenges. Then $\gamma_0^{(k)}$ *and* $\gamma_1^{(k)}$ *are computable in time* $N \cdot \mathrm{poly}(\log(N), \log(p), \lambda)$*, space* $\mathrm{poly}(\log(N), \log(p), \lambda)$ *bits, and using a single pass over the columns of* $Z^{(0)}$*.*

Sketch. At a high level, we leverage linearity and the additive homomorphism of $\mathbb{F}$ to compute $\gamma_0^{(k)}$ and $\gamma_1^{(k)}$ in small space. Notice that the values $\gamma_0^{(k)}, \gamma_1^{(k)}$ are linear combinations of the stream $Z^{(k)}$, given by

$$\gamma_0^{(k)} = \sum_{b \in \{0,1\}^{n-k-1}} (Z^{(k)}(*, 0b) \bmod p) \cdot \prod_{j=1}^{n-k-1} \chi(b_j, \zeta_{j+k+1}),$$
$$\gamma_1^{(k)} = \sum_{b \in \{0,1\}^{n-k-1}} (Z^{(k)}(*, 1b) \bmod p) \cdot \prod_{j=1}^{n-k-1} \chi(b_j, \zeta_{j+k+1}).$$

By Lemma 4.7.10, the columns of $Z^{(k)}$ are linear combinations of the columns of $Z^{(0)}$. Thus both $\gamma_0^{(k)}$ and $\gamma_1^{(k)}$ are linear combinations of the columns of $Z^{(0)}$. Focusing on $\gamma_0^{(k)}$, we have

$$\begin{aligned}\gamma_0^{(k)} &= \sum_{b \in \{0,1\}^{n-k-1}} \left(\sum_{c \in \{0,1\}^k} M_c \cdot Z^{(0)}(*, c \circ 0b) \right) \bmod p \cdot \prod_{j=1}^{n-k-1} \chi(b_j, \zeta_{j+k+1}) \\ &= \sum_{c \in \{0,1\}^k} M_c \cdot \left(\sum_{b \in \{0,1\}^{n-k-1}} Z^{(0)}(*, c \circ 0b) \bmod p \cdot \prod_{j=1}^{n-k-1} \chi(b_j, \zeta_{j+k+1}) \right),\end{aligned}$$

and by symmetry, for $\gamma_1^{(k)}$ we have

$$\gamma_1^{(k)} = \sum_{c \in \{0,1\}^k} M_c \cdot \left(\sum_{b \in \{0,1\}^{n-k-1}} Z^{(0)}(*, c \circ 1b) \bmod p \cdot \prod_{j=1}^{n-k-1} \chi(b_j, \zeta_{j+k+1}) \right).$$

Notice that by iterating over $c \in \{0,1\}^k$ in lexicographic order, then over $(a \circ b) \in \{0,1\} \times \{0,1\}^{n-k-1}$ in lexicographic order per iteration of c, the string $\hat{c} = (c \circ a \circ b) \in \{0,1\}^n$ iterates over $\{0,1\}^n$ in lexicographic order. Thus the above equations access the columns of $Z^{(0)}$ in lexicographic order, and we can compute $\gamma_0^{(k)}$ and $\gamma_1^{(k)}$ in a single pass over the columns of $Z^{(0)}$. The algorithm gammaStreamGen exactly computes $\gamma_0^{(k)}$ and $\gamma_1^{(k)}$ as in the above equations.

Given the above equations, we observe that: (1) computing the product $\chi = \prod_i \chi(b_j, \zeta_{j+k+1})$ takes $\log(N) \cdot \mathrm{polylog}(p)$ time and $O(\log(N) \cdot \log(p))$ bits of space; (2) computing the inner

comStreamGen $(k, q, g, (U^{(i)})_{i\in\{0,1,\ldots,k-1\}})$

Input : Recurion level $k \in \{0, 1, \ldots, n-1\}$, integer $q \in \mathbb{N}$, group element $g \in \mathbb{G}$, and receiver challenges $U^{(i)} = [U_0^{(i)} \| U_1^{(i)}] \in \{0,1\}^{\lambda\times 2\lambda}$ for every $i \in \{0, 1, \ldots, k-1\}$.

Given : Streaming access to the columns of $Z^{(0)}$ in lexicographic order.

Output : A tuple $(C'_0, C'_1) \in \mathbb{G}^\lambda \times \mathbb{G}^\lambda$.

Set $C'_0 = C'_1 = \mathbf{1}^\lambda \in \mathbb{G}^\lambda$.
foreach $c \in \{0,1\}^k$ *(in lexicographic order)* **do**
  Set $C''_0 = C''_1 = \mathbf{1}^\lambda \in \mathbb{G}^\lambda$.
  foreach $a \in \{0,1\}$ *(in lexicographic order)* **do**
    Set $C = g$.
    foreach $b \in \{0,1\}^{n-k-1}$ *(in lexicographic order)* **do**
      Set $\hat{c} = c \circ a \circ b \in \{0,1\}^n$.
      if $a = 0$ **then**
        Compute $C''_0 = C''_0 \odot C^{Z^{(0)}(*,\hat{c})}$.
      else
        Compute $C''_1 = C''_1 \odot C^{Z^{(0)}(*,\hat{c})}$.
      Compute $C = C^q$.
  Compute

$$C'_0 = C'_0 \odot (M_c \star C''_0) \qquad C'_1 = C'_1 \odot (M_c \star C''_1)\,,$$

  where $M_c = \prod_{i=0}^{k-1} \left(U_0^{(i)} \cdot (1 - c_{k-i}) + U_1^{(i)} \cdot c_{k-i}\right)$.
return (C'_0, C'_1)

Figure 4.9. Space-Efficient computation of recursive values C'_0, C'_1.

summation takes time $2^{n-k-1} \cdot \lambda \cdot \log(N) \cdot \text{polylog}(p)$ and $O(\lambda \cdot \log(N) \cdot \log(p))$ bits of space; and (3) computing M_c times the inner summation takes time $\text{poly}(\log(p), \lambda)$ and $\text{poly}(\lambda)$ bits of space. So the overall complexity of the entire summation is $N \cdot \text{poly}(\log(N), \log(p), \lambda)$ time and $\text{poly}(\log(N), \log(p), \lambda)$ bits of space. □

Next we show that $C_0^{(k)}, C_1^{(k)}$ are computable in small space.

Lemma 4.7.12. *Let $C_0^{(k)}, C_1^{(k)} \in \mathbb{G}^\lambda$ be the left and right commitments computed by the committer in recursion level $k \in \{0, 1, \ldots, n-1\}$, and let $(U_0^{(j)}, U_1^{(j)})_{j\in\{0,1,\ldots,k-1\}}$ be the receiver challenges. Then $C_0^{(k)}$ and $C_1^{(k)}$ are computable in time $N \cdot \text{poly}(\log(N), \log(q), \lambda)$, space $\text{poly}(\log(N), \log(q), \lambda)$ bits, and using a single pass over the columns of $Z^{(0)}$.*

Sketch. At a high level, we leverage the linear homomorphic properties of the group $\mathbb{G}$ to compute the values of $C_0^{(k)}$ and $C_1^{(k)}$ in small space. The values $C_0^{(k)}$ and $C_1^{(k)}$ are computed via $g^{\ell^{(k)}}$ and $g^{r^{(k)}}$, where $\ell^{(k)}$ and $r^{(k)}$ are linear combinations of the columns of $Z^{(k)}$, given by the equations

$$\ell^{(k)} = \sum_{b \in \{0,1\}^{n-k-1}} q^b \cdot Z^{(k)}(*, 0b) \qquad r^{(k)} = \sum_{b \in \{0,1\}^{n-k-1}} q^b \cdot Z^{(k)}(*, 1b).$$

By Lemma 4.7.10, the columns of $Z^{(k)}$ are linear combinations of the columns of $Z^{(0)}$, so the powers $\ell^{(k)}$ and $r^{(k)}$ are linear combinations of the columns of $Z^{(0)}$. Focusing on $\ell^{(k)}$, we have

$$\begin{aligned}\ell^{(k)} &= \sum_{b \in \{0,1\}^{n-k-1}} q^b \sum_{c \in \{0,1\}^k} M_c \cdot Z^{(0)}(*, c \circ 0b) \\ &= \sum_{c \in \{0,1\}^k} M_c \cdot \sum_{b \in \{0,1\}^{n-k-1}} q^b \cdot Z^{(0)}(*, c \circ 0b),\end{aligned}$$

and by symmetry, for $r^{(k)}$ we have

$$r^{(k)} = \sum_{c \in \{0,1\}^k} M_c \cdot \sum_{b \in \{0,1\}^{n-k-1}} q^b \cdot Z^{(0)}(*, c \circ 1b).$$

Note again that the string $\hat{c} = (c \circ a \circ b) \in \{0,1\}^n$ iterates over $\{0,1\}^n$ in lexicographic order. Thus we can compute the powers $\ell^{(k)}$ and $r^{(k)}$ using a single pass over the stream of columns of $Z^{(0)}$, which allows us to compute $C_0^{(k)}$ and $C_1^{(k)}$ in a single pass over the columns of $Z^{(0)}$. The algorithm $\mathsf{comStreamGen}$ exactly computes $C_0^{(k)}$ and $C_1^{(k)}$.

Given $\mathsf{comStreamGen}$, we observe that: (1) Lines 9 and 11 take time and space $\mathrm{poly}(\lambda)$ to compute; (2) Line 12 takes time $\log(q) \cdot \mathrm{poly}(\lambda)$ and $\mathrm{poly}(\lambda)$ space to compute; and (3) Line 13 takes time and space $\log(N) \cdot \mathrm{poly}(\lambda)$ space to compute. Thus the complexity is $N \cdot \mathrm{poly}(\log(N), \log(p), \lambda)$ time and $\mathrm{poly}(\log(N), \log(q), \lambda)$ space. □

Efficiency of PoE.

During any recursive round $k \in \{0, 1, \ldots, n-1\}$ of the algorithm EvalReduce, the committer P and receiver V additionally engage in a *proof of exponent* protocol PoE, with inputs $(C_1^{(k)}, C^{(k)}/C_0^{(k)}, q, n-k-1, \lambda)$, which is an interactive proof of the statement

$$\forall i \in [\lambda]\colon\ (C_1^{(k)})_i^{q^{2^{n-k-1}}} = (C^{(k)}/C_0^{(k)})_i,$$

where $C^{(k)} \in \mathbb{G}^\lambda$ is the current commitment given as input to EvalReduce. Section 4.7.6 discusses the protocol PoE in detail. For the purposes of EvalReduce, we are interested in the time and space overhead incurred by the committer P during any execution of PoE. By Theorem 4.7.15, setting $t = n-k-1 = O(\log(N))$ and recalling that $|\mathbb{G}| \leqslant 2^\lambda$, we have that the committer P runs in time $N \cdot \mathrm{poly}(\log(q), \mathrm{loglog}(N), \lambda)$ and space $\mathrm{poly}(\log(q), \mathrm{loglog}(N), \lambda)$ bits during any execution of PoE in any recursive round $k \in \{0, 1, \ldots, n-1\}$ of EvalReduce. Further, PoE in any recursive round k has round complexity $O(\log N)$ and communication complexity $\log(N) \cdot \mathrm{poly}(\lambda)$.

Computing the Final Committer Message Efficiently.

We now show that the final committer message $Z^{(n)}$ is computable in small space.

Lemma 4.7.13. *Let $(U_0^{(j)}, U_1^{(j)})_{j \in \{0,1,\ldots,n-1\}}$ be all receiver challenges. Then $Z^{(n)} \in \mathbb{Z}^\lambda$ is computable in time $N \cdot \mathrm{poly}(\log(N), \log(p), \lambda)$ and space $O(\lambda^2 \cdot \log(N) \cdot \log(p))$ bits using a single pass over the stream $Z^{(0)}$.*

Proof. Let $(U_0^{(j)}, U_1^{(j)})_{j \in \{0,1,\ldots,n-1\}}$ be all receiver challenges. By Eq. (4.17) of Lemma 4.7.10, we have that

$$Z^{(n)} = \sum_{c \in \{0,1\}^n} \left(\prod_{i=0}^{n-1} U_0^{(i)} \cdot (1 - c_{k-i}) + U_1^{(i)} \cdot c_{k-i} \right) \cdot Z^{(0)}(*, c).$$

Notice that per iteration of $c \in \{0,1\}^n$, the column $Z^{(0)}(*, c)$ is multiplied on the left by $n = \log(N)$ binary matrices. Computing this product is dominated by $O(\log(N) \cdot \lambda^2)$ integer additions, and this product is computed N times. Thus computing $Z^{(n)}$ takes

$N \cdot \text{poly}(\log(N), \log(p), \lambda)$ time. For space, note that $\left\|Z^{(n)}\right\|_\infty = O(N \cdot p \cdot \lambda^n)$, $Z^{(n)}$ is a vector of length λ, and that the committer stores all receiver challenges. Thus the space complexity is $O(\lambda^2 \cdot \log(N) \cdot \log p)$ bits. □

4.7.5 Receiver Efficiency

We have so far only discussed the efficiency of the committer algorithm P. We now argue that the receiver of Eval is efficient.

Lemma 4.7.14. *The receiver of* EvalReduce *runs in time* $\text{poly}(\log(N), \log(q), \log(p), \lambda)$ *and space* $\log(N) \cdot \text{poly}(\log(q), \log(p), \lambda)$.

Proof. In the Eval protocol, the receiver only performs computation in the sub-protocol EvalReduce. First consider round $k = n$: the receiver checks if a vector $Z \in \mathbb{Z}^\lambda$ is properly bounded, checks if $\gamma \equiv Z \bmod p$, and checks if $C = g^Z$. The receiver complexity at this step is dominated by computing g^Z. In an honest execution, Z is a vector such that $\left\|Z\right\|_\infty = O(N \cdot p \cdot \lambda^n)$, so computing g^Z takes time $\text{poly}(\log(N), \log(p), \lambda)$.

Now consider any round $k \in \{0, 1, \ldots, n-1\}$ of the EvalReduce protocol. In round k, the input to EvalReduce includes vectors $C^{(k)} \in \mathbb{G}^\lambda$ and $\gamma^{(k)} \in \mathbb{F}^\lambda$. The receiver receives values $(\gamma_0^{(k)}, \gamma_1^{(k)}) \in \mathbb{F}^\lambda \times \mathbb{F}^\lambda$ and $(C_0^{(k)}, C_1^{(k)}) \in \mathbb{G}^\lambda \times \mathbb{G}^\lambda$ from the committer. The receiver first checks of the claimed vector of evaluations $\gamma^{(k)}$ is equal to $\gamma_0^{(k)} \cdot (1 - \zeta_{k+1}) + \gamma_1^{(k)} \cdot \zeta_{k+1}$; this check takes time $\lambda \cdot \text{polylog}(p)$. Next, the committer P and receiver V run $\mathsf{PoE}(C_1, C/C_1, q, n-k-1, \lambda)$. By Theorem 4.7.15, for $t = n-k-1 = O(\log N)$, the receiver runs in time $\log(N) \cdot \text{poly}(\log(q), \lambda)$. Finally, the receiver samples $U_0^{(k)}, U_1^{(k)} \xleftarrow{\$} \{0,1\}^{\lambda \times \lambda}$ and computes $\gamma^{(k+1)} = U_0^{(k)} \cdot \gamma_0^{(k)} + U_1^{(k)} \cdot \gamma_1^{(k)}$ and $C^{(k+1)} = (U_0^{(k)} \star C_0^{(k)}) \odot (U_1^{(k)} \star C_1^{(k)})$. Since $U_0^{(k)}, U_1^{(k)}$ are binary matrices, the computation of $\gamma^{(k+1)}$ is dominated by $O(\lambda^2)$ field additions and the computation of $C^{(k+1)}$ is dominated by $O(\lambda^2)$ group multiplications. Thus this step takes $\text{poly}(\log(p), \lambda)$ time. Therefore the receiver in Eval runs in $\text{poly}(\log(N), \log(q), \log(p), \lambda)$ time.

During any recursive round $k \in \{0, 1, \ldots, n-1\}$, outside of the PoE protocol, the receiver stores $O(\lambda + \log(N))$ field elements, $O(\lambda)$ group elements, and a single binary matrix with $O(\lambda^2)$ entries. So outside of the PoE protocol, the receiver stores $\text{poly}(\log(N), \log(p), \lambda)$ bits. Within the PoE protocol, the receiver only stores q, $O(\lambda)$ group elements, and $O(\lambda^2)$ bits for

its challenge matrices. Thus in the PoE protocol, the receiver stores $\text{poly}(\log(q), \lambda)$ bits. In the final recursion round $k = n$, in addition to $O(\lambda)$ field and group elements, the receiver also obtains the integer vector Z, where $\left\|Z\right\|_\infty = O(N \cdot p \cdot \lambda^n)$, which uses $\log(N) \cdot \text{poly}(, \log(p), \lambda)$ bits. Finally note that the receiver only stores λ field and group elements between rounds. Therefore the receiver space complexity is $\log(N) \cdot \text{poly}(\log(q), \log(p), \lambda)$ bits. □

Proof of Theorem 4.7.7

We first note that to obtain $O(\log(N))$ round complexity, we push all PoE instances to the final round of EvalReduce and run all instances in parallel. Then by Section 4.7.2, for $q = \Theta(p \cdot 2^{\log(N) \cdot \text{poly}(\lambda)})$ we have that Theorem 4.7.7 follows from Lemmas 4.7.8, 4.7.9, 4.7.11, 4.7.12 and 4.7.14 and Theorem 4.7.15.

4.7.6 Proof-of-Exponentiation in Arbitrary Groups

For some group $\mathbb{G}$ and base $q \in \mathbb{Z}$ consider the language

$$\mathcal{L}_{\mathbb{G},q} = \left\{ (x, y, t) \in \mathbb{G} \times \mathbb{G} \times \mathbb{N} \ : \ x^{q^{2^t}} = y \right\}. \tag{4.18}$$

Note that this problem can be solved in time roughly 2^t (by repeated squaring), but for some groups it is conjectured to not be solvable in significantly less time (even when leveraging parallelization). Indeed, an instantiation of this language using RSA groups underlies the original time-lock puzzle construction by Rivest, Shamir and Wagner [227]. This problem has also been used recently for constructing *verifiable delay functions* (VDFs). We show a extension of a recent protocol due to Pietrzak [219] that works for general groups.

Theorem 4.7.15. *Let $\mathbb{G}$ be a group whose elements have $O(\log(|\mathbb{G}|))$-bit descriptions, and whose group operations take time* $\text{polylog}(|\mathbb{G}|)$*, and let $q \in \mathbb{N}$. There exists a perfectly correct, statistically sound public-coin interactive-proof for $\mathcal{L}_{\mathbb{G},q}$ with the following efficiency properties for exponent parameter t:*

1. *The communication complexity is $O(t\lambda^2 + t\lambda \log(|\mathbb{G}|))$ and there are t rounds.*

2. *The prover runs in time* $2^t \cdot \text{poly}(\log(q), \log(|\mathbb{G}|), \lambda)$ *and uses space* $O(\lambda \cdot \log(|\mathbb{G}|)) + \log(t) + \log(q) + \lambda^2$

3. *The verifier runs in time* $t \cdot \text{poly}(\log(|\mathbb{G}|), \log(q), \lambda)$.

Pietrzak [219] gave an elegant interactive protocol for verifying membership in this group in time roughly t, for groups $\mathbb{G}$ where subgroups of small order do not exist (or are hard to find). When such small order subgroups do not exist, as is the case for RSA groups,[21] his protocol is *statistically* sound. The main downside of RSA groups is that they require a trusted setup (i.e., in the terminology of Definition 4.5.1 they are *private-coin*). As an alternative, Pietrzak's protocol can be instantiated with class groups which are public-coin but the resulting protocol only achieves computational soundness under the assumption that small-order subgroups for class groups are computationally hard to find [65]. We mention that Wesolowski [256] also presented a computationally sound protocol for class groups which is concretely efficient at the cost of making very strong assumptions (i.e., the Adaptive Root Assumption).

We overcome the limitations of both works and give a Proof of Exponentiation (PoE) proof-system for $\mathcal{L}_{\mathbb{G},q}$ which: (1) works for *arbitrary* groups (without any assumptions) and (2) achieves *statistical* soundness. We emphasize that our protocol and the security proof are oblivious to the structure of the group, and our proof of soundness is (arguably) simpler than Pietrzak's. In particular, we only rely on the following elementary fact about random subset products in groups.

Fact 4.7.16 (Random Subset Product Principle)**.** *Let* $\mathbb{G}$ *be a group, let* $\mathbf{1}_{\mathbb{G}}$ *be its identity element and let* $g_1, \ldots, g_n \in \mathbb{G}$*, so that at least one of them is not the identity element. Then:*

$$\Pr_{S \subseteq [n]} \left[\prod_{i \in S} g_i = \mathbf{1}_{\mathbb{G}} \right] \leqslant 1/2.$$

[21] To be more precise, the group of signed quadratic residues modulo an RSA integer N which is a product of safe primes (where a prime p is safe if $(p-1)/2$ is also a prime).

Proof. Let $i \in [n]$ be such that $g_i \neq \mathbf{1}_{\mathbb{G}}$. Note that:

$$\begin{aligned}\Pr_{S\subseteq[n]}\left[\prod_{j\in S} g_j = \mathbf{1}_{\mathbb{G}}\right] &= \Pr_{b_1\ldots,b_n\in\{0,1\}}\left[\prod_{j\in[n]} g_j^{b_j} = \mathbf{1}_{\mathbb{G}}\right] \\ &= \Pr_{b_1\ldots,b_n\in\{0,1\}}\left[g_i^{b_i} = g_{i-1}^{-b_{i-1}}\cdot\ldots\cdot g_1^{-b_1}\cdot g_n^{-b_n}\cdot\ldots\cdot g_{i+1}^{-b_{i+1}}\right] \qquad (4.19)\end{aligned}$$

Fix $b_1,\ldots,b_{i-1},b_{i+1},\ldots,b_n$ and let $h = g_{i-1}^{-b_{i-1}}\cdot\ldots\cdot g_1^{-b_1}\cdot g_n^{-b_n}\cdot\ldots\cdot g_{i+1}^{-b_{i+1}}$. Note that $g_i^1 = g_i \neq \mathbf{1}_{\mathbb{G}} = g_i^0$. Since g_i^1 and g_i^0 differ, it holds that g^{b_i} is equal to the fixed value h with probability at most $1/2$. Hence, the RHS of Equation (4.19) is also at most $1/2$. □

Getting back to our PoE protocol, in order to facilitate the recursive step, we actually present an interactive-proof for the λ-fold repetition $\mathcal{L}^{\lambda}_{\mathbb{G},q}$ of $\mathcal{L}_{\mathbb{G},q}$, where we use the same exponent q^{2^t} across all λ repetitions and where λ is the statistical security parameter. More specifically, consider the language

$$\mathcal{L}^{\lambda}_{\mathbb{G},q} = \left\{ (x,y,t)\in\mathbb{G}^{\lambda}\times\mathbb{G}^{\lambda}\times\mathbb{N} \;:\; \begin{array}{c}\forall i\in[\lambda] \text{ we have } (x_i,y_i,t)\in\mathcal{L}_{\mathbb{G},q},\\ \text{where } x=(x_1,\ldots,x_\lambda),\\ y=(y_1,\ldots,y_\lambda)\end{array} \right\}.$$

Note that $\mathcal{L}_{\mathbb{G},q}$ can be easily reduced to $\mathcal{L}^{\lambda}_{\mathbb{G},q}$ with only a $\mathrm{poly}(\lambda)$ overhead in complexity: to get a proof for (x,y,t) simply invoke the protocol for $\mathcal{L}^{\lambda}_{\mathbb{G},q}$ on (x,y,t) where $x=(x,\ldots,x)$ and $y=(y,\ldots,y)$. Thus, Theorem 4.7.15 follows immediately from the following lemma.

Lemma 4.7.17. *The language $\mathcal{L}^{\lambda}_{\mathbb{G},q}$ has a perfectly correct, statistically sound public-coin interactive-proof with efficiency parameters that are exactly as stated in Theorem 4.7.15.*

Our PoE Protocol

Throughout this section, we will be working with λ-sized vectors. Recall that (in Section 4.7.1), for $g=(g_1,\ldots,g_\lambda)$ and $h=(h_1,\ldots,h_\lambda)\in\mathbb{G}^{\lambda}$, we denoted their coordinate wise product by $g\odot h = (g_1h_1,\ldots,g_\lambda h_\lambda)$. For $g=(g_1,\ldots,g_n)\in\mathbb{G}^{\lambda}$ and $u=(u_1,\ldots,u_n)\in\{0,1\}^{\lambda}$ we denoted by $u\star g$ the subset product of elements in g cor-

$\underline{\mathsf{PoE}(x, y, q, t, \lambda):}$
Input: $x, y \in \mathbb{G}^\lambda$, $q \in \mathbf{N}$, $t \in \mathbf{N}$ and statistical security parameter $\lambda \in \mathbf{N}$.
Claim: $y = x^{q^{2^t}}$.

1. If $t = 0$ then output accept if and only if $y = x^q$.
2. Else
   (a) P computes $\mu = x^{q^{2^{t-1}}} \in \mathbb{G}^\lambda$ and sends μ to V.
   (b) V samples $U = [U_0 \| U_1] \xleftarrow{\$} \{0,1\}^{\lambda \times 2\lambda}$ for $U_0, U_1 \in \{0,1\}^{\lambda \times \lambda}$ and sends U to P.
   (c) P and V compute $x' \in \mathbb{G}^\lambda, y' \in \mathbb{G}^\lambda$ where

$$x' = (U_0 \star \mu) \odot (U_1 \star x)$$
$$y' = (U_0 \star y) \odot (U_1 \star \mu).$$

   (d) P and V recursively call $\mathsf{PoE}(x', y', q, t-1, \lambda)$.

Figure 4.10. Proof-of-Exponentiation Protocol

responding to u (i.e., $u \star g = \prod_{i \in [\lambda]} g_i^{u_i}$). The PoE protocol establishing Lemma 4.7.17 is described in Figure 4.10.

The bounds on communication complexity, running times and space usage specified in Lemma 4.7.17 follow immediately from the description, noting that the prover can re-use its space across different rounds. We next show that completeness and soundness hold, which completes the proof of Lemma 4.7.17.

Proposition 4.7.1. *The* PoE *protocol of Figure 4.10 has perfect completeness.*

Proof. We prove by induction on t. The base case $t = 0$ is immediate. For larger t, suppose that $(x, y, t) \in \mathcal{L}^\lambda_{\mathbb{G},q}$). We show that with probability 1 it holds that $(x', y', t-1) \in \mathcal{L}^\lambda_{\mathbb{G},q}$.

Indeed, using the fact that $x_j^{q^{2^t}} = y_j$ and $\mu_j = x_j^{q^{2^{t-1}}}$ for every $j \in [\lambda]$, we have that for every $i \in [\lambda]$:

$$\begin{aligned}
(x'_i)^{q^{2^{t-1}}} &= \prod_{j \in [\lambda]} \mu_j^{U_0[i,j] \cdot q^{2^{t-1}}} \cdot x_j^{U_1[i,j] \cdot q^{2^{t-1}}} \\
&= \prod_{j \in [\lambda]} x_j^{U_0[i,j] \cdot q^{2^t}} \cdot x_j^{U_1[i,j] \cdot q^{2^{t-1}}} \\
&= \prod_{j \in [\lambda]} y_j^{U_0[i,j]} \cdot \mu_j^{U_1[i,j]}
\end{aligned}$$

$$= y'_i.$$

□

Proposition 4.7.2. *For any $(x, y, t) \notin \mathcal{L}^{\lambda}_{\mathbb{G},q}$ and any (computationally unbounded) malicious prover P^*, the probability that the verifier in Figure 4.10 accepts when interacting with P^* is at most $t/2^{\lambda}$.*

Proposition 4.7.2 follows from the following claim and the union bound (over the t rounds).

Claim 4.7.18. *For any $(x, y, t) \notin \mathcal{L}^{\lambda}_{\mathbb{G},q}$ and prover message μ, it holds that $(x', y', t-1) \notin \mathcal{L}^{\lambda}_{\mathbb{G},q}$ with all but $2^{-\lambda}$ probability (over the choice of verifier message U).*

Proof. Fix $(x, y, t) \notin \mathcal{L}^{\lambda}_{\mathbb{G},q}$ where $x = (x_1, \ldots, x_\lambda) \in \mathbb{G}^\lambda$, $y = (y_1, \ldots, y_\lambda) \in \mathbb{G}^\lambda$. Let $\mu = (\mu_1, \ldots, \mu_\lambda) \in \mathbb{G}^\lambda$ be the prover message and let the verifier message be $U \in \{0,1\}^{\lambda \times 2\lambda}$. Recall that $x' = (x'_1, \ldots, x'_\lambda)$ and $y' = (y'_1, \ldots, y'_\lambda)$ are defined as follows:

$$\begin{aligned} x' &= (U_0 \star \mu) \odot (U_1 \star x) \\ y' &= (U_0 \star y) \odot (U_1 \star \mu). \end{aligned} \tag{4.20}$$

To show the claim, we first show that the probability that $(x'_1)^{q^{2^t/2}} = y'_1$ is at most $1/2$. Since each (x'_i, y'_i) is defined using independent randomness, it follows that the probability that $(x', y', 2^t/2) \in \mathcal{L}^{\lambda}_{\mathbb{G},q}$ is at most $2^{-\lambda}$.

Bounding the probability that $(x'_1)^{q^{2^t/2}} = y'_1$. For every $i \in [\lambda]$, let $e_i = x_i^{q^{2^{t-1}}} \cdot \mu_i^{-1}$ and $f_i = \mu_i^{q^{2^{t-1}}} \cdot y_i^{-1}$. Since $(x, y, t) \notin \mathcal{L}^{\lambda}_{\mathbb{G},q}$, there exists some $i^* \in [\lambda]$ such that either $e_{i^*} \neq 1$ or $f_{i^*} \neq 1$. Thus,

$$\Pr\left[(x'_1)^{q^{T/2}} = y'_1\right] = \Pr_{u,v}\left[\prod_{i \in [\lambda]} e_i^{u_i} f_i^{v_i} = 1\right] \leqslant 1/2, \tag{4.21}$$

where $u = (u_1, \ldots, u_\lambda), v = (v_1, \ldots, v_\lambda) \in \{0,1\}^\lambda$ are sampled uniformly at random, and the inequality follows from Fact 4.7.16. □

4.8 Obtaining Space-Efficient Arguments for NP

We obtain space efficient arguments for any NP relation verifiable by time-T space-S RAM computations by compiling our polynomial commitment scheme with a suitable space-efficient *polynomial interactive oracle proof* (IOP) [35, 76, 226]. Informally, a polynomial IOP is a multi-round interactive PCP such that in each round the verifier sends a message to the prover and the prover responds with a proof oracle that the verifier can query via random access, with the additional property that the proof oracle is a polynomial.

We begin by giving a detailed overview the construction of our polynomial IOP, after which we discuss how to use this polynomial IOP with our polynomial commitment schemes to obtain time- and space-efficient arguments for NP. Towards the first goal, we shall prove the following theorem.

Theorem 4.8.1. *There exists a public-coin polynomial* IOP *over a channel which encodes prover messages as multi-linear extensions for* NP *relations verifiable by a time-T space-S random access machine M such that if $y = M(x; w)$ then*

1. *The* IOP *has perfect completeness and statistical soundness, and has $O(\log(T))$ rounds;*
2. *The prover runs in time $T \cdot \text{polylog}(T)$ and space $S \cdot \text{polylog}(T)$ (not including the space required for the oracle) when given input-witness pair $(x; w)$ for M, sends a single polynomial oracle in the first round, and has $\text{polylog}(T)$ communication in all subsequent rounds; and*
3. *The verifier runs in time $(|x| + |y|) \cdot \text{polylog}(T)$, space $\text{polylog}(T)$, and has query complexity 3.*

To begin, formally define Random Access Machines.

Definition 4.8.1 (Random Access Machine)**.** *A* (non-deterministic) Random Access Machine (RAM) *is a tuple $M = \langle k, r, \mathbb{A}, \mathbb{L} \rangle$ where $\ell \in \mathbb{N}$ is the register size, $r \in \mathbb{N}$ is the number of registers, $\mathbb{A} \subseteq \{f \colon \{0,1\}^\ell \times \{0,1\}^\ell \to \{0,1\}^\ell\}$ is the arithmetic unit, and $\mathbb{L} = (I_0, \ldots, I_m)$, where $m \in [2^\ell]$ and each I_j is an instruction, is the code. For any input $x \in \{0,1\}^n$ and*

witness $w \in \{0,1\}^*$*, a RAM* M *runs in time* $T(n)$ *and space* $S(n)$ *if* $M(x;w)$ *halts after executing at most* $T(n)$ *instructions and uses at most* $S(n)$ *space.*

We let $\mathcal{R}_{RAM}$ *denote the set of all tuples* $(M,x,y,T,S;w)$ *such that* M *is a RAM and* $M(x;w)$ *outputs* y *in time* T *and space* S*. We define the language* $\mathcal{L}_{RAM}$ *as*

$$\mathcal{L}_{RAM} = \{(M,x,y,T,S) \mid \exists w\colon (M,x,y,T,S;w) \in \mathcal{R}_{RAM}\}.$$

Our construction is in fact a polynomial IOP for the NP relation $\mathcal{R}_{\text{RAM}}$. At a high-level, our IOP construction reduces checking membership of a $\mathcal{R}_{\text{RAM}}$ instance to performing the classical sum-check protocol [192, 236] of some appropriately constructed polynomial. We proceed in two parts. First we reduce an $\mathcal{R}_{\text{RAM}}$ instance into a circuit satisfiability instance for some appropriate circuit. We then reduce the circuit satisfiability instance to a polynomial statement compatible with the sum-check protocol. Our construction is inspired from previous approaches [33, 64, 95, 131, 231, 248, 255] re-imagined in the language of IOPs.

4.8.1 RAMs to Circuits.

Let $(M,x,y,T,S;w) \in \mathcal{R}_{\text{RAM}}$. The first step in the IOP is for the prover to compile the RAM M into an appropriate (non-deterministic) arithmetic circuit over some appropriate finite field $\mathbb{F}$.

For finite field $\mathbb{F}$ and arithmetic circuit $C\colon \mathbb{F}^n \to \mathbb{F}^k$, we let $|C|$ denote the size of the circuit.. We assume a canonical ordering on the gates of C (known by both the prover and verifier), and label every gate in C with unique $a \in \{0,1\}^s$ for $s = \lceil \log |C| \rceil$. Without loss of generality we assume the first n input gates of C correspond to the s-bit representation of the integers $\{1,\ldots,n\}$.

Definition 4.8.2 (Circuit Transcript)**.** *A* transcript *for arithmetic circuit* C *with input* x *and output* y *is an assignment* $W\colon \{0,1\}^s \to \mathbb{F}$ *of values to the circuit gates, where* $s = \lceil \log |C| \rceil$*. We say that a transcript* W *is* correct *for* (C,x,y) *if* $W(a) = x_a$ *for all input gates* a*,* $W(a) = y_i$ *if* a *is the* i*-th output gate, and for every* $a,b,c \in \{0,1\}^s$ *such that* b *and* c *are parents of* a*,* $W(a) = W(b) + W(c)$ *if* a *is an add gate and* $W(a) = W(b) \cdot W(c)$ *if* a *is a*

multiplication gate. Given a tuple (C, x, y)*, the problem of determining whether there exists a correct transcript* W *for* (C, x, y) *is referred to as the* non-deterministic circuit evaluation problem.

We use the RAM to circuit transformation of Blumberg et al. [64].

Lemma 4.8.2 ((Non-Deterministic) RAM to Circuit [64, Lemma 4.2])**.** *For* $(M, x, y, T, S; w) \in \mathcal{R}_{RAM}$*,* M *can be transformed into an equivalent (non-deterministic) arithmetic circuit* C_M *over a finite field* $\mathbb{F}$ *of size* $\text{polylog}(T)$ *with the following properties:*

1. C_M *has size* $T \cdot \text{polylog}(T)$*.*
2. *An (input, witness) pair* $(x; w)$ *such that* $(M, x, y, T, S; w) \in \mathcal{R}_{RAM}$ *can be mapped to a correct transcript* W *for* C_M *in time* $T \cdot \text{polylog}(T)$ *and space* $S \cdot \text{polylog}(T)$ *such that* $|W| = 2^s$ *for some* $s = O(\log |C_M|)$*. Furthermore,* w *is a substring of the transcript* W*, and any correct* W' *for* C_M *possesses a witness* w' *certifying* $(M, x, y, T, S) \in \mathcal{L}_{RAM}$ *as a substring.*
3. C_M *can be evaluated "gate-by-gate" in time* $T \cdot \text{polylog}(T)$ *and space* $S \cdot \text{polylog}(T)$*.*

4.8.2 Circuits to Polynomials

Consider $(M, x, y, T, S; w) \in \mathcal{R}_{\text{RAM}}$ and let C_M be the arithmetic circuit given by Lemma 4.8.2, and let $s = \lceil \log |C_M| \rceil$. Let W be a correct transcript for (C_M, x, y). We reduce showing W is a correct transcript for (C_M, x, y) to showing that a polynomial we construct over $\mathbb{F}$ is the 0-polynomial. In particular, the constructed polynomial is the 0-polynomial if and only if W is a correct transcript. Let $\widetilde{W}$ be the multi-linear extension of a transcript W.

We associate three *wiring predicates* $\mathsf{add}, \mathsf{mult}, \mathsf{io} \colon \{0,1\}^{3s} \to \{0,1\}$ with C_M such that the following hold. For all $a, b, c \in \{0,1\}^s$, $\mathsf{add}(a, b, c) = 1$ if and only if a is an addition gate with parent gates b and c (with $\mathsf{mult}(a, b, c)$ being defined analogously), and $\mathsf{io}(a, b, c) = 1$ if and only b and c are parents of a and a is an output gate or a (non-auxiliary) input gate. We also define $I_{x,y} \colon \{0,1\}^s \to \mathbb{F}$ for $a \in \{0,1\}^s$ as $I_{x,y}(a) = x_a$ if a is an input gate, $I_{x,y}(a) = y_i$ if a is the i-th output gate, and $I_{x,y}(a) = 0$ otherwise.

Let $\widetilde{I}_{x,y}, \widetilde{\mathsf{io}}$ be the multi-linear extensions of $I_{x,y}, \mathsf{io}$, and let $\widetilde{\mathsf{add}}, \widetilde{\mathsf{mult}}$ be some degree-3 extensions of $\mathsf{add}, \mathsf{mult}$. Define polynomials $\widetilde{G}_{x,y}\colon \mathbb{F}^{3s} \to \mathbb{F}$ and $F_{x,y}\colon \mathbb{F}^{3s} \to \mathbb{F}$ as

$$\begin{aligned}\widetilde{G}_{x,y}(\zeta^{(1)},\zeta^{(2)},\zeta^{(3)}) &= \widetilde{\mathsf{io}}(\zeta^{(1)},\zeta^{(2)},\zeta^{(3)}) \cdot (\widetilde{I}_{x,y}(\zeta^{(1)}) - \widetilde{W}(\zeta^{(1)})) \\ &+ \widetilde{\mathsf{add}}(\zeta^{(1)},\zeta^{(2)},\zeta^{(3)}) \cdot (\widetilde{W}(\zeta^{(1)}) - \widetilde{W}(\zeta^{(2)}) - \widetilde{W}(\zeta^{(3)})) \\ &+ \widetilde{\mathsf{mult}}(\zeta^{(1)},\zeta^{(2)},\zeta^{(3)}) \cdot (\widetilde{W}(\zeta^{(1)}) - \widetilde{W}(\zeta^{(2)}) \cdot \widetilde{W}(\zeta^{(3)}))\ ,\end{aligned} \tag{4.22}$$

and

$$F_{x,y}(\boldsymbol{X}) = \sum_{c\in\{0,1\}^{3s}} \widetilde{G}_{x,y}(c) \cdot \prod_{i=1}^{3s} \beta(c_i, X_i) = \sum_{c\in\{0,1\}^{3s}} \widetilde{G}_{x,y}(c) \cdot g(c, \boldsymbol{X})\ . \tag{4.23}$$

Lemma 4.8.3. *The polynomial $F_{x,y}$ is the 0-polynomial if and only if $\widetilde{G}_{x,y}(c) = 0$ for all $c \in \{0,1\}^{3s}$ if and only if $\widetilde{W}$ is a multi-linear extension of a correct transcript W of C_M. Further, there exist degree-3 extensions of $\widetilde{\mathsf{add}}$ and $\widetilde{\mathsf{mult}}$ such that $\widetilde{\mathsf{add}}$ and $\widetilde{\mathsf{mult}}$ can be evaluated at any point in* polylog(T) *time without explicit access to C_M.*

Proof. We first note that the definition of $\widetilde{G}_{x,y}$ immediately gives us that $\widetilde{G}_{x,y}|_{\{0,1\}^{3s}} \equiv 0$ if and only if $\widetilde{W}$ is a multi-linear extension of a correct transcript W of C_M [64, 131, 231, 255]. Next, we note that the polynomial $F_{x,y}$ is defined as the multi-linear extension of the sequence $(\widetilde{G}_{x,y}(c)\colon c \in \{0,1\}^{3s})$. In particular, $F_{x,y}(c) = \widetilde{G}_{x,y}(c)$ for all $c \in \{0,1\}^{3s}$. This directly implies that $F_{x,y}$ is the 0-polynomial if and only if $\widetilde{G}_{x,y}(c) = 0$ for all $c \in \{0,1\}^{3s}$. To finish the proof, we note that existence of degree-3 $\widetilde{\mathsf{add}}$ and $\widetilde{\mathsf{mult}}$ with the desired properties follows directly from [64, Theorem 4.1 and Lemma 4.2]. □

Finally, for any $\boldsymbol{\tau} \in \mathbb{F}^{3s}$, we define the polynomial $h_{\boldsymbol{\tau}}(\zeta) := \widetilde{G}_{x,y}(\zeta) \cdot g(\zeta, \boldsymbol{\tau})$. By Lemma 4.8.3 the satisfiability of the circuit C_M is reduced to checking if $F_{x,y}$ is the 0-polynomial. In particular by Schwartz-Zippel a verifier is convinced that $F_{x,y}$ is the 0-polynomial if $F_{x,y}(\boldsymbol{\tau}) = 0$ for $\boldsymbol{\tau} \xleftarrow{\$} \mathbb{F}^{3s}$. However, a verifier would perform $O(|C_M|^3) =$

$T^3 \cdot \text{polylog}(T)$ operations to compute $F_{x,y}(\boldsymbol{\tau})$, which gives a non-succinct verifier. Instead, checking $F_{x,y}(\boldsymbol{\tau}) = 0$ is offloaded to a prover via a sum-check protocol for the statement

$$0 = \sum_{c \in \{0,1\}^{3s}} h_{\boldsymbol{\tau}}(c) = F_{x,y}(\boldsymbol{\tau}).$$

4.8.3 Polynomial IOP Construction

We present our polynomial IOP construction in Figure 4.11, which we call PIOP. The protocol PIOP takes as input $(M, x, y, T, S; w)$ and the prover compiles it into circuit instance (C_M, x, y) via the reduction guaranteed by Lemma 4.8.2. The prover next sends an oracle to the multi-linear extension of the transcript of (C_M, x, y). The prover compiles C_M into a suitable polynomial $F_{x,y}$ given by Lemma 4.8.3, receives $\boldsymbol{\tau} \xleftarrow{\$} \mathbb{F}^{3s}$ from the verifier, and engages in a sum-check with the verifier for the statement $\sum_c h_{\boldsymbol{\tau}}(c) = F_{x,y}(\boldsymbol{\tau}) = 0$. Finally the verifier uses the challenges $\alpha \in \mathbb{F}^{3s}$ given by the sum-check to query the oracle at 3 points and evaluates polynomial $h_{\boldsymbol{\tau}}(\alpha)$ locally and compares it to the value output by the sum-check.

We now argue that PIOP satisfies Theorem 4.8.1. Perfect completeness follows directly from the protocol description, and the round complexity follows since $s = O(\log |C_M|) = \text{polylog}(T)$. To finish proving the theorem, we show the efficiency and soundness of the protocol.

Verifier Efficiency

The verifier samples $O(\log(T))$ random field elements $\boldsymbol{\tau}$ and α_j for $j \in \{1, \ldots, 3s\}$. During each round of the sum-check the verifier evaluates the polynomial $h^{(j)}_{\boldsymbol{\tau}}(X_j)$ at 3 points $\{0, 1, \alpha_j\} \subset \mathbb{F}$. Since $h_{\boldsymbol{\tau}}$ has individual degree at most 6, each univariate $h^{(j)}_{\boldsymbol{\tau}}$ has degree at most 6, giving $O(1)$ multiplications to evaluate $h^{(j)}_{\boldsymbol{\tau}}$. This gives that the verifier has complexity $\text{polylog}(T)$ multiplications and $\text{polylog}(T)$ communication during the sum-check phase. Next the verifier queries the oracle $\widetilde{W}$ at 3 points.As a last step, the verifier uses the 3 oracle queries to compute $h_{\boldsymbol{\tau}}(\alpha)$, which by Lemma 4.8.3 takes time $(|x| + |y|) \cdot \text{polylog}(T)$ and is computable without explicit access to C_M.

$\mathsf{PIOP}(M, x, T, S; w)$

Prover Input : RAM M, RAM input x, time parameter T, space parameter S, witness w

Verifier Input : RAM M, RAM input x, time parameter T, space parameter S

Output : Accept or Reject

P compiles circuit C_M and transcript W via the reduction of [64].

P sends W to V as an oracle, which is encoded as $\widetilde{W}$ by the channel.

V samples $\boldsymbol{\tau} \xleftarrow{\$} \mathbb{F}^{3s}$ and sends $\boldsymbol{\tau}$ to P.

P computes polynomial $h_{\boldsymbol{\tau}}$ and sets $\gamma \leftarrow 0$. P sends γ to V.

V sets $\gamma' \leftarrow \gamma$.

foreach $j \in [3s]$ **do** ; /* sum-check */

P sends sends $h_{\boldsymbol{\tau}}^{(j)}(X_j)$ to V, where

$$h_{\boldsymbol{\tau}}^{(j)}(X_j) \leftarrow \sum_{c' \in \{0,1\}^{3s-j}} h_{\boldsymbol{\tau}}(\alpha_1, \ldots, \alpha_{j-1}, X_j, c').$$

V checks $\gamma' \stackrel{?}{=} h_{\boldsymbol{\tau}}^{(j)}(0) + h_{\boldsymbol{\tau}}^{(j)}(1)$, rejecting if equality doesn't hold.

V samples $\alpha_j \xleftarrow{\$} \mathbb{F}$ and sets $\gamma' \leftarrow h_{\boldsymbol{\tau}}^{(j)}(\alpha_j)$.

if $j < 3s$ **then**

V sends α_j to P.

V queries oracle $\widetilde{W}$ and obtains $\gamma_i \leftarrow \widetilde{W}(\alpha^{(i)})$ for each $i \in [3]$, where $\alpha^{(i)} \leftarrow (\alpha_{i\cdot 1}, \ldots, \alpha_{i\cdot s})$.

V computes $h_{\boldsymbol{\tau}}(\alpha)$ using oracle queries γ_i and accepts if and only if $\gamma' = h_{\boldsymbol{\tau}}(\alpha)$.

Figure 4.11. Formal description of our polynomial interactive oracle proof for time-T space-S RAM computations.

During any round of the sum-check, the verifier stores $O(1)$ field elements for the description of $h_{\boldsymbol{\tau}}^{(j)}$ and stores all challenges $\boldsymbol{\tau} \in \mathbb{F}^{3s}$ and $\alpha \in \mathbb{F}^{3s}$, which is polylog$(T)$ field elements of storage. Computing extensions $\overline{\mathsf{add}}$ and $\overline{\mathsf{mult}}$ can be done in polylog(T) time, so they can be computed using at most polylog(T) space. This gives a verifier with space complexity of polylog(T).

Prover Efficiency

We examine the complexity of the prover in PIOP after sending the oracle $\widetilde{W}$, an oracle to the multi-linear extension of a correct transcript W for $(C_M, x, y, T, S; w)$. The prover receives

verifier challenge $\boldsymbol{\tau}$ and now must run the sum-check with polynomial $h_{\boldsymbol{\tau}}(\zeta)$ of Lemma 4.8.3. We leverage the following lemma to allow the prover to perform this computation efficiently in each round.

Lemma 4.8.4 ([64, Theorem 4.1, Lemma 4.2])**.** *Let $(M, x, y, T, S; w) \in \mathcal{R}_{RAM}$ and let C_M be the equivalent arithmetic circuit given by Lemma 4.8.2. Then given (M, x, y, T, S, w), one can run in time $T \cdot \text{polylog}(T)$ and space $S \cdot \text{polylog}(T)$ to compute sum-check messages for the polynomial $h_{\boldsymbol{\tau}}(\boldsymbol{Y})$.*

In particular, Lemma 4.8.4 implies that the prover's computation of the polynomial $h_{\boldsymbol{\tau}}^{(j)}(X_j)$ in each round of the sum-check can be done in $T \cdot \text{polylog}(T)$ time and $S \cdot \text{polylog}(T)$ space. Note also that each polynomial $h_{\boldsymbol{\tau}}^{(j)}(X_j)$ is a degree at most 6 polynomial, and therefore uses $O(1)$ space. Finally, the prover also stores verifier challenges α_j for each $j \in \{1, \ldots, 3s-1\}$, which requires $O(\text{polylog}(T))$ space. Since $s = O(\log |C_M|)$ and $|C_M| = T \cdot \text{polylog}(T)$, we have that the total prover time is $T \cdot \text{polylog}(T)$ and total space is $S \cdot \text{polylog}(T)$.

Soundness

The soundness of PIOP follows from the soundness of the sum-check protocol.

Lemma 4.8.5 (Sum-check Soundness [192, 236])**.** *For $\gamma \in \mathbb{F}$, $v, d \in \mathbb{N}$, let $\mathcal{L}_{\gamma,v,d}$ be the language of all v-variate polynomials f of individual degree at most d such that $\gamma = \sum_{c \in \{0,1\}^v} f(c)$. Then the sum-check protocol is an interactive proof system for $\mathcal{L}_{\gamma,v,d}$ with perfect completeness and soundness error $\varepsilon_{\mathsf{sc}} \leqslant dv/|\mathbb{F}|$, where the verifier is given oracle access to the function f.*

We now show the soundness of PIOP.

Proposition 4.8.1. *Let V be the verifier of PIOP. For every $\mathbb{x} = (M, x, y, T, S) \notin \mathcal{L}_{RAM}$ and every P^*, over the randomness of V we have that $\langle P^*, V(\mathbb{x}) \rangle = 1$ with probability at most $\frac{36s}{|\mathbb{F}|}$.*

Proof. Let P^* be an arbitrary prover. Let $(M, x, y, T, S) \notin \mathcal{L}_{\text{RAM}}$ be the input to both verifier V and P^*. By assumption, there does not exist a correct transcript W for the circuit C_M. Let W^* be the polynomial oracle sent by P^*.

By Lemma 4.8.3, the polynomial $F_{x,y}$ is the 0-polynomial if and only if W^* is a multi-linear extension of a correct transcript, so by assumption $F_{x,y}$ is not the 0-polynomial and has individual degree at most 6. For a verifier to be convinced that W^* is a correct transcript, it suffices for the verifier to sample $\boldsymbol{\tau} \xleftarrow{\$} \mathbb{F}^{3s}$ and check $F_{x,y}(\boldsymbol{\tau}) \stackrel{?}{=} 0$. Since $F_{x,y}$ is a polynomial of individual degree at most 6, by Schwartz-Zippel we have

$$\Pr_{\boldsymbol{\tau} \xleftarrow{\$} \mathbb{F}^{3s}} [F_{x,y}(\boldsymbol{\tau}) = 0 | F_{x,y} \not\equiv 0] \leqslant \frac{6 \cdot 3s}{|\mathbb{F}|}.$$

So after receiving oracle W^* the verifier samples and sends $\boldsymbol{\tau} \xleftarrow{\$} \mathbb{F}^{3s}$ to P^*.

Now P^* and V engage in a sum-check protocol for the statement

$$0 = \sum_{c \in \{0,1\}^{3s}} h_{\boldsymbol{\tau}}(c) = F_{x,y}(\boldsymbol{\tau}). \tag{4.24}$$

Conditioning on $F_{x,y}(\boldsymbol{\tau}) \neq 0$, the soundness now reduces to the soundness of the sum-check. In particular, P^* must convinced V that Equation (4.24) holds, but $F_{x,y}(\boldsymbol{\tau}) \neq 0$. In this case the probability P^* succeeds is at most $(3s \cdot 6)/|\mathbb{F}|$ by Lemma 4.8.5. Since $F_{x,y}$ is not the 0-polynomial we have that

$$\Pr[\langle P^*, V(\mathbb{x}) \rangle = 1] \leqslant \Pr_{\boldsymbol{\tau}}[F_{x,y}(\boldsymbol{\tau}) = 0] + \Pr[P^* \text{ breaks sum-check}] \leqslant \frac{36s}{|\mathbb{F}|}.$$

□

4.8.4 Time- and Space-Efficient Arguments for RAM

We obtain space-efficient arguments $\langle P_{\text{arg}}, V_{\text{arg}} \rangle$ for NP relations that can be verified by time-T space-S RAMs by composing the polynomial commitment schemes of Theorems 4.6.4 and 4.7.7 and the polynomial IOP of Figure 4.11. Specifically, the prover P_{arg} and V_{arg} runs the prover and the verifier of the underlying PIOP except two changes: (1) P_{arg} (Line 2 of Figure 4.11) instead provides V_{arg} with a commitment to the multi-linear extension of the circuit transcript W using either Theorem 4.6.4 or Theorem 4.7.7. Here P_{arg} crucially relies on streaming access to W to compute the commitment in small-space using Com. (2) P_{arg} and

V_{arg} run the appropriate Eval protocol in place of all verifier queries to the oracle $\widetilde{W}$ (Line 13 of Figure 4.11). We state a formal theorem to capture using both polynomial commitment schemes of Theorems 4.6.4 and 4.7.7.

Theorem 4.8.6 (Small-space Arguments for RAMs)**.** *In the random oracle model, assuming the hardness of discrete-log in obliviously sampelable prime-order groups, there exists a public-coin interactive argument for* NP *relations verifiable by time-T space-S random access machines with the following complexity.*

1. *The protocol has perfect completeness, has* $O(\log(T))$ *rounds and* $\text{polylog}(T)$ *communication, and has witness-extended emulation.*

2. *The prover runs in time* $T \cdot \text{polylog}(T)$ *and space* $S \cdot \text{polylog}(T)$ *given input-witness pair* $(x; w)$ *for* M*; and*

3. *The verifier runs in time* $T \cdot \text{polylog}(T)$ *and space* $\text{polylog}(T)$*.*

Alternatively, assuming the existence of a group for which the hidden order assumption holds, there exists a public-coin interactive argument for NP *relations verifiable by time-T space-S random access machines with the following complexity.*

1. *The protocol has perfect completeness, has* $O(\log(T))$ *rounds and* $\text{polylog}(T)$ *communication, and has (statistical) witness-extended emulation.*

2. *The prover runs in time* $T \cdot \text{polylog}(T)$ *and space* $S \cdot \text{polylog}(T)$ *given input-witness pair* $(x; w)$ *for* M*; and*

3. *The verifier runs in time and space* $\text{polylog}(T)$*.*

Proof. We compose our commitment schemes of Theorem 4.6.4 (for the discrete-log setting) or of Theorem 4.7.7 (for the hidden order setting) with the Polynomial IOP of Theorem 4.8.1. The algorithm Setup is identical to that of the chosen commitment scheme. The protocol is identical to the protocol PIOP except for the following changes. First, instead of providing the verifier with an oracle $\widetilde{W}$, the prover instead sends a commitment to $\widetilde{W}$ using the appropriate

commitment scheme, noting that W uniquely defines the multi-linear extension $\widetilde{W}$. Second, the 3 verifier queries to the oracle $\widetilde{W}$ are replaced with 3 invocations of the protocol Eval^{ρ}.

By Lemma 4.8.2, the transcript W is computable in a streaming manner in time $T \cdot \text{polylog}(T)$ and space $S \cdot \text{polylog}(T)$, so computing the commitment requires $S \cdot \text{polylog}(T)$ space for either polynomial commitment scheme. Replacing the oracle queries to $\widetilde{W}$ with 3 invocations of the Eval protocol requires $T \cdot \text{polylog}(T)$ time (in the discrete-log setting) or $\text{polylog}(T)$ time (in the hidden order setting) from the verifier, since $|W| = T \cdot \text{polylog}(T)$.

Let P^* be a cheating prover for our argument system. The we construct an emulator $\mathcal{E}$ for our scheme which does the following.

1. Runs P^* until the first Eval query is generated, yielding partial transcript t_1.

2. Run the emulator $\mathcal{E}^{\rho}_{\mathsf{pc}}$ of the polynomial commitment scheme, yielding extracted witness $W \colon \{0,1\}^s \to \mathbb{F}$ and some transcript t_2. We note that if $\mathcal{E}^{\rho}_{\mathsf{pc}}$ aborts, then $\mathcal{E}$ aborts.

3. Finish the interaction with P^*, yielding transcript t_3. Let $\mathsf{tr} = t_1 \circ t_2 \circ t_3$.

4. If W is not consistent with (C_M, x, y) or t is rejecting, output $(\mathsf{tr}, 0)$. If W is consistent with (C_M, x, y), extract witness w from W (since w is a substring of W by Lemma 4.8.2). If tr is accepting and $M(x; w) = y$, output (tr, w). Else output $\perp$.

Note that $\mathcal{E}$ as run above runs in expected polynomial time if $\mathcal{E}^{\rho}_{\mathsf{pc}}$ runs in expected polynomial time. Fix polynomial time adversary A. We need to show that

$$\Pr\left[\mathsf{tr} \leftarrow \langle P^*, V\rangle \colon A(\mathsf{tr}) = 1\right] \approx_{\mathsf{negl}(\lambda)} \Pr\left[(\mathsf{tr}, w) \leftarrow \mathcal{E} \colon \begin{array}{c} A(\mathsf{tr}) = 1 \wedge \\ \mathsf{tr} \text{ is accepting} \implies M(x; w) = y \end{array}\right], \tag{4.25}$$

where λ is the security parameter.

If $\mathcal{E}$ outputs a pair (tr, w), the transcript tr is a transcript produced by an honest verifier interacting with P^*. In this case, the probability that $A(\mathsf{tr}) = 1$ when $\mathcal{E}$ does not abort differs from the probability that $A(\mathsf{tr}) = 1$ for tr output by $\langle P^*, V\rangle$ by at most $\mathsf{negl}(\lambda)$. This

is due to the negligible error of the emulator $\mathcal{E}^{\rho}_{\mathsf{pc}}$. We now turn to show that the probability $\mathcal{E}$ aborts is negligible.

Let E be the event that $\mathcal{E}$ aborts. In particular, $\mathcal{E}$ aborts if $\mathcal{E}^{\rho}_{\mathsf{pc}}$ fails to extract a witness, or if $\mathcal{E}^{\rho}_{\mathsf{pc}}$ extracts a witness (circuit transcript) W and W is a correct transcript for (C_M, x, y) and tr is an accepting transcript but $M(x; w) \neq y$ for witness w that is a substring of W. By the witness-extended emulation property of our polynomial commitment scheme, it holds that $\mathcal{E}^{\rho}_{\mathsf{pc}}$ aborts with negligible probability. Now suppose $\mathcal{E}^{\rho}_{\mathsf{pc}}$ does not abort and that it successfully extracts a witness W. Let E' be the event that W is a correct transcript for (C_M, x, y) and tr is accepting but $M(x; w) \neq y$, conditioned on $\mathcal{E}^{\rho}_{\mathsf{pc}}$ not aborting and successfully extracting witness W. Suppose $\Pr[E'] = \varepsilon$ for some $\varepsilon \in [0, 1]$. We now construct an adversary $\hat{P}$ which breaks soundness of our IOP with probability at least $\varepsilon - \mathsf{negl}(\lambda)$.

We first note the differences between the our argument verifier V and our IOP verifier V_{IOP}. The IOP verifier V_{IOP} receives a polynomial oracle W^* from the prover while V does not receive an oracle to the multi-linear extension of a transcript and instead receives a commitment to the the oracle W^*. Further, V_{IOP} simply queries said oracle to check the final statement of the sum-chekc, while V interacts with P^* to obtain evaluations of this oracle.

We now describe how $\hat{P}$ breaks the soundness of the IOP. $\hat{P}$ simulates the interaction between P^* and V until V makes an eval query. $\hat{P}$ then runs the emulator $\mathcal{E}^{\rho}_{\mathsf{pc}}$ to extract out witness W, rewinding P^* and V as necessary. Once a witness W is extracted, $\hat{P}$ rewinds P^* to the point just after the commitment to W was sent. Now $\hat{P}$ computes the multi-linear extension of W as $\widetilde{W}$ and sends this to V_{IOP}. $\hat{P}$ then forwards all verifier messages to P^* and forwards all messages from P^* to V_{IOP}. Finally V_{IOP} outputs accept or reject after querying the oracle $\widetilde{W}$ and computing a final check.

We note that $\mathcal{E}^{\rho}_{\mathsf{pc}}$ has error probability $\mathsf{negl}(\lambda)$; that is, the emulator may output an incorrect W with negligible probability. Suppose this is not the case and that W is a correct transcript for (C_M, x, y) but the witness w extracted from W is such that $M(x; w) \neq y$. By assumption, we have that P^* convinces V to accept this scenario with probability ε. By our

construction of $\hat{P}$, we also have that in this scenario V_{IOP} accepts with probability ε. So we have that

$$\Pr\left[\langle \hat{P}, V_{\mathsf{IOP}} \rangle = 1\right] \geqslant \varepsilon(1 - \mathsf{negl}(\lambda)) \geqslant \varepsilon - \mathsf{negl}(\lambda).$$

Note that by soundness of the IOP we have that

$$\Pr\left[\langle \hat{P}, V_{\mathsf{IOP}} \rangle = 1\right] \leqslant \frac{36s}{|\mathbb{F}|},$$

where $|\mathbb{F}|$ is exponential in the security parameter and $s = O(\log T)$. Thus we have that

$$\varepsilon \leqslant \frac{36s}{|\mathbb{F}|} + \mathsf{negl}(\lambda) = \mathsf{negl}(\lambda).$$

Therefore $\Pr[E] = \varepsilon \leqslant \mathsf{negl}(\lambda)$ as desired, and the total probability that $\mathcal{E}$ aborts is at most some negligible function of λ. This along with the negligible error of $\mathcal{E}^{\rho}_{\mathsf{pc}}$ gives that Equation (4.25) holds for some function $\mathsf{negl}(\lambda)$, showing witness-extended emulation. □

4.8.5 Obtaining Theorems 4.2.3 and 4.2.4

We discuss how to modify our interactive argument of knowledge from Theorem 4.8.6 to satisfy zero-knowledge, obtaining Theorems 4.2.3 and 4.2.4. We then discuss how to make the resulting zero-knowledge arguments non-interactive via the Fiat-Shamir transformation, obtaining time- and space-efficient zk-SNARKs.

Zero-Knowledge

For obtaining zero-knowledge, we employ the standard technique (due to Ben-Or et al. [29]) of having the prover commit to its messages and then, at the end, prove that it knows valid openings. Since this is a small NP statement, we can afford to use basically any zero-knowledge protocol from the literature. However, since we do not want the round complexity to grow by a $\mathrm{poly}(\lambda)$ factor, we use the constant-round public-coin zero-knowledge argument of Barak [24].

Lemma 4.8.7. *Assume that collision-resistant hash functions exist. Suppose that the relation $R \in \mathbf{NP}$ has a public-coin holographic argument-system with a time T_V verifier. Then it also has a* zero-knowledge *public-coin (holographic) argument-system with only a* $\mathrm{poly}(\lambda, T_V)$ *multiplicative overhead in prover time, prover space, communication complexity and verifier time. The round complexity increases additively by* $O(1)$.

Proof Sketch. Following [29] we modify the protocol so that in every round, rather than having the prover send it's message in the clear, it commits to it using a cryptographic commitment scheme (which can be constructed assuming one-way functions). At the end of the protocol the verifier makes its queries to the input (which, importantly, depend only on its random coin tosses). At this point all that is left to check is an NP statement of the form: do there exist openings for the commitment that would make the verifier accept. Note that this NP statement has instances of size $\leqslant T_V$, witnesses of size $\leqslant T_V$ and can be decided in time $\mathrm{poly}(T_V)$. Thus, we can use a generic zero-knowledge argument (of knowledge) for NP of [24]. □

Remark 4.8.8. *Note that if one is planning to apply the Fiat-Shamir transform on the resulting protocol, then it suffices to prove* honest-verifier *zero-knowledge, and so a more basic approach should suffice.*

We further remark, that it is likely that there are far more practical ways to make our protocol zero-knowledge. For example, by ensuring that the polynomial commitment is hiding. We leave the exploration of this possibility to future work.

Non-Interactivity

We apply the Fiat-Shamir (FS) transform [116] to our zero-knowledge argument of knowledge, thereby obtaining a non-interactive, zero-knowledge argument of knowledge. However, note that it is folklore that applying FS to a t-round public-coin argument of knowledge yields a non-interactive argument of knowledge where the extractor runs in time exponential in t. Since our protocol has $O(\log T)$ rounds our extractor runs in $\mathrm{poly}(T)$-time.

Part II

COMPOSING CRYPTOGRAPHIC PRIMITIVES AND TECHNIQUES WITH CODING THEORY CONSTRUCTIONS

5. COMPILING HAMMING LOCALLY DECODABLE CODES TO INSERTION-DELETION LOCALLY DECODABLE CODES: USING CRYPTOGRAPHIC THINKING TO SIMPLIFY ANALYSIS

A portion of this chapter appears in the 40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2020) [48], available at https://doi.org/10.4230/LIPIcs.FSTTCS.2020.16. A portion of this chapter appears in arXiv [47], available at https://arxiv.org/abs/2103.14122. The article [47] is the full version of the article which appears in the 2021 IEEE International Symposium on Information Theory [46], available at https://doi.org/10.1109/ISIT45174.2021.9518249.

One of the most widespread ways cryptography has been composed with coding theory (and theoretical computer science in general) is through the use of *cryptographic thinking*. Cryptographic thinking (e.g., adversarial thinking, bounds on computational power) has influenced many areas of computer science. For example, the seminal work of Goldwasser, Micali, and Rackoff [132] introduced the complexity class of languages recognizable by interactive proofs in polynomial time, or IP, primarily to study the cryptographic tool of zero-knowledge. The class IP was extensively studied by complexity theorists, leading to the seminal results of Shamir [236] and Lund et al. [192] showing that $\mathsf{IP} = \mathsf{PSPACE}$.

This impact is plainly on display in the context of *locally decodable codes*. A $(\ell, \delta, p, \text{dist})$-locally decodable code (LDC) (Definition 2.7.5) is a coding scheme $C[K, k, q_1, q_2]$ such that the decoder $\mathsf{Dec} : [k] \to \Sigma_1$ is a probabilistic oracle algorithm such that: for any message $x \in \Sigma_1^k$, when the decoder is given oracle access to any $\tilde{y} \in \Sigma_2^*$ such that $\text{dist}(\tilde{y}, \mathsf{Enc}(x)) \leqslant \delta$, the decoder makes at most ℓ queries to its oracle and outputs $x_i \in \Sigma_1$ with probability at least p. Here ℓ is the *locality* (also known as the *query complexity*), δ is the *error-tolerance*, p is the *success probability*, and dist is some normalized distance. In this chapter, we turn our attention to how cryptographic thinking has inspired the construction of LDCs which are resilient against *insertion-deletion errors*: a more powerful corruption model where the codeword is subject to a bounded number of arbitrary symbol insertions and deletions.

5.1 Locally Decodable Codes for Insertion-Deletion Errors

For decades, the focus of locally decodable codes (LDCs) was directed towards giving optimal constructions of LDCs for the *Hamming channel*. That is, locally decodable codes which are resilient to (a bounded number of) worst-case bit flips in the codeword. Such codes have been studied extensively for decades [70, 85–89, 142–145, 148–152, 179, 185, 190, 241], but even here there are still many fascinating and challenging open problems. For example, in the 2-query regime, it is known that the Hadamard code is an optimal 2-query Hamming LDC with exponential block length [173]. However, recently there has been more focus on the construction of LDCs which are resilient to *insertion-deletion errors*. That is, LDCs which can still decode when given a codeword that has been subjected to a bounded number of arbitrary insertions and deletions of symbols. Such LDCs, which we call InsDel LDCs, have been notoriously difficult to construct, with only recent results achieving constructions with some asymptotically optimal parameters [144, 148, 149, 151, 190].

Of interest to us is a compiler result due to Ostrovsky and Paskin-Cherniavsky [215]. They construct a so-called "Hamming-to-InsDel" compiler: this compiler takes as in put any Hamming LDC and outputs an InsDel LDC. This compiler (asymptotically) preserves the rate and error-tolerance of the code, and increases the locality of the code by only a poly-logarithmic factor. The construction utilizes a well-known technique in coding theory known as *code concatenation* (a.k.a. concatenated codes) [119]. Such codes utilize an *outer code* C_{out} and an *inner code* C_{in} and encode a message x first using Enc_{out}, obtaining $y = \mathsf{Enc}_{out}(x)$. The codeword y is then divided into some number of m blocks $y^{(1)}, \ldots, y^{(m)}$ such that $y = y^{(1)} \circ \cdots \circ y^{(m)}$, and each block $y^{(i)}$ is encoded using the inner code to obtain $Y^{(i)} = \mathsf{Enc}_{in}(y^{(i)})$. The final codeword given as output is $Y = Y^{(1)} \circ \cdots Y^{(m)}$.

The Hamming-to-InsDel compiler works in much the same way: the compiler takes as input a Hamming LDC and uses this code as the outer code. The inner code is fixed to be the well-known Schulman-Zuckerman InsDel code [230], which we refer to as the SZ code. The SZ code is an asymptotically optimal code for insertions-deletions errors, but it is not a locally decodable code. Thus the compiler splits the encoded LDC codeword into logarithmic-sized blocks and encodes these blocks with the SZ code. Intuitively, this is one reason for the

poly-logarithmic increase in locality since the local decoder needs to decode entire blocks of SZ codewords, which are of logarithmic size. The remaining increase in locality is due to a *noisy binary search* procedure. Informally, this noisy binary search procedure allows one to search for an element of a sorted list where some constant fraction of the original entries may be modified arbitrarily (e.g., they are arbitrarily modified, swapped, deleted, inserted, etc.). This procedure is used to recover corrupt insertion-deletion blocks in order to recover the original SZ codeword blocks, when then are decoded using the SZ decoder, followed by the LDC decoder to obtain the desired symbol.

5.2 Our Results

We revisit the compiler result of [215] and provide an alternative proof using different combinatorial techniques. Our combinatorial techniques borrow from analysis tools used widely in the design and analysis of *memory-hard functions*—important cryptographic primitives that, intuitively, are functions that require a large amount of space to compute [4–10, 39, 59, 62, 83, 117].

Theorem 5.2.1 ([48])**.** *Let C be a q-query LDC with encoding function $\mathsf{Enc}\colon \Sigma^k \to \Sigma^n$ that can decode from δ-fraction of Hamming errors. Then there exists a binary $q \cdot \mathrm{polylog}(n)$-query InsDel LDC with codeword length $\Theta(n \cdot \log(|\Sigma|))$ that can decode from $\Theta(\delta)$-fraction of insertion-deletion errors.*

In this work we focus on LDCs but note that Theorem 5.2.1 extends to the setting of *locally correctable codes* [37] as well. These codes are a variant of LDCs where the decoder is locally corrects every entry of the encoded message, instead of decoding entries of the message itself. Similar to the results of [215], our result does not have implications in the constant-locality regime.

Classical Hamming LDC constructions fall into three locality-complexity regimes: constant locality, poly-logarithmic (in k) locality, and sub-polynomial (but super logarithmic) locality. In the constant locality regime, the best known constructions are via matching-vector codes which have block length $\exp(\exp(\sqrt{\log(k) \cdot \mathrm{loglog}(k)}))$ [108, 109, 265], while the best known lower bounds are only quadratic (e.g., $K = \Omega(k^2)$) [260, 261]. For poly-logarithmic locality,

Reed-Muller codes have locality $\log^c(k)$ for some constant $c > 0$ and have rate $k^{1/(c-1)+o(1)}$ [264]. Finally, in the sub-polynomial, super-logarithmic locality regime, there exists Hamming LDCs with constant-rate [182, 183].

Theorem 5.2.1 in conjunction with the above results yield the following (asymptotic) results.

Corollary 5.2.2 ([48])**.** *There exist InsDel LDCs of message length k with rate $o(k)$, locality* $\text{polylog}(k)$*, and constant error-rate.*

Corollary 5.2.3 ([48])**.** *There exist InsDel LDCs of message length k with constant rate, locality* $\log(k)^{O(\log\log(k))}$*, and constant error-rate.*

5.2.1 Extension to Private and Resource-Bounded Locally Decodable Codes

We additionally show that the compiler of Theorem 5.2.1 extends to both the private Hamming LDC and resource-bounded Hamming LDC setting. In Chapter 6, we discuss both private and resource-bounded LDCs in more detail, but give an overview here. Ostrovsky, Pandey, and Sahai [214] introduce the notion of *private locally decodable codes*: LDCs which are secure against computationally bounded channels and equip both the encoding and decoding algorithm with a shared *secret key* generated via standard cryptographic techniques (e.g., one-way functions, pseudo-random generators), or with enough uniform bits (in which case security holds against information-theoretic adversaries); see Definition 5.5.1 for the formal definition.Key to their construction is exactly that the key *remains a secret from the channel*. The secret key assumption yields a construction of a private LDC with constant rate and slightly super-logarithmic query complexity $\omega(\log(\lambda))$, and success probability $1 - \mathsf{negl}(\lambda)$, where λ is the security parameter. Blocki, Kulkarni, and Zhou [60] introduce and study *resource-bounded locally decodable codes*. In such LDC assumptions, the channel is assumed to belong to some algorithm class $\mathbb{C}$ which is resource-constrained in some way; e.g., the channel is a low-depth circuit, the channel is PPT, or the channel is memory-constrained. This is a generalization of Lipton's computational channel model [188] and (arguably) captures many channels encountered in nature. Informally, a code $C[K, k, q_1, q_2]$ is a $(\ell, \delta, p, \varepsilon, \delta, \mathbb{C})$-*LDC against class* $\mathbb{C}$ if on any input $i \in [k]$, the decoder makes at most ℓ queries to its (possibly

δ-corrupt) codeword oracle and outputs the i^{th} symbol with probability at least p. The $(\mathbb{C}, \varepsilon)$-security property informally states that any adversary cannot produce a δ-corrupt codeword oracle that causes the decoder to output some symbol x_i of the original encoded message with probability less than p (see Definition 5.5.2 for formal definition).

We show that compiling a private or resource-bounded Hamming LDC with our compiler yields a private or resource-bounded InsDel LDC with comparable parameters.

Theorem 5.2.4 ([46]). *Let $C[K, k, \lambda, q_1, q_2]$ be a $(\ell, \delta, p, \varepsilon)$-one time private Hamming LDC for constant $\delta > 0$. Then there exists a binary code $C_{\mathsf{f}}[n, k, \lambda, q_1, 2]$ that is a $(\ell_{\mathsf{f}}, \delta_{\mathsf{f}}, p_{\mathsf{f}}, \varepsilon_{\mathsf{f}})$-one time private InsDel LDC, where $\ell_{\mathsf{f}} = \ell \cdot O(\log^4(n))$, $\delta_{\mathsf{f}} = \Theta(\delta)$, $p_{\mathsf{f}} < p$, $\varepsilon_{\mathsf{f}} = \varepsilon/(1 - (p_{\mathsf{f}}/p) - (\vartheta_1(n)/p) - \vartheta_2(n))$, and $n = \Theta(K \cdot \log(q_2))$. Here ϑ_1, ϑ_2 are fixed negligible functions.*

Note that Ostrovsky, Pandey, and Sahai [214] give a construction of a one-time private Hamming LDC with $K = \Theta(k)$, $\delta = \Theta(1)$, $\ell = \log^{1+\epsilon}(\lambda)$ for any constant $\epsilon > 0$, $p = 1 - \mathsf{negl}(\lambda)$, and $\varepsilon = \mathsf{negl}(\lambda)$, where negl are unspecified negligible functions. This gives us the following corollary.

Corollary 5.2.5. *Let $\ell_{\mathsf{f}} := \ell_f(\lambda, n) = O(\log^{1+\epsilon}(\lambda) \cdot \log^4(n))$ for constant $\epsilon > 0$. There exists a binary code $C_{\mathsf{f}}[n, k, \lambda, q_1, 2]$ that is an $(\ell_{\mathsf{f}}, \delta_{\mathsf{f}}, p_{\mathsf{f}}, \varepsilon_{\mathsf{f}})$-one time private InsDel LDC with $n = \Theta(k \cdot \log(q_1))$, $\delta_{\mathsf{f}} = \Theta(1)$, $p_{\mathsf{f}} = \Theta(1)$, and $\varepsilon_{\mathsf{f}} \leqslant \varsigma(\lambda, n)$ for some fixed negligible function ς.*

For the case of resource-bounded InsDel LDCs, if we want the final InsDel LDC to be secure against an algorithm class $\mathbb{C}$, then the original Hamming LDC needs to be secure against a larger class $\overline{\mathbb{C}}(N)$ which we call the *closure of* $\mathbb{C}$ with respect to parameter N. See Section 5.7.3 for more details.

Theorem 5.2.6 ([46]). *Let $\mathbb{C}$ be the class of parallel PPT algorithms running in sequential time T and space S, and let $C[K, k, q_1, q_2]$ be a $(\ell, \delta, p, \varepsilon, \overline{\mathbb{C}}(n))$-Hamming LDC for constant $\rho, p > 0$ and $n = O(K \cdot \log(q_2))$. There exists a binary code $C_{\mathsf{f}}[n, k, q_1, 2]$ that is a $(\ell_{\mathsf{f}}, \delta_{\mathsf{f}}, p_{\mathsf{f}}, \varepsilon_{\mathsf{f}}, \mathbb{C})$-InsDel LDC against class $\mathbb{C}$, where $\ell_{\mathsf{f}} = \ell \cdot O(\log^4(n))$, $\delta_{\mathsf{f}} = \Theta(\delta)$, $p_{\mathsf{f}} < p$, and $\varepsilon_{\mathsf{f}} = \varepsilon/(1 - (p_{\mathsf{f}}/p) - (\vartheta_1(n)/p) - \vartheta_2(n))$. Here ϑ_1, ϑ_2 are fixed negligible functions.*

5.3 Technical Overview

The main technical hurdle in the Hamming-to-InsDel compiler is the implementation of the local decoding algorithm. Unlike traditional Hamming concatenation codes, in the insertion-deletion setting the code blocks of the inner code may be arbitrarily corrupted with insertions, may be moved to another part of the codeword, or may be deleted all together. Thus the key challenge is implementing a noisy block decoding algorithm that is successful with high probability. To build intuition for this decoding procedure, we work with the simpler problem of searching a nearly sorted array.

5.3.1 Searching a Nearly Sorted Array

We consider the following problem. Suppose we are given a nearly sorted array A of n distinct elements. By nearly sorted we mean that there is another sorted array A' such that $A'[i] = A[i]$ on all but n' indices. Given an input x we would like to quickly find x in the original array. In the worst case this would require time at least $\Omega(n')$ so we relax the requirement that we always find x to say that there are at most cn' items that we fail to find x for some constant $c > 0$.

To design our noisy binary search algorithm that meets these requirement we borrow a notion of local goodness used in the design and analysis of depth-robust graphs—a combinatorial object that has found many applications in cryptography [4, 7, 111]. In particular, fixing A and A' (sorted) we say that an index j is corrupted if $A[j] \neq A'[j]$. We say that an index i is θ-locally good if for any $r \geqslant 0$ at most θ fraction of the indices $j \in [i, \ldots, i + r]$ are corrupted and at most θ fraction of the indices in $[i - r, i]$ are corrupted. If at most n' indices are corrupted then one can prove that at least $n - 2n'/\theta$ indices are θ-locally good [111].

As long as the constant θ is suitably small we can design an efficient randomize search procedure which, with high probability, correctly locates x whenever $x = A[i]$, provided that the unknown index i is θ-locally good. Intuitively, suppose we have already narrowed down our search to the smaller range $I = [i_0, i_1]$. The rank of $x = A[i]$ in $A'[i_0], \ldots, A'[i_1]$ is exactly $i - i_0 + 1$ since $A[i]$ is uncorrupted and the rank of x in $A[i_0], \ldots, A[i_1]$ can change by at most $\pm\theta(i - i_0 + 1)$ — at most $\theta(i_1 - i_0 + 1)$ indices $j' \in [i_0, i_1]$ can be corrupted since $i \in [i_0, i_1]$

is θ-locally good. Now suppose that we sample $t = \text{polylog}(n)$ indices $j_1, \ldots, j_t \in [i_0, i_1]$ and select the median y_{med} of $A[j_1], \ldots, A[j_t]$. With high probability the rank r of y_{med} in $A[j_1], \ldots, A[j_t]$ will be close to $(i_1 - i_0 + 1)/2$; i.e., $|r - (i_1 - i_0 + 1)/2| \leqslant \delta(i_1 - i_0 + 1)$ for some arbitrarily constant δ which may depend on the number of samples t. Thus, for suitable constants θ and δ whenever $x > y_{med}$ (resp. $x < y_{med}$) we can safely conclude that $i > i_0 + (i_1 - i_0 + 1)/8$ (resp. $i < i_1 - (i_1 - i_0 + 1)/8$) and search in the smaller interval $I' = [i_0 + (i_1 - i_0 + 1)/8, i_1]$ (resp. $I' = [i_0, i_1 - (i_1 - i_0 + 1)/8]$). In both cases the size of the search space is reduced by a constant multiplicative factor so the procedure will terminate after $O(\log(n))$ rounds making $O(t \cdot \log(n))$ queries. At its core our local decoding algorithm relies on a very similar idea

5.3.2 The Encoding Algorithm

Our encoder builds off of the known techniques of concatenation codes. First, a message x is encoded via the outer code to obtain some (intermediate) encoding y. We then partition y into some number k blocks $y = y_1 \circ \cdots \circ y_k$ and append each block y_i with index i to obtain $y_i \circ i$. Each $y_i \circ i$ is then encoded with the inner encoder to obtain some d_i. Then each d_i is prepended and appended with a run of 0s (i.e., buffers), to obtain c_i. The encoder then outputs $c = c_1 \circ \cdots \circ c_k$ as the final codeword. For our inner encoder, as mentioned before we in fact use the Schulman-Zuckerman (SZ) [230] edit distance code.

5.3.3 The Decoding Algorithm

Given oracle access to some corrupted codeword c', on input index i, the decoder simulates the outer decoder and must answer the outer decoder oracle queries. The decoder uses the inner decoder to answer these queries. However, there are two major challenges: (1) Unlike the Hamming-type errors, even only a few insertions and deletions make it difficult for the decoder to know where to probe; and (2) The boundaries between blocks can be ambiguous in the presence of InsDel errors. We overcome these challenges via a variant of binary search, which we name NoisyBinarySearch, together with a buffer detection algorithm, and make use of a block decomposition of the corrupted codeword to facilitate the analysis.

5.3.4 Analysis

The analysis of the binary search and the buffer detection algorithms are based on the notion of "good blocks" and "locally good blocks", which are natural extensions of the notion of θ-locally good indices discussed above. Recall that our encoder outputs a final codeword that is a concatenation of k smaller codeword "blocks"; namely $\mathsf{Enc}(x) = c_1 \circ \cdots \circ c_k$. Suppose c' is the corrupted codeword obtained by corrupting c with δ-fraction of insertion-deletion errors, and suppose we have a method of partitioning c' into k blocks $c'_1 \circ \cdots \circ c'_k$. Then we say that block c'_j is a γ-good block if it is within γ-fractional edit distance to the *uncorrupted* block c_j. Moreover, c'_j is (θ, γ)-locally good if at least $(1 - \theta)$ fraction of the blocks in every neighborhood around c'_j are γ-good and if the total number of corruptions in every neighborhood is bounded. Here θ and γ are suitably chosen constants. Both notions of good and locally good blocks are necessary to the success of our binary search algorithm NoisyBinarySearch.

The goal of NoisyBinarySearch is to locate a block with a given index j, and the idea is to decode the corrupted codeword at random positions to get a list of decoded indices (recall that the index of each block is appended to it). Since a large fraction of blocks are γ-good blocks, the sampled indices induce a new search interval for the next iteration. In order to apply this argument recursively, we need that the error density of the search interval does not increase in each iteration. Locally good blocks provide precisely this property.

5.3.5 Comparison of Techniques

The InsDel LDC construction of [215] also uses Schulman-Zuckerman (SZ) [230] codes, except it opens them up and directly uses the inefficient greedy inner codes used for the final efficient SZ codes themselves. In our case, we observe that the efficiently decodable codes of [230] have the additional property described in Lemma 5.6.2, which states that small blocks have large weight. This observation implies a running time that is polynomial in the query complexity of the final codes, since it helps make the buffer-finding algorithms local. The analysis of [215] also uses a binary search component, but our analysis and their analysis differ significantly.

5.3.6 Extending the Compiler to the Private and Resource-Bounded Setting

Let the compiler given by Theorem 5.2.1 be denoted as the BBGKZ compiler. The BBGKZ compiler at its core consists of two functions: $\mathsf{Compile}$ and $\mathsf{Recover}$. The function $\mathsf{Compile}$ takes as input a codeword $y \in \Sigma^K$ that is resilient to ρ-fraction of Hamming errors and outputs a codeword $Y \in \{0,1\}^n$ that is resilient to ρ'-fraction of insertion-deletion errors. The compiled encoding function operates as follows: it encodes a message x using the given Hamming LDC to obtain the Hamming codeword y, then it applies the function $\mathsf{Compile}$ to y and outputs the final InsDel codeword Y. The function $\mathsf{Recover}$, when given query access to some $Y' \in \{0,1\}^*$, on input i makes $\mathrm{polylog}(|Y'|)$ queries to Y' and attempts to recover y_i, the ith bit of the Hamming codeword y. The BBGKZ compiler guarantees that if $\mathsf{ED}(Y, Y') \leqslant \rho'$ then for *most* indices $i \in [K]$, $\mathsf{Recover}$ outputs the correct bit y_i with high probability.

The challenge in applying the BBGKZ compiler to a private Hamming LDC or a resource-bounded LDC is that we cannot assume that decoding will be correct for *every* corrupted codeword with small Hamming distance. Instead, we require that the channel cannot produce a codeword which fools the decoding algorithm except with negligible probability. In particular, if y is our encoding of a message x then we say that a corrupted codeword y' *fools* the decoder if:

1. the (Hamming/Edit) distance between y and y' is small; and
2. for some index i, the probability that the local decoder, given oracle access to y', outputs the correct bit x_i is less than p.

The security requirement is that any adversary $\mathcal{A}$ produces such a fooling codeword y' with probability at most ε. The difficulty here is proving that applying the BBGKZ compiler to a private code or resource-bounded code preserves the security of the underlying code. Proving the security of our compiled private/resource-bounded code lies in the algorithm $\mathsf{Recover}$: given an adversary $\mathcal{A}$ against the compiled InsDel code, we construct a new adversary $\mathcal{A}'$ against the original Hamming code which does the following:

1. obtains challenge message x and Hamming codeword y;

2. obtains InsDel codeword $Y = \mathsf{Compile}(y)$;

3. obtains $Y' \leftarrow \mathcal{A}(x, Y)$; and

4. obtains $y'_j \leftarrow \mathsf{Recover}^{Y'}(j)$ for all j.

Applying the key property of Recover one can show that the Hamming distance between y and y' is suitably small. Furthermore, if Y' fools our local InsDel decoder then one can argue that (w.h.p.) y' fools our local Hamming decoder. Thus, the compiler transforms secret key Hamming LDCs into secret key InsDel LDCs and resource bounded Hamming LDCs into resource bounded InsDel LDCs. For resource bounded channels, there is another subtlety we must account for. Our Hamming adversary $\mathcal{A}'$ requires *slightly* more resources than the original InsDel adversary $\mathcal{A}$; i.e., we need to run Recover for each index j (though this can be accomplished in parallel to minimize computation depth). Thus, to obtain an InsDel LDC secure against the channel class $\mathbb{C}$ we need to start with a Hamming LDC secure against a slightly larger class $\mathbb{C}'$ (i.e., the closure of $\mathbb{C}$).

5.4 Additional Related Work

Levenshtein [185] initiated the study of codes for insertions and deletions. Since this initiation, there has been a large body of works examining InsDel codes (see excellent surveys [205, 208, 245]). Recently, [241] constructed k-deletion correcting binary codes with optimal redundancy, which was extended to systematic binary codes and q-ary codes in [242, 243]. This line of work answered long standing open problems in the construction of k-deletion correcting codes with optimal redundancy. Random codes with positive information rate and correcting a large fraction of deletion errors were studied in [143, 179], and efficiently encodable and decodable codes with constant rate and resilient to a constant fraction of insertion-deletion errors were studied extensively in [70, 85, 86, 88, 89, 142–144, 150, 151, 230]. Recently, there has been interest in extending "list-decoding" to the setting of InsDel codes. These codes are resilient to a larger fraction of insertion-deletion errors at the cost of outputting a small list of potential codewords (i.e., the loss of unique decoding) [144, 152, 190]. Another direction due to Haeupler and Shahrasbi [151] involves constructing explicit

synchronization strings which can be "locally decoded" in the following sense: each index of the string can be computed using values located at a small number of other indices. These explicit and locally decodable synchronization strings are used to imply near linear time interactive coding schemes for insertion-deletion errors.

Cheng, Li, and Zheng [90] propose the notion of locally decodable codes with randomized encoding, in both the Hamming and edit distance regimes. They study such codes in various settings, including where the encoder and decoder share randomness, or the channel is oblivious to the codeword, and hence adds error patterns non-adaptively. For insertion-deletion errors they obtain codes with $K = O(k)$ or $K = k \cdot \log(k)$ and polylog(k) locality for message length k.

Blocki, Gandikota, Grigorescu, and Zhou [58] construct *relaxed* locally correctable and locally decodable Hamming codes in computationally bounded channels. Here, *local correction* states that a corrupt codeword c' can be corrected to codeword c by only querying c' at a bounded number of locations, and *relaxed* means that the correcting or decoding algorithm is allowed to output the value $\perp$ for a small fraction of inputs [37]. Their construction requires a public parameter setup for a collision-resistant hash function, and they obtain relaxed binary locally correctable and decodable Hamming codes with constant information rate and poly-logarithmic locality. Recently, Blocki, Kulkarni, and Zhou [60] introduced Hamming LDCs that are secure against resource-bounded adversaries, in the random oracle model. Here, they construct codes (in the random oracle model) which are resilient to classes of adversaries $\mathbb{C}$ for which there exists a function f that is uncomputable by any $\mathcal{A} \in \mathbb{C}$. They obtain explicit Hamming LDCs with constant information rate and polylogarithmic locality against various classes $\mathbb{C}$ of resource-bounded adversaries.

There are various other notions of "noisy search" that have been studied in the literature. Dhagat, Gács, and Winkler [103] consider a noisy version of the game "Twenty Questions". In this problem, an algorithm searches an array for some element x, and a bounded number of incorrect answers can be given to the algorithm queries, and the goal is to minimize the number of queries made by an algorithm. Feige et al. [115] study the depth of *noisy decision trees*: decision trees where each node gives the incorrect answer with some constant probability, and moreover each node success or failure is independent. Karp and Kleinberg [164] study

noisy binary search where direct comparison between elements is not possible; instead, each element has an associated biased coin. Given n coins with probabilities $p_1 \leqslant \cdots \leqslant p_n$, target value $\tau \in [0,1]$, and error ε, the goal is to design an algorithm which, with high probability, finds index i such that the intervals $[p_i, p_{i+1}]$ and $[\tau - \varepsilon, \tau + \varepsilon]$ intersect. Braverman and Mossel [71], Klein et al. [180] and Geissmann et al. [123] study noisy sorting in the presence of recurrent random errors: when an element is first queried, it has some (independent) probability of returning the incorrect answer, and all subsequent queries to this element are fixed to this answer. We note that each of the above notions of "noisy search" are different from each other and, in particular, different from our noisy search.

5.5 Preliminaries

We recall the definition of locally decodable codes.

Definition 2.7.5 (Locally Decodable Codes)**.** *A coding scheme* $C[K, k, q_1, q_2] = (\mathsf{Enc}, \mathsf{Dec})$ *is an* $(\ell, \rho, p, \mathrm{dist})$-locally decodable code *(LDC) if for all* $x \in \Sigma_1^k$ *and* $y \in \Sigma_2^*$ *such that* $\mathrm{dist}(\mathsf{Enc}(x), y) \leqslant \rho$*, the algorithm* Dec*, with query access to word* y*, on input index* $i \in [k]$*, makes at most* ℓ *queries to* y *and outputs* x_i *with probability at least* p *over the randomness of the decoder. Here,* ℓ *is the* locality *of* C *and* p *is the* success probability*.*

Given the definition of a LDC, we now define a *private LDC* [214].

Definition 5.5.1 (One-Time Private Key LDC)**.** *Let* $C[K, k, \lambda, q_1, q_2] = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *be a triple of probabilistic algorithms. We say* C *is a* $(\ell, \delta, p, \varepsilon, \mathrm{dist})$-private locally decodable code *(private LDC) if*

1. $\mathsf{Gen}(1^\lambda)$ *is the key generation algorithm that takes as input* 1^λ *and outputs secret key* $\mathsf{sk} \in \{0,1\}^*$ *for security parameter* λ*;*

2. $\mathsf{Enc}\colon \Sigma_1^k \times \{0,1\}^* \to \Sigma_2^K$ *is the encoding algorithm that takes as input message* $x \in \Sigma_1^k$ *and secret key* sk *and outputs a codeword* $y \in \Sigma_2^K$*; and*

3. $\mathsf{Dec}^{y'}\colon [k] \times \{0,1\}^* \to \Sigma_1$ *is the decoding algorithm that takes as input index* $i \in [k]$ *and secret key* sk*, is additionally given query access to a corrupted codeword* $y' \in \Sigma^{K'}$*, and outputs* $b \in \Sigma_1$ *after making at most* ℓ *queries to* y'*.*

We define a predicate $\mathsf{Fool}(y', \delta, p, \mathsf{sk}, x, y) = 1$ *if and only if*

1. $\mathrm{dist}(y, y') \leqslant \delta$; *and*
2. $\exists i \in [k]$ *such that* $\Pr\left[\mathsf{Dec}^{y'}(i, \mathsf{sk}) = x_i\right] < p$, *where the probability is taken over the random coins of* Dec.

We require that for all adversaries $\mathcal{A}$ *and all* $x \in \Sigma_1^k$, *it holds that* $\Pr\left[\mathsf{Fool}(\mathcal{A}(y), \delta, p, \mathsf{sk}, x, y) = 1\right] \leqslant \varepsilon$, *where* $y \leftarrow \mathsf{Enc}(x, \mathsf{sk})$ *and the probability is taken over the random coins of* $\mathcal{A}$ *and* Gen *and* Enc *(if encoding is randomized).*

For ease of presentation we present Definition 5.5.1 with respect to all adversaries $\mathcal{A}$; in this case, it is understood that the generation algorithm Gen outputs a sufficient number of purely random bits.

Next we present the definition of a *resource-bounded LDC* [60].

Definition 5.5.2 ($\mathbb{C}$-Secure LDC)**.** *For class of algorithms* $\mathbb{C}$, *a coding scheme* $C[K, k, q_1, q_2]$ *is an* $(\ell, \delta, p, \varepsilon, \mathrm{dist}, \mathbb{C})$-locally decodable code *if*

1. $\mathsf{Enc}\colon \Sigma_1^k \to \Sigma_2^K$ *is the encoding algorithm that takes as input message* $x \in \{0,1\}^k$ *and outputs a codeword* $y \in \{0,1\}^K$; *and*
2. $\mathsf{Dec}^{y'}\colon [k] \to \Sigma_1$ *is the decoding algorithm that takes as input index* $i \in [k]$, *is additionally given query access to a corrupted codeword* $y' \in \{0,1\}^{K'}$, *and outputs* $b \in \Sigma_1$ *after making at most* ℓ *queries to* y'.

We define predicate $\mathsf{Fool}(y', \delta, p, x, y) = 1$ *if and only if*

1. $\mathrm{dist}(y, y') \leqslant \delta$; *and*
2. $\exists i \in [k]$ *such that* $\Pr\left[\mathsf{Dec}^{y'}(i) = x_1\right] < p$,

where the probability is taken over the random coins of Dec; *otherwise* $\mathsf{Fool}(y', \delta, p, x, y) = 0$. *We require that for all adversaries* $\mathcal{A} \in \mathbb{C}$ *and all* $x \in \Sigma_1^k$, *it holds that* $\Pr\left[\mathsf{Fool}(\mathcal{A}(y), \delta, p, y) = 1\right] \leqslant \varepsilon$, *where the probability is taken over the random coins of* $\mathcal{A}$ *and the generation of the codeword* $y \leftarrow \mathsf{Enc}(x)$.

5.6 Insertion-Deletion LDCs from Hamming LDCs

We give our main construction of InsDel LDCs from Hamming LDCs. Our construction can be viewed as a procedure which, given outer codes C_{out} and binary inner codes C_{in} satisfying certain properties, produces binary codes $C(C_{out}, C_{in})$. This is formulated in the following theorem, which implies Theorem 5.2.1.

Theorem 5.6.1. *Let C_{out} and C_{in} be codes such that*

- *C_{out} defined by $\mathsf{Enc}_{out} \colon \Sigma^k \to \Sigma^m$ is an a $(\ell_{out}, \delta_{out}, \epsilon_{out})$-Hamming LDC.*
- *C_{in} is family of binary polynomial-time encodable/decodable codes with rate $1/\beta_{in}$ capable of correcting δ_{in} fraction of insertion-deletion errors. In addition, there are constants $\alpha_1, \alpha_2 \in (0,1)$ such that for any codeword c of C_{in}, any substring of c with length at least $\alpha_1|c|$ has fractional Hamming weight at least α_2.*

Then $C(C_{out}, C_{in})$ is a binary $\left(\ell_{out} \cdot O(\log^4(n')), \Omega(\delta_{out}\delta_{in}), \epsilon - \mathsf{negl}(n')\right)$-InsDel LDC. Here the codewords of C have length $n = \beta m$ where $\beta = O(\beta_{in} \log(|\Sigma|))$, and n' denotes the length of received word.

For the inner code, we make use of the following efficient code constructed by Schulman-Zuckerman [230].

Lemma 5.6.2 (SZ-code [230])**.** *There exist constants $\beta_{in} \geq 1$, $\delta_{in} > 0$, such that for large enough values of $t > 0$, there exists a code $SZ(t) = (\mathsf{Enc}, \mathsf{Dec})$ where $\mathsf{Enc} : \{0,1\}^t \to \{0,1\}^{\beta_{in} t}$ and $\mathsf{Dec} : \{0,1\}^{\beta_{in} t} \to \{0,1\}^t \cup \{\perp\}$ capable of correcting δ_{in} fraction of InsDel errors, having the following properties:*

1. *Enc and Dec run in time $poly(t)$;*
2. *For all $x \in \{0,1\}^t$, every interval of length $2 \log t$ of $\mathsf{Enc}(x)$ has fractional Hamming weight at least $2/5$.*

We formally complete the proof of correctness of Theorem 5.6.1 in Section 5.6.3. We dedicate the remainder of this section to outlining the construction of the encoding and decoding algorithms.

5.6.1 Encoding and Decoding Algorithms

In our construction of $C(C_{out}, C_{in})$, we denote the specific code of Lemma 5.6.2 as our inner code $C_{in} = (\mathsf{Enc}_{in}, \mathsf{Dec}_{in})$. For our purpose, we view a message $x \in \Sigma^m$ as a pair in $[m] \times \Sigma^{\log(m)}$. The encoding function $\mathsf{Enc}_{in} \colon [m] \times \Sigma^{\log(m)} \to \{0,1\}^{\beta_{in}(1+\log(|\Sigma|))\log(m)}$ maps a string in Σ of length $\log(m)$ appended with an index from set $[m]$ — i.e., a (padded) message of bit-length $(1 + \log(|\Sigma|))\log(m)$ — to a binary string of length $\beta_{in}(1 + \log(|\Sigma|))\log(m)$. The inner decoder Dec_{in} on input y' returns x if $\mathsf{ED}(y', y) \leq \delta_{in} \cdot 2|y|$ where $y = \mathsf{Enc}_{in}(x)$. The information rate of this code is $R_{in} = 1/\beta_{in}$.

The Encoder (Enc)

Given an input string $x \in \Sigma^k$ and outer code $C_{out} = (\mathsf{Enc}_{out}, \mathsf{Dec}_{out})$, our final encoder Enc does the following:

1. Computes the outer encoding of x as $s = \mathsf{Enc}_{out}(x)$;
2. For each $i \in [m/\log(m)]$, groups $\log m$ symbols $s[(i-1)\log(m), i\log(m) - 1]$ into a single block $b_i \in \Sigma^{\log(m)}$;
3. For each $i \in [m/\log(m)]$, computes the i^{th} block of the inner encoding as $Y^{(i)} = \mathsf{Enc}_{in}(i \circ b_i)$ — i.e., computes the inner encoding of the i^{th} block concatenated with the index i;
4. For some constant $\alpha \in (0,1)$ (to be decided), appends a $\alpha\log(m)$-long buffer of zeros before and after each block; and
5. Outputs the concatenation of the buffered blocks (in indexed order) as the final codeword $c = \mathsf{Enc}(x) \in \{0,1\}^n$, where

$$c = \left(0^{\alpha\log(m)} \circ Y^{(1)} \circ 0^{\alpha\log(m)}\right) \circ \cdots \circ \left(0^{\alpha\log(m)} \circ Y^{(m/\log(m))} \circ 0^{\alpha\log(m)}\right). \quad (5.1)$$

Denoting $\beta = 2\alpha + \beta_{in}\,(1 + \log(|\Sigma|))$, the length of $c = \mathsf{Enc}(x)$ is

$$n = (2\alpha \log m + \beta_{in}\,(1 + \log(|\Sigma|)) \log(m)) \cdot \frac{m}{\log(m)} = \beta m.$$

The LDC Decoder (Dec)

We start off by describing the high-level overview of our decoder Dec and discuss the challenges and solutions behind its design. As defined in Equation (5.1), our encoder Enc, on input $x \in \Sigma^k$, outputs a codeword $c = c_1 \circ \cdots \circ c_d \in \{0,1\}^n$, where $d = m/\log(m)$. The decoder setting is as follows: on input $i \in [k]$ and query access to the corrupted codeword $c' \in \{0,1\}^{n'}$ such that $\mathsf{ED}(c, c') \leqslant 2n\delta$, our final decoder Dec needs to output the message symbol $x[i]$ with high probability. Notice that if Dec had access to the original codeword $s = \mathsf{Enc}_{out}(x)$, then Dec could simply run $\mathsf{Dec}_{out}(i)$ while supplying it with oracle access to this codeword s. This naturally motivates the following decoding strategy: simulate oracle access to the codeword s by answering the queries of Dec_{out} by decoding the appropriate bits using Dec_{in}. We give a detailed description of this strategy next.

Let $Q_i = \{q_1, \ldots, q_{\ell_{out}}\} \subset [m]$ be a set of indices which $\mathsf{Dec}_{out}(i)$ queries.[1] We observe that if our decoder had oracle access to the uncorrupted codeword c, then answering these queries would be simple:

1. For each $q \in Q_i$, let $b_j = s[(j-1)\log(m), j\log(m) - 1]$ be the block which contains $s[q]$. In particular, $q = (j-1)\log(m) + r_j$ for some $r_j \in [0, \log(m) - 1]$,
2. Obtain block c_j by querying oracle c and obtain $Y^{(j)}$ by removing the buffers from c_j,
3. Obtain $j \circ b_j$ by running $\mathsf{Dec}_{in}(Y^{(j)})$, then return $s[q] = b_j[r_j]$ to Dec_{out}.

In fact, it suffices to answer the queries of Dec_{out} with symbols consistent with any string s' such that $\mathsf{HAM}(s, s') \leqslant m\delta_{out}$. Then the correctness of the output would follow from the correctness of Dec_{out}. We carry out the strategy mentioned above, except that now we are given a corrupted codeword c'.

[1] ↑Our construction also supports adaptive queries, but we use non-adaptive queries for ease of presentation.

For the purposes of analysis, we first define the notion of a *block decomposition* of the corrupted codeword c'. Informally, a block decomposition is simply a partitioning of c' into contiguous blocks. Our first requirement for successful decomposition is that there must exist a block decomposition $c' = c'_1 \circ \cdots \circ c'_d$ that is "not too different" from the original decomposition $c = c_1 \circ \cdots \circ c_d$.[2] In particular, we require that $\sum_j \mathsf{ED}(c'_j, c_j) \leqslant 2n\delta$, which is guaranteed by Proposition 5.6.1. Next, we define the notion of γ-*good* (see Definition 5.6.3). The idea here is that if a block c'_j is γ-good (for appropriate γ), then we can run Dec_{in} on c'_j and obtain $j \circ b_j$. As the total number of errors is bounded, it is easy to see that all but a small fraction of blocks are γ-good (Lemma 5.6.3). At this point, we are essentially done if we can decode c'_j for any given γ-good block j.

An immediate challenge we are facing is that of *locating* a specific γ-good block c'_j, while maintaining overall locality. The presence of insertions and deletions may result in uneven block lengths and misplaced blocks, making the task of locating a specific block non-trivial. However, γ-good blocks make up the majority of the blocks and enjoy the property that they are in correct relative order, it is conceivable to perform a *binary search* style of algorithm over the blocks of c' to find block c'_j. The idea is to maintain a search interval and iteratively reduce its size by a constant multiplicative factor. In each iteration, the algorithm samples a small number of blocks and obtains their (appended) indices. As the vast majority of blocks are γ-good, these indices guide the binary search algorithm in narrowing down the search interval. Though there is one problem with this argument: the density of γ-good blocks may decrease as the search interval becomes smaller. In fact, it is impossible to *locally* locate a block c'_j surrounded by many bad blocks, even if c'_j is γ-good. This is where the notion of (θ, γ)-locally good (see Definition 5.6.5) helps us: if a block c'_j is (θ, γ)-locally good, then $(1-\theta)$-fraction of blocks in every neighborhood around c'_j are γ-good, and every neighborhood around c'_j has a bounded number of errors. Therefore, as long as the search interval contains a locally good block, we can lower bound the density of γ-good blocks and recover c'_j with high probability.

[2]↑We note that we do not need to know this decomposition explicitly, and that its existence is sufficient for our analysis.

Our *noisy binary search algorithm* essentially implements this idea. On input block index j, the algorithm searches for block j. If block j is (θ, γ)-locally good, then we can guarantee that our noisy binary search algorithm will find j except with negligible probability (see Theorem 5.6.6). Thus it is desirable that the number of (θ, γ)-locally good blocks is large; if this number is large, the noisy binary search is effectively providing oracle access to a string s' which is close to s in Hamming distance, and thus the outer decoder is able to decode $x[i]$ with high probability. Lemma 5.6.4 exactly guarantees this property.

The discussion above requires knowing the boundaries of each block c'_j, which is non-trivial even in the no corruption case. As the decoder is oblivious to the block decomposition, the decoder works with *approximate boundaries* which can be found locally by a *buffer search algorithm*, described as follows. Recall that by construction c_j consists of $Y^{(j)}$ surrounded by buffers of $(\alpha \log m)$-length 0-runs. So to find $Y^{(j)}$, it suffices to find the buffers surrounding $Y^{(j)}$. Our buffer search algorithm can be viewed as a "local variant" of the buffer search algorithm of Schulman and Zuckerman [230]. This algorithm is designed to find approximate buffers surrounding a block c'_j if it is γ-good. Then the string in between two buffers is identified as a corrupted codeword and is decoded to $j \circ b_j$. The success of the algorithm depends on γ-goodness of the block being searched and requires that any substring of a codeword from C_{in} has "large enough" Hamming weight. In fact, our inner code given by Lemma 5.6.2 gives us this exact guarantee. All together, this enables the noisy binary search algorithm to use the buffer finding algorithm to search for a block c'_j.

We formalize the decoder outlined above. On input $i \in [k]$, Dec simulates $\mathsf{Dec}_{out}(i)$ and answers its queries. Whenever $\mathsf{Dec}_{out}(i)$ queries an index $j \in [m]$, Dec expresses $j = (p-1)\log m + r_j$ for $p \in [m/\log m]$ and $r_j \in [0, \log m - 1]$, and runs $\mathsf{NoisyBinarySearch}(c', p)$ (which calls the algorithm Buff-Find) to obtain a string $b' \in \Sigma^{\log m}$ (or $\perp$). Then it feeds the r_j-th symbol of b' (or $\perp$) to $\mathsf{Dec}_{out}(i)$. Finally, Dec returns the output of $\mathsf{Dec}_{out}(i)$.

5.6.2 Block Decomposition of Corrupted Codewords

The analysis of our decoding procedure relies on a so-called buffer finding algorithm and a noisy binary search algorithm. To analyze these algorithms, we introduce the notion of a

block decomposition for (corrupted) codewords, as well as what it means for a block to be *(locally) good*.

For convenience, we now fix some notation for the remainder of the paper. We fix an arbitrary message $x \in \Sigma^k$. We use $s = \mathsf{Enc}_{out}(x) \in \Sigma^m$ for the encoding of x by the outer encoder. Let $\tau = \log(m)$ be the length of each block and $d = m/\log m$ be the number of blocks. For $i \in [d]$, we let $b_i \in \Sigma^\tau$ denote the i-th block $s[(i-1)\tau, i\tau - 1]$, and let $Y^{(i)}$ denote the encoding $\mathsf{Enc}_{in}(i \circ b_i)$. Recall that $\alpha\tau$ is the length of the appended buffers for some $\alpha \in (0,1)$, and the parameter $\beta = 2\alpha + \beta_{in}(1 + \log|\Sigma|)$. Thus $|Y^{(i)}| = (\beta - 2\alpha)\tau$. The final encoding is given by

$$c = \tilde{Y}^{(1)} \circ \tilde{Y}^{(2)} \circ \cdots \circ \tilde{Y}^{(d)},$$

where $\tilde{Y}^{(j)} = 0^{\alpha\tau} \circ Y^{(j)} \circ 0^{\alpha\tau}$ and $|\tilde{Y}^{(j)}| = \beta\tau$. The length of c is $n = d\beta\tau = \beta m$. We let $c' \in \{0,1\}^{n'}$ denote a corrupted codeword satisfying $\mathsf{ED}(c, c') \leq 2n \cdot \delta$.

Definition 5.6.1 (Block Decomposition)**.** *A* block decomposition *of a (corrupted) codeword c' is a non-decreasing mapping $\phi\colon [n'] \to [d]$ for $n', d \in \mathbb{Z}^+$.*

We say a set $I \subseteq [n']$ is an interval if $I = \emptyset$ (i.e., an empty interval) or $I = \{l, l+1, \ldots, r-1\}$ for some $1 \leq l < r \leq n'$, in which case we write $I = [l, r)$. For an interval $I = [l, r)$, we write $c'[I]$ for the substring $c'[l]c'[l+1]\ldots c'[r-1]$. Finally, $c[\emptyset]$ stands for the empty string.

We remark that for a given block decomposition ϕ, since ϕ is non-decreasing we have that for every $j \in [d]$ the pre-image $\phi^{-1}(j)$ is an interval. Since ϕ is a total function, it induces a partition of $[n']$ into d intervals $\{\phi^{-1}(j)\colon j \in [d]\}$. The following definition plays an important role in the analysis.

Definition 5.6.2 (Closure Intervals)**.** *The* closure of an interval $I = [l, r) \subseteq [n']$ *is defined as* $\cup_{i=l}^{r-1} \phi^{-1}(\phi(i))$. *An interval I is a* closure interval *if the closure of I is itself. Equivalently, every closure interval has the form $\mathcal{I}[a, b] \coloneqq \bigcup_{j=a}^{b} \phi^{-1}(j)$ for some $a, b \in [d]$.*

Proposition 5.6.1. *There exists a block decomposition $\phi\colon [n'] \to [d]$ such that*

$$\sum_{j \in [d]} \mathsf{ED}\left(c'[\phi^{-1}(j)],\ \tilde{Y}^{(j)}\right) \leq \delta \cdot 2n.$$

Proof. Let $\phi_0 \colon [n] \to [d]$ be the block decomposition for c satisfying $\phi_0(i) = j$ if i lies in block $\tilde{Y}^{(j)}$. Without loss of generality, we assume the adversary performs the following corruption process:

1. The adversary picks some $j \in [d]$;
2. The adversary corrupts $\tilde{Y}^{(j)}$.

Steps (1) and (2) are repeated up to the specified edit distance bound of $2\delta n$. We construct $\phi \colon [n'] \to [d]$ by modifying the decomposition ϕ_0 according to the above process. It is clear that ϕ satisfies the desired property. □

We now introduce the notion of *good blocks*. In the following definitions, we also fix an arbitrary block decomposition ϕ of c' enjoying the property guaranteed by Proposition 5.6.1.

Definition 5.6.3 (γ-good block)**.** *For $\gamma \in (0,1)$ and $j \in [d]$ we say that block j is* γ-good *if* $\mathsf{ED}(c'[\phi^{-1}(j)], \widetilde{Y}^{(j)}) \leqslant \gamma\alpha\tau$. *Otherwise we say that block j is* γ-bad.

Definition 5.6.4 ((θ,γ)-good interval)**.** *We say a closure interval $\mathcal{I}[a,b]$ is (θ,γ)-good if the following hold:*

1. $\sum_{j=a}^{b} \mathsf{ED}\left(c'[\phi^{-1}(j)], \tilde{Y}^{(j)}\right) \leq \gamma \cdot (b-a+1)\alpha\tau$.
2. *There are at least $(1-\theta)$-fraction of γ-good blocks among those indexed by $\{a, a+1, \cdots, b\}$.*

Definition 5.6.5 ((θ,γ)-local good block)**.** *For $\theta, \gamma \in (0,1)$ we say that block j is* (θ,γ)-local good *if for every $a, b \in [d]$ such that $a \leqslant j \leqslant b$ the interval $\mathcal{I}[a,b]$ is (θ,γ)-good. Otherwise, block j is (θ,γ)-locally bad.*

Note that in Definition 5.6.5, if j is (θ,γ)-locally good, then j is also γ-good by taking $a = b = j$.

Proposition 5.6.2. *The following bounds hold:*

1. *For any γ-good block j, $(\beta - \alpha\gamma)\tau \leqslant |\phi^{-1}(j)| \leqslant (\beta + \alpha\gamma)\tau$.*

2. *For any (θ,γ)-good interval $\mathcal{I}[a,b]$, $(b-a+1)(\beta-\alpha\gamma)\tau \le |\mathcal{I}[a,b]| \le (b-a+1)(\beta+\alpha\gamma)\tau$.*

Proof. For item (1) note that an uncorrupted block has length $\beta\tau$. Since j is γ-good, we know that $\mathsf{ED}(c'[\phi^{-1}(j)], \widetilde{Y}^{(j)}) \leqslant \gamma\alpha\tau$, which implies that $(\beta-\alpha\gamma)\tau \leqslant |\phi^{-1}(j)| \leqslant (\beta+\alpha\gamma)\tau$.

For item (2), we first note that $|a| - \Delta \le |b| \le |a| + \Delta$ where $\Delta = \mathsf{ED}(a,b)$. Let $\Delta_j = \mathsf{ED}\left(c'[\phi^{-1}(j)], \widetilde{Y}^{(j)}\right)$. By definition of (θ,γ)-good interval, we have that $\sum_{j=a}^{b} \Delta_j \leqslant \gamma(b-a+1)\alpha\tau$. This gives us the following two properties.

$$|\mathcal{I}[a,b]| = \sum_{j=a}^{b} \left|\phi^{-1}(j)\right| \le \sum_{j=a}^{b} \beta\tau + \Delta_j \le (b-a+1)(\beta+\alpha\gamma)\tau,$$
$$|\mathcal{I}[a,b]| = \sum_{j=a}^{b} \left|\phi^{-1}(j)\right| \ge \sum_{j=a}^{b} \beta\tau - \Delta_j \ge (b-a+1)(\beta-\alpha\gamma)\tau.$$

□

The following lemmas upper bound the number of γ-bad and (θ,γ)-locally bad blocks.

Lemma 5.6.3. *The total fraction of γ-bad blocks is at most $2\beta\delta/(\gamma\alpha)$.*

Proof. Let $\Delta_j = \mathsf{ED}(c'[\phi^{-1}(j)], \widetilde{Y}_j)$ for every $j \in [d]$. By our choice of ϕ and Proposition 5.6.1 we have that:

$$\sum_{j=1}^{d} \Delta_j \leqslant 2n \cdot \delta.$$

Let $\mathsf{Bad} \subseteq [d]$ be the set of γ-bad blocks. Then we have

$$\delta \cdot 2n \ge \sum_{j=1}^{d} \Delta_j \geqslant \sum_{i \in \mathsf{Bad}} \Delta_i > |\mathsf{Bad}| \cdot \gamma\alpha\tau$$

where the latter inequality follows by the definition of γ-bad. Thus we obtain $|\mathsf{Bad}| < \delta n/\gamma\alpha\tau$. Recalling that $n = \beta d\tau$ we have that $|\mathsf{Bad}|/d < 2\beta\delta/(\gamma\alpha)$ as desired. □

Lemma 5.6.4. *The total fraction of (θ,γ)-local bad blocks is at most $(4/\gamma\alpha)(1+1/\theta)\delta\beta$.*

Proof. First we count the number of blocks which violate condition (1) of Definition 5.6.4. We proceed by counting in two steps. Suppose that $i_1 \in [d]$ is the smallest index such that block

i_1 violates (1) of Definition 5.6.4 with witness (i_1, b_1); that is, $\sum_{j=i_1}^{b_1} \mathsf{ED}(c'[\phi^{-1}(j)], \widetilde{Y}^{(j)}]) > \gamma \cdot (b_1 - i_1 + 1)\tau$. Continuing inductively, let $i_k \in [d]$ be the smallest index such that $i_k > i_{k-1} + b_{k-1}$ and i_k violates condition (1) of Definition 5.6.4 with witness (i_k, b_k). Let $\{(i_k, b_k)\}_{k=1}^{t}$ for some t be the result of this procedure. Further let $D_k = \sum_{i=i_k}^{b_k} \mathsf{ED}\left(c'[\phi^{-1}(i)], \widetilde{Y}^{(i)}\right)$ for every $k \in [t]$. Let $n_\gamma^{(1)}$ be the total number of locally bad blocks j of the form (j, b) for some b. Then we claim that (1) $n_\gamma^{(1)} \leqslant \sum_{k=1}^{t} b_k - i_k$, (2) for all $k \in [t]$ we have that $D_k > \gamma\tau(b_k - i_k)$, and (3) $\sum_{k=1}^{t} D_k \leqslant \mathsf{ED}(c, c')$. The first equation follows from the fact that any locally bad block j with witness (j, b) for some $b \geqslant j$ must fall into some interval $[i_k, b_k]$, else this would contradict the minimality of the chosen i_k. The second equation follows directly by definition of local good. The third equation follows from the fact that the sum of D_k is at most the sum of all possible blocks, which is upper bounded by the edit distance. Combining these equations we see that $n_\gamma^{(1)} \leqslant 2\delta n/(\gamma\alpha\tau)$. Symmetrically, we can consider all bad blocks j which violate condition (1) of Definition 5.6.4 and have witnesses of the form (a, j). For this bound we obtain $n_\gamma^{(2)} \leqslant 2\delta n/(\gamma\alpha\tau)$.

Now we consider the number of bad blocks which violate condition (2) of Definition 5.6.4. By identical analysis and first considering bad blocks j with witnesses of the form (j, b), we obtain a set of minimally chosen witnesses $\{(i_k, b_k)\}_{k=1}^{t}$. Let $n_\theta^{(1)}$ be the total number of bad blocks j with witnesses of the form (j, b). Further, let B_k denote the number of γ-bad blocks in the interval $[i_k, b_k]$. Then we have (1) $n_\theta^{(1)} \leqslant \sum_{k=1}^{t} b_k - i_k$, (2) for all $k \in [t]$, $B_k > \theta(b_k - i_k)$, and (3) $\sum_{k=1}^{t} B_k \leqslant \mathsf{ED}(c, c')/(\gamma\alpha\tau)$. Then by these three equations we have that $n_\theta^{(1)} \leqslant 2\delta n/(\gamma\theta\alpha\tau)$. By a symmetric argument, if $n_\theta^{(2)}$ is the total number of blocks j which violate condition (2) of Definition 5.6.4 with witnesses of the form (a, j) then we have $n_\theta^{(2)} \leqslant 2\delta n/(\theta\gamma\alpha\tau)$.

Thus the total number of possible bad blocks violating either condition is at most $(4/\gamma\alpha\tau)(1 + 1/\theta)\delta n$. Recalling that $n = \beta d\tau$, we have that the total fraction of locally bad blocks is at most $(4/\gamma\alpha)(1 + 1/\theta)\delta\beta$ as desired. □

5.6.3 Outer Decoder

At a high level, the our decoding algorithm Dec runs the outer decoder Dec_{out} and must answer all oracle queries of Dec_{out} by simulating oracle access to some corrupted string s'. Recall that C_{out}, with encoding function $\mathsf{Enc}_{out} \colon \Sigma^k \to \Sigma^m$, is a $(\ell_{out}, \delta_{out}, \varepsilon_{out})$-LDC for Hamming errors. Further, C_{out} has probabilistic decoder Dec_{out} such that for any $i \in [k]$ and string $s' \in (\Sigma \cup \{\perp\})^m$ such that $\mathsf{HAM}(s', s) \leq m \cdot \delta_{out}$ for some codeword $s = \mathsf{Enc}_{out}(x)$, we have

$$\Pr\left[\mathsf{Dec}_{out}^{s'}(i) = x[i]\right] \geq \frac{1}{2} + \varepsilon_{out}.$$

Additionally, Dec_{out} makes at most ℓ_{out} queries to s'.

In order to run Dec_{out}, we need to simulate oracle access to such a string s'. To do so, we present our noisy binary search algorithm Figure 5.1 in Section 5.6.4. For now, we assume Figure 5.1 has the properties stated in the following proposition and theorem.

Proposition 5.6.3. *Figure 5.1 has query complexity* $O\left(\log^4 n'\right)$.

Theorem 5.6.5. *For* $j \in [d]$, *let* $\mathbf{b}_j \in \Sigma^\tau \cup \{\perp\}$ *be the random variable denoting the output of Figure 5.1 on input* $(c', 1, n' + 1, j)$. *We have*

$$\Pr\left[\Pr_{j \in [d]}\left[\mathbf{b}_j \neq b_j\right] \geq \delta_{out}\right] \leq \mathsf{negl}(n'),$$

where the probability is taken over the joint distribution of $\{\mathbf{b}_j \colon j \in [d]\}$.

We note that in Theorem 5.6.5, the random variables $\mathbf{b}_j$ do not need to be independent, i.e., two runs of Figure 5.1 can be correlated. For example, we can fix the random coin tosses of Figure 5.1 before the first run and reuse them in each call.

5.6.4 Noisy Binary Search

We present Figure 5.1 in this section. As mentioned in Section 5.6.3, the binary search algorithm discussed in this section can be viewed as providing the outer decoder with oracle

access to some string $s' \in (\Sigma \cup \{\perp\})^m$. Namely whenever the outer decoder queries an index $j \in [m]$ which lies in block p, we run Noisy-Binary-Search on $(c', 1, n'+1, p)$ and obtain a string $b'_p \in \Sigma^{\log m}$ which contains the desired symbol $s'[j]$.

Noisy Binary Search $^{c'}(j)$

Input : An index $j \in [d]$, and oracle access to a codeword $c' \in \{0,1\}^{n'}$.

Output : A string $b \in \Sigma^{\tau}$ or $\perp$.

$N \leftarrow \Theta(\log^2 n')$

$\rho \leftarrow \min\{\frac{1}{4} \cdot \frac{\beta-\gamma}{\beta+\gamma}, 1 - \frac{3}{4} \cdot \frac{\beta+\gamma}{\beta-\gamma}\}$

$C \leftarrow 36(\beta+\gamma)\tau$

function `Noisy-Binary-Search`(c', l, r, j)

  if $r - l \leq C$ **then**

    $s \leftarrow$ `Interval-Decode`$(l,\ r,\ j)$

    return s

  $m_1 \leftarrow (1-\rho)l + \rho r$, $m_2 \leftarrow \rho l + (1-\rho)r$

  foreach $t \in [N]$ **do**

    Randomly sample $i \xleftarrow{\$} \{m_1, m_1+1, \ldots, m_2 - 1\}$

    $j_t \leftarrow$ `Block-Decode`(i)

  $\tilde{j} \leftarrow$ median of $j_1, \ldots, j_N$ (ignore j_t if $j_t = \perp$)

  if $j \leqslant \tilde{j}$ **then**

    return `Noisy-Binary-Search`(c', l, m_2, j)

  else

    return `Noisy-Binary-Search`(c', m_1, r_j)

Figure 5.1. Our Noisy Binary Search Algorithm

We analyze the query complexity of Figure 5.1 and prove Proposition 5.6.3.

Proposition 5.6.3. *Figure 5.1 has query complexity* $O\left(\log^4 n'\right)$.

Proof. The number of iterations T is at most $O\left(\log \frac{n'}{C}\right) = O(\log n')$ as $r - l$ is reduced by a constant factor $1 - \rho$ in each iteration until it goes below C. In each iteration (except for the last iteration), the algorithm makes $N = \Theta(\log^2 n')$ calls to Block-Decode, which has query complexity $O(\log n')$. In the last iteration, it calls Interval-Decode which has query complexity $O(\log n')$. Thus the overall query complexity is $O\left(\log^4 n'\right)$. □

The following theorem shows that the set of indices which can be correctly returned by Figure 5.1 is captured by the locally good property.

Theorem 5.6.6. *If $j \in [d]$ is a (θ, γ)-locally good block, running Figure 5.1 on input $(c', 1, n'+1, j)$ outputs b_j with probability at least $1 - \mathsf{negl}(n')$.*

We defer the proof of Theorem 5.6.6 to Section 5.B, as the proof requires many auxiliary claims and lemmas. For now, we assume Theorem 5.6.6 and work towards proving Theorem 5.6.5.

We first observe that the only time Figure 5.1 interacts with c' is when it queries Block-Decode and Interval-Decode. Thus the properties of these two algorithms is essential to our proof. We briefly describe these two subroutines now.

- Block-Decode: On input index $i \in [n']$, Block-Decode tries to find the block j that contains i, and attempts to decode the block to $j \circ b_j$. It returns the index j if the decoding was successful, and $\perp$ otherwise.
- Interval-Decode: On input $l, r \in [n']$ and $j \in [d]$, Interval-Decode (roughly) runs the buffer search algorithm of Schulman and Zuckerman [230] over the substring $c'[l, r]$ to obtain a set of approximate buffers, and attempts to decode all strings separated by the approximate buffers. It returns b if any string is decoded to $j \circ b$, and $\perp$ otherwise.

For convenience, we model Block-Decode as a function $\varphi \colon [n'] \to [d] \cup \{\perp\}$, and model Interval-Decode as a function $\psi \colon [n'] \to \Sigma^\tau \cup \{\perp\}$. The functions φ and ψ have the following properties, which are crucial to the proof of Theorem 5.6.5.

Theorem 5.6.7. *The functions φ and ψ satisfy the following properties:*

1. *For any γ-good block j we have*

$$\Pr_{i \in \phi^{-1}(j)} [\varphi(i) \neq j] \leq \gamma.$$

2. *Let $[l, r)$ be an interval with closure $\mathcal{I}[L, R-1]$, satisfying that every block $j \in \{L, \ldots, R-1\}$ is γ-good. Then for every block j such that $\phi^{-1}(j) \subseteq [l, r)$, we have $\psi(j, l, r) = b_j$.*

Given Theorem 5.6.6 and Theorem 5.6.7, we recall and prove Theorem 5.6.5.

Theorem 5.6.5. *For $j \in [d]$, let $\mathbf{b}_j \in \Sigma^\tau \cup \{\perp\}$ be the random variable denoting the output of Figure 5.1 on input $(c', 1, n'+1, j)$. We have*

$$\Pr\left[\Pr_{j\in[d]}[\mathbf{b}_j \neq b_j] \geq \delta_{out}\right] \leq \mathsf{negl}(n'),$$

where the probability is taken over the joint distribution of $\{\mathbf{b}_j : j \in [d]\}$.

Proof. Let $\mathsf{Good} \subseteq [d]$ be the set of (θ, γ)-locally-good blocks, and let $\overline{\mathsf{Good}} = [d] \setminus \mathsf{Good}$. Lemma 5.6.4 implies that

$$\left|\overline{\mathsf{Good}}\right| \leq \left(1 + \frac{1}{\theta}\right)\frac{\delta d \beta}{\alpha\gamma} = \frac{\delta_{out} d}{2}.$$

For each $j \in \mathsf{Good}$, denote by E_j the event $\{\mathbf{b}_j \neq b_j\}$. Theorem 5.6.6 in conjunction with a union bound implies that

$$\Pr\left[\bigcup_{j\in\mathsf{Good}} E_j\right] \leq \mathsf{negl}(n').$$

Since

$$\begin{aligned}\Pr_{j\in[d]}[\mathbf{b}_j \neq b_j] &\leq \Pr_{j\in[d]}\left[j \in \overline{\mathsf{Good}}\right] + \Pr_{j\in[d]}[\mathbf{b}_j \neq b_j \mid j \in \mathsf{Good}] \\ &\leq \frac{\delta_{out}}{2} + \Pr_{j\in[d]}[\mathbf{b}_j \neq b_j \mid j \in \mathsf{Good}],\end{aligned}$$

we have

$$\begin{aligned}\Pr\left[\Pr_{j\in[d]}[\mathbf{b}_j \neq b_j] \geq \delta_{out}\right] &\leq \Pr\left[\Pr_{j\in[d]}[\mathbf{b}_j \neq b_j \mid j \in \mathsf{Good}] \geq \frac{\delta_{out}}{2}\right] \\ &\leq \Pr\left[\bigcup_{j\in\mathsf{Good}} E_j\right] \leq \mathsf{negl}(n').\end{aligned}$$

□

5.6.5 Block Decode Algorithm

A key component of the Noisy Binary Search algorithm is the ability to decode γ-good blocks in the corrupted codeword c'. In order to do so, our algorithm will take explicit advantage of the γ-good properties of a block. We present our block decoding algorithm, named Block-Decode, in Figure 5.2.

Block-Decode$^{c'}(i)$
Input : An index $i \in [n']$ and oracle access to (corrupted) codeword $c' \in \{0,1\}^{n'}$.
Output : Some string $\mathsf{Dec}(s)$ for a substring s of c', or $\perp$.

```
buff ← Buff-Find_η^{c'}(i)
if buff = ⊥ then
    return ⊥
else
    return Dec_in(c'[b + 1, a' − 1])
return ⊥
```

Figure 5.2. Our Block Decoding Algorithm

Buff-Find$^{c'}_{\eta}(i)$
Input : An index $i \in [n']$ and oracle access to (corrupted) codeword $c' \in \{0,1\}^{n'}$.
Output : Two consecutive δ_{b}-approximate buffers $(a,b),(a',b')$, or $\perp$.

```
j_s ← max{1, i − ητ}, j_e ← min{n' − τ + 1, i + ητ}
buffs ← [ ] ;                                   /* i.e., empty buffer */
while j_s ⩽ j_e do
    if ED(0^τ, c'[j_s, j_s + τ − 1]) ⩽ δ_b ατ then
        buffs.append((j_s, j_s + τ − 1))
    j_s ← j_s + 1
foreach k ∈ {0, 1, ..., |buffs| − 2} do
    (a, b) ← buffs[k], (a', b') ← buffs[k + 1]
    if b < i < a' then
        return (a, b), (a', b')
return ⊥
```

Figure 5.3. Our Buffer Finding Algorithm

Buff-Find

The algorithm Block-Decode makes use of the sub-routine Buff-Find, presented in Figure 5.3. At a high-level, the algorithm Buff-Find on input i and given oracle access to (corrupted) codeword c' searches the ball $c'[i-\eta\tau, i+\eta\tau]$ for all δ_{b}-*approximate buffers* in the interval, where $\eta \geqslant 1$ is a constant such that if $i \in \phi^{-1}(j)$ for any good block j then $c'[\phi^{-1}(j)] \subseteq c'[i-\eta\tau, i+\eta\tau]$. Briefly, for any $k \in \mathbb{N}$ and $\delta_{\mathsf{b}} \in (0, 1/2)$ a string $w \in \{0,1\}^k$ is a δ_{b}-*approximate buffer* if $\mathsf{ED}(w, 0^k) \leqslant \delta_{\mathsf{b}} \cdot k$. For brevity we refer to approximate buffers simply as buffers. Once all buffers are found, the algorithm attempts to find a pair of consecutive buffers such that the index i is between these two buffers. If two such buffers are found, then the algorithm returns these two consecutive buffers. For notational convenience, for integers $a < b$ we let the tuple (a, b) denote a (approximate) buffer.

Lemma 5.6.8. *Let $i \in [n']$ and $j \in [d]$. There exist constants $\gamma < \delta_{\mathsf{b}} \in (0, 1/2)$ such that if $i \in \phi^{-1}(j)$ then Buff-Find finds buffers (a_1, b_1) and (a_2, b_2) such that $\mathsf{Dec}_{in}(c'[b_1+1, a_2-1]) = j \circ b_j$. Further, if $b_1 < i < a_2$ then Block-Decode outputs $j \circ b_j$.*

Proof. We first examine an uncorrupted block which has the form $B = 0^{\alpha\tau} \circ Y \circ 0^{\alpha\tau}$ for some $Y \in C_{in}$. Let $s = |B| = \beta\tau$ and note that $\tau = \log m$ and $|Y| = (\beta - 2\alpha)\tau$. Note also that $B[1, \alpha\tau] = B[s, s-\alpha\tau+1] = 0^{\alpha\tau}$ and $B[\alpha\tau+1, s-\alpha\tau] = Y$. We observe that approximate buffers (a, b) exist such that $b > \alpha\tau$ or $a < s - \alpha\tau + 1$; that is, an approximate buffer can cut into the codeword Y. We are interested in bounding how large this "cut" can be in a γ-good block and first examine how large this "cut" can be in an uncorrupted block.

Our inner code has the property that any interval of length $2\log(\alpha\tau)$ has at least fractional weight $\geqslant 2/5$. That is, an interval of length $2\log(\alpha\tau)$ in Y has at least $(4/5)\log(\alpha\tau)$ number of 1's. Also any approximate buffer has weight at most $(\delta_{\mathsf{b}}/2)\alpha\tau$. Let $\ell = c_0\tau$ for some constant c_0. We count the number of 1's in any $c_0\tau$ interval of Y. Note that in such an interval there are at most $c_0\tau/(2\log(\alpha\tau))$ disjoint intervals of length $2\log(\alpha\tau)$. Since the weight of each of these $2\log(\alpha\tau)$ intervals is at least $(4/5)\log(\alpha\tau)$ and the intervals are disjoint, we have that the weight of the interval $c_0\tau$ in Y is at least $c_0\tau/(2\log(\alpha\tau)) \cdot (4/5)\log(\alpha\tau) = (2/5)c_0\tau$. We pick c_0 such that $(2/5)c_0\tau \geqslant (\delta_{\mathsf{b}}/2)\alpha\tau + 1$; i.e., $c_0 = (5/4)\delta_{\mathsf{b}}\alpha + 1 \geqslant (5/4)\alpha\delta_{\mathsf{b}} + 5/(2\tau)$

(for large enough m since τ is an increasing function of m). On the other hand, we can have that an interval of length $(\delta_{\mathsf{b}}/2)\alpha\tau + 1$ in Y has $(\delta_{\mathsf{b}}/2)\alpha\tau + 1$ number of 1's.

The above derivation implies that largest "cut" an approximate buffer can make into the codeword Y from the start (i.e., indices after $\alpha\tau$) (and symmetrically the end; i.e., indices before $s - \alpha\tau + 1$) has size in the range $[(1 + \delta_{\mathsf{b}}/2)\alpha\tau, (1 + 5\delta_{\mathsf{b}}/2)\alpha\tau]$. This implies that there exists $b_1, a_2 \in \mathbb{N}$ such that $(1+\delta_{\mathsf{b}}/2)\alpha\tau \leqslant b_1 \leqslant (1+5\delta_{\mathsf{b}}/2)\alpha\tau$ and $(\beta - \alpha(1+5\delta_{\mathsf{b}}/2))\tau \leqslant a_2 \leqslant (\beta - \alpha(1 + \delta_{\mathsf{b}}/2))\tau$. Further b_1 and a_2 have the following properties: (1) $B[b_1 - \tau + 1, b_1]$ and $B[a_2, a_2+\tau-1]$ are approximate buffers; and (2) for every $i \in \{b_1-\tau+2, b_1-\tau+3, \ldots, a_2-1\}$, the window $B[i, i+\tau-1]$ is not an approximate buffer. These properties follow by our choice of c_0 and by the density property we have for our inner code C_{in}.

We obtained the above bounds on b_1, a_2 by analyzing an uncorrupted block B. We use this as a starting point for analyzing a γ-good block $\tilde{B}$ (i.e., $\mathsf{ED}(B, \tilde{B}) \leqslant \gamma\alpha\tau$. Let $s' = |\tilde{B}|$. Then by γ-good we have that $(1 - \alpha\gamma)s \leqslant s' \leqslant (1 + \alpha\gamma)s$. Now by γ-good, we have that the bounds obtained on b_1 and a_2 are perturbed by at most $\alpha\gamma\tau$. That is, we have in block $\tilde{B}$

$$(1 + \delta_{\mathsf{b}}/2 - \gamma)\alpha\tau \leqslant b_1 \leqslant (1 + 5\delta_{\mathsf{b}}/2 + \gamma)\alpha\tau$$
$$(\beta - \alpha(1 + 5\delta_{\mathsf{b}}/2 + \gamma))\tau \leqslant a_2 \leqslant (\beta - \alpha(1 + \delta_{\mathsf{b}}/2 - \gamma))\tau.$$

This gives us

$$\begin{aligned} a_2 - b_1 &\leqslant (\beta - \alpha(1 + \delta_{\mathsf{b}}/2 - \gamma))\tau - (1 + \delta_{\mathsf{b}}/2 - \gamma)\alpha\tau \\ &= (\beta - 2\alpha(1 + \delta_{\mathsf{b}}/2 - \gamma))\tau \\ a_2 - b_1 &\geqslant (\beta - \alpha(1 + 5\delta_{\mathsf{b}}/2 + \gamma))\tau - (1 + 5\delta_{\mathsf{b}}/2 + \gamma)\alpha\tau \\ &= (\beta - 2\alpha(1 + 5\delta_{\mathsf{b}}/2 + \gamma))\tau. \end{aligned}$$

Now we want to ensure decoding is possible on $c'[b_1 + 1, a_2 - 1]$. We observe that $(\beta - 2\alpha)\tau - (a_2 - b_1)$ is the number of insertion-deletion errors that are introduced because of the buffer finding algorithm. This quantity can be written as

$$(\delta_{\mathsf{b}} - 2\gamma)\alpha\tau \leqslant (\beta - 2\alpha)\tau - (a_2 - b_1) \leqslant (5\delta_{\mathsf{b}} + 2\gamma)\alpha\tau.$$

Note that since $|(\delta_{\mathsf{b}} - 2\gamma)\alpha\tau| \leqslant |(5\delta_{\mathsf{b}} + 2\gamma)\alpha\tau|$, we can correctly decode if γ and δ_{b} are chosen such that

$$(5\delta_{\mathsf{b}} + 2\gamma)\alpha\tau + \gamma\alpha\tau \leqslant \delta_{\mathsf{in}}(\beta - 2\alpha)\tau$$
$$\frac{(5\delta_{\mathsf{b}} + 3\gamma)\alpha}{\beta - 2\alpha} \leqslant \delta_{\mathsf{in}}.$$

To finish, we note that the constant η is chosen so that if $i \in \phi^{-1}(j)$ for any good block j then we have that $c'[\phi^{-1}(j)] \subset c'[i - \eta\tau, i + \eta\tau]$. Since the algorithm Buff-Find finds every δ_{b}-approximate buffer in the interval $c'[i - \kappa\tau, i + \kappa\tau]$ and since this interval contains γ-good block j, we have that the algorithm indeed Buff-Find returns approximate buffers (a_1, b_1) and (a_2, b_2) such that $\mathsf{Dec}_{in}(c'[b_1 + 1, a_2 - 1]) = j \circ Y^{(j)}$ if $b_1 < i < a_2$, thus proving the lemma. □

We now recall and prove Theorem 5.6.7.

Theorem 5.6.7. *The functions φ and ψ satisfy the following properties:*

1. *For any γ-good block j we have*

$$\Pr_{i \in \phi^{-1}(j)}[\varphi(i) \neq j] \leq \gamma.$$

2. *Let $[l, r)$ be an interval with closure $\mathcal{I}[L, R-1]$, satisfying that every block $j \in \{L, \ldots, R-1\}$ is γ-good. Then for every block j such that $\phi^{-1}(j) \subseteq [l, r)$, we have $\psi(j, l, r) = b_j$.*

Proof. First we analyze the probability $\Pr_{i \in \phi^{-1}(j)}[\varphi(i) \neq j]$. By Lemma 5.6.8 the algorithm Block-Decode on input i correctly outputs the block $Y^{(j)} \circ j$ if $i \in [b_1 + 1, a_2 - 1] \subset \phi^{-1}(j)$. Since j is γ-good and by Proposition 5.6.2 we have that $|\phi^{-1}(j)| \leqslant (\beta + \alpha\gamma)\tau$. Finally, by correctness of the decoder Dec_{in}, Lemma 5.6.8 gives us a lower bound on the distance $a_2 - b_1$. In particular,

$$a_2 - b_1 \geqslant (\beta - 2\alpha(1 + 5\delta_{\mathsf{b}}/2 + \gamma))\tau.$$

Thus we have that

$$\begin{aligned}\Pr_{i\in\phi^{-1}(j)}[\varphi(i)\neq j] &= 1-\Pr_{i\in\phi^{-1}(j)}[\varphi(i)=j] = 1-\frac{a_2-b_1}{|\phi^{-1}(j)|} \leqslant 1-\frac{a_2-b_1}{(\beta+\alpha\gamma)\tau}\\ &\leqslant 1-\frac{(\beta-2\alpha(1+5\delta_{\mathsf{b}}/2+\gamma))\tau}{(\beta+\alpha\gamma)\tau} = 1-\frac{\beta-2\alpha(1+5\delta_{\mathsf{b}}/2+\gamma)}{(\beta+\alpha\gamma)}\\ &\leqslant \frac{\alpha\gamma+6\alpha}{\beta+\alpha\gamma} \leqslant \frac{(\gamma+6)\,\alpha}{2} \leq \gamma,\end{aligned}$$

where we assumed that that $\delta_{\mathsf{b}} < 1/2$, $\gamma = 1/12$ and $\alpha \leq 2\gamma/(\gamma+6)$. More generally, there exists constants $\delta_{\mathsf{b}}, \gamma$, and α such that the above inequalities hold with $\alpha \leqslant 2\gamma/(\gamma+6)$.

For the second statement of Theorem 5.6.7, we analyze the algorithm Interval-Decode. Note we are only concerned with γ-good blocks which are wholly contained in the interval $[l, r)$. Let $\mathcal{I}[L, R-1]$ be the closure of $[l, r)$. We note that ϕ restricted to $\mathcal{I}[L, R-1]$ is a sub-decomposition which captures the errors introduced to blocks $L, \ldots, R-1$. The algorithm Interval-Decode is similar to the global buffer-finding algorithm of SZ codes applied to the interval $[l, r)$: it searches intervals of length $\alpha\tau$ in $\{l, l+1, \ldots, r-1\}$ from left to right until an approximate buffer $c'[i, i+\alpha\tau-1]$ is found. Then the algorithm marks it and continue scanning for approximate buffers, starting with left endpoint of the first new interval at the right endpoint of the presumed buffer. Then once the whole interval has been scanned, the algorithm finds pairs of consecutive buffers which are far apart and attempts to decode the section of the block that falls between these two buffers.

According to the analysis of the SZ buffer finding algorithm, as long as block j and $j+1$ are γ-good (for small enough constant γ), the buffers surrounding block $j+1$ should be located approximately correctly, and block j will appear close to a codeword. Since every block in the closure of $[l, r)$ is γ-good, all the buffers in this interval should be located approximately correctly, and every block j such that $\phi^{-1}(j) \subseteq [l, r)$ should be decoded properly. Therefore there will be exactly one block decoded to (j, b) and it must hold that $b = b_j$.

There is one minor issue with the above argument. The searching process starts from an index l which does not necessarily align with the left boundary of $\mathcal{I}[L, R-1]$. However, we note that this only affects the location of the first approximate buffer, and all subsequent buffers are going to be consistent with what the algorithm would have found if it started

from the left boundary of $\mathcal{I}[L, R-1]$. In order to decode the first block, Interval-Decode performs another SZ buffer finding algorithm, but *from right to left*, and decodes the leftmost block. □

5.6.6 Parameter Setting and Proof of Theorem 5.6.1

In this section we list a set of constraints which our setting of parameters must satisfy, and then complete the proof of Theorem 5.6.1. These constraints are required by different parts of the analysis. Recall that $\delta_{out}, \delta_{in} \in (0,1)$ and $\beta_{in} \geq 1$ are given as parameters of the outer code and the inner code, and that $\beta = 2\alpha + \beta_{in}(1 + \log|\Sigma|)$. We have that $\beta \geq 2$ for any non-negative α.

Proposition 5.6.4. *There exists constants $\gamma, \theta \in (0,1)$ and $\alpha = \Omega(\delta_{in})$ such that the following constraints hold:*

1. $\gamma \leq 1/12$ *and* $\theta < 1/50$;
2. $(\beta + \gamma)/(\beta - \gamma) < 4/3$;
3. $\alpha \leq 2\gamma/(\gamma + 6)$;
4. $\alpha(1 + 3\gamma)/(\beta - 2\alpha) < \delta_{in}$.

Proof. For convenience of the reader and simplicity of the presentation we work with explicit values and verify that they satisfy the constraints in Proposition 5.6.4. Let $\gamma = 1/12$ and $\theta = 1/51$, which satisfies constraint (1). Note that $\gamma < 2/7 \leq \beta/7$, hence

$$\frac{\beta + \gamma}{\beta - \gamma} < \frac{4}{3}$$

and constraint (2) is satisfied. We take $\alpha = 2\gamma\delta_{in}/(\gamma + 6)$ so that $\alpha = \Omega(\delta_{in})$ and constraint (3) is satisfied. Note also that $\beta - 2\alpha = \beta_{in}(1 + \log|\Sigma|) \geq 2$ which implies

$$\frac{\alpha(1 + 3\gamma)}{\beta - 2\alpha} \leq \frac{\alpha(1 + 3\gamma)}{2} = \frac{\alpha(\gamma + 3\gamma^2)}{2\gamma} < \frac{\alpha(\gamma + 6)}{2\gamma} = \delta_{in}.$$

Therefore, constraint (4) is also satisfied. □

We let

$$\delta = \frac{\delta_{out}\alpha\gamma}{2\beta(1+1/\theta)} = \Omega\left(\delta_{in}\delta_{out}\right).$$

We now recall and prove Theorem 5.6.1, which shows Theorem 5.2.1.

Theorem 5.6.1. *Let C_{out} and C_{in} be codes such that*

- *C_{out} defined by $\mathsf{Enc}_{out}\colon \Sigma^k \to \Sigma^m$ is an a $(\ell_{out}, \delta_{out}, \epsilon_{out})$-Hamming LDC.*
- *C_{in} is family of binary polynomial-time encodable/decodable codes with rate $1/\beta_{in}$ capable of correcting δ_{in} fraction of insertion-deletion errors. In addition, there are constants $\alpha_1, \alpha_2 \in (0,1)$ such that for any codeword c of C_{in}, any substring of c with length at least $\alpha_1|c|$ has fractional Hamming weight at least α_2.*

Then $C(C_{out}, C_{in})$ is a binary $\left(\ell_{out}\cdot O(\log^4(n')), \Omega(\delta_{out}\delta_{in}), \epsilon - \mathsf{negl}(n')\right)$-InsDel LDC. Here the codewords of C have length $n = \beta m$ where $\beta = O(\beta_{in}\log(|\Sigma|))$, and n' denotes the length of received word.

Proof. Recall that the decoder Dec works as follows. Given input index $i \in [k]$ and oracle access to $c' \in \{0,1\}^{n'}$, $\mathsf{Dec}^{c'}(i)$ simulates $\mathsf{Dec}^{s'}_{out}(i)$. Whenever $\mathsf{Dec}^{s'}_{out}(i)$ queries an index $j \in [m]$, the decoder expresses $j = (p-1)\tau + r_j$ for $p \in [d]$ and $0 \le r_j < \tau$, and runs Figure 5.1 on input $(c', 1, n'+1, p)$ to obtain a τ-long string b'_p. Then it feeds the (r_j+1)-th symbol of b'_p to $\mathsf{Dec}^{s'}_{out}(i)$. At the end of the simulation, $\mathsf{Dec}^{c'}(i)$ returns the output of $\mathsf{Dec}^{s'}_{out}(i)$.

For $p \in [d]$, let $b'_p \in \Sigma^\tau \cup \{\perp\}$ be a random variable that has the same distribution as the output of Figure 5.1 on input $(c', 1, n'+1, p)$. Define a random string $s' \in (\Sigma \cup \{\perp\})^m$ as follows. For every $i \in [m]$ such that $i = (p-1)\tau + r$ for $p \in [d]$ and $0 \le r < \tau$,

$$s'[i] = \begin{cases} b'_p[r] & \text{if } b'_p \neq \perp, \\ \perp & \text{if } b'_p = \perp. \end{cases}$$

Since $b'_p = b_p$ implies $s'[(p-1)\tau + r] = s[(p-1)\tau + r]$ for all $0 \le r < \tau$, the event $E_s := \{\Pr_{j\in[m]}[s'[j] \neq s[j]] \le \delta_{out}\}$ is implied by the event $E_b := \{\Pr_{j\in[d]}\left[b'_j \neq b_j\right] \le \delta_{out}\}$. Theorem 5.6.5 implies that $\Pr[E_s] \ge \Pr[E_b] \ge 1 - \mathsf{negl}(n')$. According to the construction

of Dec, from the perspective of the outer decoder, the string s' is precisely the string it is interacting with. Hence by properties of Dec_{out} we have that

$$\forall i \in [k], \quad \Pr\left[\mathsf{Dec}_{out}^{s'}(i) = x[i] \;\middle|\; E_s\right] \geq \frac{1}{2} + \varepsilon_{out}.$$

Therefore by construction of Dec we have

$$\begin{aligned}\forall i \in [k], \quad \Pr\left[\mathsf{Dec}^{c'}(i) = x[i]\right] &\geq \Pr[E_s] \cdot \Pr\left[\mathsf{Dec}_{out}^{s'}(i) = x[i] \;\middle|\; E_s\right] \\ &\geq (1 - \mathsf{negl}(n')) \cdot \left(\frac{1}{2} + \varepsilon_{out}\right) \geq \frac{1}{2} + \varepsilon_{out} - \mathsf{negl}(n').\end{aligned}$$

The query complexity of Dec is $\ell_{out} \cdot O\left(\log^4 n'\right)$ since it makes ℓ_{out} calls to Figure 5.1, which by Proposition 5.6.3 has query complexity $O\left(\log^4 n'\right)$. □

5.7 Private/Resource-Bounded Insertion-Deletion LDCs from Private/Resource-Bounded Hamming LDCs

We show that our compiler extends to both the private and resource-bounded LDC settings, yielding private and resource-bounded LDCs for insertion-deletions errors. To the best of our knowledge, this is the first such result constructing LDCs for insertion-deletion errors in this setting.

5.7.1 Abstraction of Theorem 5.2.1

We present the following abstraction of Theorem 5.6.1 that describes the key properties of the compiler.

Lemma 5.7.1. *There exist functions* $\mathsf{Compile}$ *and* $\mathsf{Recover}$ *such that for any constant* $\delta > 0$ *and any Hamming LDC* $C[K, k, q_1, q_2] = (\mathsf{Enc}, \mathsf{Dec})$ *with locality* ℓ*, there exists* $\delta_{\mathsf{f}} = \Theta(\delta)$ *such that for any message* x *and any* c' *with* $\mathsf{ED}(c', y) \leqslant \delta_{\mathsf{f}}$ *for* $y = \mathsf{Compile}(\mathsf{Enc}(x)) \in \{0,1\}^*$*:*

1. $\mathsf{Dec}_{\mathsf{f}}$ *has locality* $\ell \cdot O(\log^4(K \cdot \log(q_2)))$ *and* $|y| = \Theta(K \cdot \log(q_2))$*;*
2. *For* $c'' = \mathsf{Recover}^{Y'}(1) \circ \cdots \circ \mathsf{Recover}^{Y'}(K \cdot \log(q_2))$*, we have* $\Pr\left[\mathsf{Dec}_{\mathsf{f}}^{c'}(i) = x_i\right] \geqslant \Pr\left[\mathsf{Dec}^{c''}(i) = x_i\right] - \vartheta_1(K \cdot \log(q_2))$*; and*

3. *If* $\mathsf{ED}(c', \mathsf{Enc}_{\mathsf{f}}(x)) \leqslant \delta_{\mathsf{f}}$ *then* $\mathsf{HAM}(c'', \mathsf{Enc}(x)) \leqslant \delta$ *except with probability* $\vartheta_2(K \cdot \log(q_2))$.

Here, ϑ_1 *and* ϑ_2 *are fixed negligible functions,* $\mathsf{Compile}$ *is computable in parallel time* $\mathrm{polylog}(K)$*, and* c'' *is computable in parallel time* $\mathrm{polylog}(K)$.

5.7.2 One-Time Private InsDel LDCs

We prove Theorem 5.2.4. We restate the theorem for completeness.

Theorem 5.2.4 ([46])**.** *Let* $C[K, k, \lambda, q_1, q_2]$ *be a* $(\ell, \delta, p, \varepsilon)$*-one time private Hamming LDC for constant* $\delta > 0$*. Then there exists a binary code* $C_{\mathsf{f}}[n, k, \lambda, q_1, 2]$ *that is a* $(\ell_{\mathsf{f}}, \delta_{\mathsf{f}}, p_{\mathsf{f}}, \varepsilon_{\mathsf{f}})$*-one time private InsDel LDC, where* $\ell_{\mathsf{f}} = \ell \cdot O(\log^4(n))$*,* $\delta_{\mathsf{f}} = \Theta(\delta)$*,* $p_{\mathsf{f}} < p$*,* $\varepsilon_{\mathsf{f}} = \varepsilon/(1 - (p_{\mathsf{f}}/p) - (\vartheta_1(n)/p) - \vartheta_2(n))$*, and* $n = \Theta(K \cdot \log(q_2))$*. Here* ϑ_1, ϑ_2 *are fixed negligible functions.*

Proof. Let $C[K, k, q_1, q_2, \lambda] = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be a $(\ell, \rho, p, \varepsilon, \mathsf{HAM})$-one time private Hamming LDC. We define $\mathsf{Gen}_{\mathsf{f}}(1^\lambda) := \mathsf{Gen}(1^\lambda)$. Then for any message x and secret key sk we define $\mathsf{Enc}_{\mathsf{f}}(x, \mathsf{sk}) := \mathsf{Compile}(\mathsf{Enc}(x, \mathsf{sk}))$. Fixing the secret key sk and applying Lemma 5.7.1 to the encoding scheme, we see that $\mathsf{Dec}_{\mathsf{f}}$ has locality $\ell \cdot O(\log^4(n))$ and the output length of $\mathsf{Enc}_{\mathsf{f}}$ is $n = \Theta(K \log q_2)$ bits. Next, we obtain $\mathsf{Enc}_{\mathsf{f}} \colon \Sigma_1^k \times \{0,1\}^* \to \{0,1\}^n$ and $\mathsf{Dec}_{\mathsf{f}} \colon \{0,1\}^{\log k} \times \{0,1\}^* \to \Sigma_1$ by applying Lemma 5.7.1 to the code C, with the modification that both the encoder $\mathsf{Enc}_{\mathsf{f}}$ and $\mathsf{Dec}_{\mathsf{f}}$ additionally receive a secret key $\mathsf{sk} \leftarrow \mathsf{Gen}_{\mathsf{f}}(1^\lambda)$ as input. The main challenge is proving the security. Suppose towards contradiction that there exists an adversary $\mathcal{A}_{\mathsf{f}}$ such that $\Pr[\mathsf{Fool}(\mathcal{A}_{\mathsf{f}}(Y), \delta_{\mathsf{f}}, p_{\mathsf{f}}, \mathsf{sk}, x, Y) = 1] > \varepsilon_{\mathsf{f}}$ for $Y \leftarrow \mathsf{Enc}_{\mathsf{f}}(x, \mathsf{sk})$. Then we construct an adversary $\mathcal{A}$ such that $\Pr[\mathsf{Fool}(\mathcal{A}(y), \rho, p, \mathsf{sk}, x, y) = 1] > \varepsilon$ for $y \leftarrow \mathsf{Enc}(x, \mathsf{sk})$. Adversary $\mathcal{A}$ works as follows:

1. $\mathcal{A}$ obtains as input $x, y, \lambda, \rho, p, k$, and K, where $y = \mathsf{Enc}(x, \mathsf{sk})$;
2. $\mathcal{A}$ then obtains $Y = \mathsf{Compile}(y)$; and
3. $\mathcal{A}$ then obtains $Y' \leftarrow \mathcal{A}_{\mathsf{f}}(x, Y, \lambda, \delta_{\mathsf{f}}, p_{\mathsf{f}}, k, n)$.

By assumption $\mathsf{ED}(Y, Y') \leqslant \delta_{\mathsf{f}}$ and with probability at least ε_{f} there exists $i \in [k]$ such that

$$\Pr[\mathsf{Dec}_{\mathsf{f}}^{Y'}(i, \mathsf{sk}) = x_i] < p_{\mathsf{f}}.$$

$\mathcal{A}$ then outputs word

$$y' = \mathsf{Recover}^{Y'}(1) \circ \cdots \circ \mathsf{Recover}^{Y'}(K \cdot \log(q_2)).$$

Suppose that $\mathsf{Fool}(Y', \delta_\mathsf{f}, p_\mathsf{f}, \mathsf{sk}, x, Y) = 1$. Then we have that $\mathsf{ED}(Y, Y') \leqslant \delta_\mathsf{f}$ and there exists $i \in [k]$ such that

$$\Pr\left[\mathsf{Dec}_\mathsf{f}^{Y'}(i, \mathsf{sk}) = x_i\right] < p_\mathsf{f}.$$

By Lemma 5.7.1, we have that $\mathsf{HAM}(y, y') \leqslant \rho$ with probability at least $1 - \vartheta_2(n)$. By definition of Dec_f and Lemma 5.7.1, we have that

$$p_\mathsf{f} > \Pr\left[\mathsf{Dec}_\mathsf{f}^{Y'}(i, \mathsf{sk}) = x_i\right] \geqslant \Pr\left[\mathsf{Dec}^{y'}(i, \mathsf{sk}) = x_i\right] - \vartheta_1(n),$$

where the randomness of the second term is taken over the coins of Dec and the coins used by $\mathsf{Recover}$ to generate y', and the randomness of the first term is taken only over the coins of Dec_f. Define the predicate $B_p(y') = 1$ if and only if $\Pr\left[\mathsf{Dec}^{y'}(i, \mathsf{sk}) = x_i\right] < p$, and $B_p(y') = 0$ otherwise, where the probability is taken over Dec's coins. Let $\alpha = \Pr\left[B_p(y')\right]$, where the probability is taken over the random coins used to generate y' from Y'. Then we have that

$$\Pr\left[\mathsf{Dec}^{y'}(i, \mathsf{sk}) = x_i\right] \geqslant p(1 - \alpha).$$

This implies that $\alpha > 1 - (p_\mathsf{f}/p) - (\vartheta_1(n)/p)$. Now consider two events

$$\mathcal{F}_\mathsf{HAM} = \mathsf{Fool}(y', \rho, p, \mathsf{sk}, x, y)$$
$$\mathcal{F}_\mathsf{ED} = \mathsf{Fool}(Y', \delta_\mathsf{f}, p_\mathsf{f}, \mathsf{sk}, x, Y).$$

Then

$$\Pr\left[\mathcal{F}_\mathsf{HAM} = 1\right] \geqslant \Pr\left[\mathcal{F}_\mathsf{ED} = 1\right] \cdot \Pr\left[\mathcal{F}_\mathsf{HAM} = 1 | \mathcal{F}_\mathsf{ED} = 1\right].$$

By assumption we have that $\Pr[\mathcal{F}_{\mathsf{ED}} = 1] > \varepsilon_{\mathsf{f}}$. Further, by Definition 5.5.1, $\mathcal{F}_{\mathsf{HAM}} = 1$ if and only if $\mathsf{HAM}(y, y') \leqslant \rho$ and there exists $i \in [k]$ such that $\Pr[\mathsf{Dec}^{y'}(i, \mathsf{sk}) = x_i] < p$. Since $\mathcal{F}_{\mathsf{ED}} = 1$, we have that $\mathsf{ED}(Y, Y') \leqslant \delta_{\mathsf{f}}$, and thus by Lemma 5.7.1 we have that $\mathsf{HAM}(y, y') \leqslant \rho$ with probability at least $1 - \vartheta_2(n)$. Thus

$$\Pr[\mathcal{F}_{\mathsf{HAM}} = 1 | \mathcal{F}_{\mathsf{ED}} = 1] \geqslant 1 - \vartheta_2(n) - (1 - \alpha)$$

and $\alpha > 1 - (p_{\mathsf{f}}/p) - (\vartheta_1(n)/p)$. Therefore we have that

$$\Pr[\mathcal{F}_{\mathsf{HAM}} = 1] > \varepsilon_{\mathsf{f}} \cdot (1 - (p_{\mathsf{f}}/p) - (\vartheta_1(n)/p) - \vartheta_2(n)),$$

which is a contradiction since the right hand side of the above equation is equal to ε. □

5.7.3 Resource-Bounded InsDel LDCs

To construct LDCs for resource-bounded InsDel channels, we first need to introduce the notion of *closure* between algorithms classes. Let $\mathbb{C}$ be a class of parallel algorithms running in at most sequential time T and maximum space usage S. For any $A \in \mathbb{C}$, let $B = \mathsf{reduce}(A)$ be a reduction from algorithm A to B. We say the class of algorithms $\mathbb{C}'$ is the *closure of* $\mathbb{C}$ *with respect to* reduce if $\mathbb{C}'$ is the minimum class of algorithms such that $\mathsf{reduce}(A) \in \mathbb{C}'$ for all $A \in \mathbb{C}$.

In our context, for parameter N we define reduce_N as a sequential time $N \cdot \mathrm{polylog}(N)$ reduction that can be executed in parallel for sequential time $\mathrm{polylog}(N)$. Parallel execution incurs an additional $N \cdot \mathrm{polylog}(N)$ space overhead, and sequential execution incurs an additional $\mathrm{polylog}(N)$ space overhead. Thus, if $\mathbb{C}$ is the class of all parallel PPT algorithms running in sequential time T, then $\overline{\mathbb{C}}(N)$ is some class of parallel PPT algorithms running in time $T + \mathrm{polylog}(N)$.

We now recall and prove Theorem 5.2.6.

Theorem 5.2.6 ([46])**.** *Let $\mathbb{C}$ be the class of parallel PPT algorithms running in sequential time T and space S, and let $C[K, k, q_1, q_2]$ be a $(\ell, \delta, p, \varepsilon, \overline{\mathbb{C}}(n))$-Hamming LDC for constant $\rho, p > 0$ and $n = O(K \cdot \log(q_2))$. There exists a binary code $C_{\mathsf{f}}[n, k, q_1, 2]$ that is a $(\ell_{\mathsf{f}}, \delta_{\mathsf{f}}, p_{\mathsf{f}}, \varepsilon_{\mathsf{f}}, \mathbb{C})$-*

InsDel LDC against class $\mathbb{C}$, where $\ell_{\mathsf{f}} = \ell \cdot O(\log^4(n))$, $\delta_{\mathsf{f}} = \Theta(\delta)$, $p_{\mathsf{f}} < p$, and $\varepsilon_{\mathsf{f}} = \varepsilon/(1-(p_{\mathsf{f}}/p)-(\vartheta_1(n)/p)-\vartheta_2(n))$. Here ϑ_1, ϑ_2 are fixed negligible functions.

Proof. The proof follows nearly identically to the proof of Theorem 5.2.4; namely, we obtain C_{f} in an identical manner by using the compiler of Lemma 5.7.1 with the code C defined above. The main challenge again is the security proof: given adversary $\mathcal{A}_{\mathsf{f}} \in \mathbb{C}$ such that $\Pr[\mathsf{Fool}(\mathcal{A}(Y), \delta_{\mathsf{f}}, p_{\mathsf{f}}, x, Y) = 1] > \varepsilon_{\mathsf{f}}$ for $Y \leftarrow \mathsf{Enc}_{\mathsf{f}}(x)$, we construct an adversary $\mathcal{A} \in \overline{\mathbb{C}}(n)$ such that $\Pr[\mathsf{Fool}(\mathcal{A}(y), \rho, p, x, y) = 1] > \varepsilon$ for $y \leftarrow \mathsf{Enc}(x)$. Adversary $\mathcal{A}$ is constructed identically as in the proof of Theorem 5.2.4, except now the constructed adversary only yields a contradiction if we can show that $\mathcal{A} \in \overline{\mathbb{C}}(n)$. By Lemma 5.7.1, we have that $\mathsf{Compile}$ is a $\mathrm{polylog}(K) = \mathrm{polylog}(n)$ parallel time algorithm, and $y' = \mathsf{Recover}^{Y'}(1) \circ \cdots \circ \mathsf{Recover}^{Y'}(K \log(q_2))$ is computable in $\mathrm{polylog}(n)$ parallel time. Finally, $\mathsf{Compile}$ and $\mathsf{Recover}$ are run independent of the adversary $\mathcal{A}_{\mathsf{f}}$, we have that the total parallel time of $\mathcal{A}$ is $T + \mathrm{polylog}(n)$, which implies $\mathcal{A} \in \overline{\mathbb{C}}(n)$, yielding our contradiction. □

5.A Hamming-to-InsDel Locally Decodable Code Compiler

5.B Proof of Theorem 5.6.6

We first recall Theorem 5.6.6.

Theorem 5.6.6. *If $j \in [d]$ is a (θ, γ)-locally good block, running Figure 5.1 on input $(c', 1, n'+1, j)$ outputs b_j with probability at least $1 - \mathsf{negl}(n')$.*

We emphasize that the exact boundaries of any block $\phi^{-1}(j)$ or interval $\mathcal{I}[L, R]$ are not known to the binary search algorithm, so it cannot do uniform sampling within the exact boundaries. Instead, as we can see in Figure 5.1, in each iteration it picks two indices m_1, m_2 and calls Block-Decode on uniformly sampled indices in $\{m_1, m_1+1, \cdots, m_2-1\}$. Depending on the results returned by Block-Decode, it either sets $l = m_1$ or $r = m_2$ and recursively search in the smaller interval $[l, r)$.

The following lemma shows that as long as the closure of an interval $[l, r)$ is (θ, γ)-good, uniform samples from $[l, r)$ does now perform much worse than uniform samples from a good block in terms of estimating ϕ.

Lemma 5.B.1. *Let $[l, r)$ be an interval with closure $\mathcal{I}[L, R-1]$. Suppose $\mathcal{I}[L, R-1]$ is a (θ, γ)-good interval. We have*

$$\Pr_{i\in[l,r)}[\varphi(i) \neq \phi(i)] \leq \gamma + \theta + \frac{\gamma}{\beta}.$$

Proof. Let $\mathsf{Good} \subseteq \{L+1, \dots R-2\}$ be the set of γ-good blocks among $\{L+1, \dots, R-2\}$, and let $\overline{\mathsf{Good}} = \{L+1, \dots, R-2\} \setminus \mathsf{Good}$. By definition of (θ, γ)-goodness we have $\overline{\mathsf{Good}} \leq \theta(R-L)$. Since for each $j \in \mathsf{Good}$, $\phi^{-1}(j) \subseteq [l, r)$. We can apply item (1) of Theorem 5.6.7 and get

$$\Pr_{i\in[l,r)}[\varphi(i) \neq \phi(i) \mid \phi(i) \in \mathsf{Good}] \leq \gamma.$$

Now we have the bound

$$\begin{aligned}\Pr_{i\in[l,r)}[\varphi(i) \neq \phi(i)] &\leq \Pr_{i\in[l,r)}[\varphi(i) \neq \phi(i) \mid \phi(i) \in \mathsf{Good}] + \Pr_{i\in[l,r)}[\phi(i) \notin \mathsf{Good}] \\ &\leq \gamma + \Pr_{i\in[l,r)}[\phi(i) \notin \mathsf{Good}],\end{aligned}$$

so it suffices to upper bound $\Pr_{i\in[l,r)}[\phi(i) \notin \mathsf{Good}]$. Denote $\Delta_j = |\phi^{-1}(j)| - \beta\tau$. It holds that $\sum_{j=L}^{R-1} |\Delta_j| \leq \gamma(R-L)\tau$. In particular $\sum_{j\in\mathsf{Good}} \Delta_j \geq -\gamma(R-L)\tau$ and $\sum_{j\in\overline{\mathsf{Good}}} \Delta_j \leq \gamma(R-L)\tau$. We have

$$\begin{aligned}\Pr_{i\in[l,r)}[\phi(i) \notin \mathsf{Good}] &\leq \frac{\sum_{j\notin\mathsf{Good}} |\phi^{-1}(j)|}{r-l} \leq \frac{\theta(R-L)\beta\tau + \sum_{j\notin\mathsf{Good}} \Delta_j}{(R-L)\beta\tau + \sum_{j\in\mathsf{Good}} \Delta_j + \sum_{j\notin\mathsf{Good}} \Delta_j} \\ &\leq \frac{\theta(R-L)\beta\tau + \gamma(R-L)\tau}{(R-L)\beta\tau} = \theta + \frac{\gamma}{\beta}.\end{aligned}$$

Hence the lemma follows. □

In the following, we set $\rho = \min\{\frac{1}{4} \cdot \frac{\beta-\gamma}{\beta+\gamma}, 1 - \frac{3}{4} \cdot \frac{\beta+\gamma}{\beta-\gamma}\}$ as in Figure 5.1. Note that by item (2) of Proposition 5.6.4 we have $\rho > 0$.

The following lemma states that any interval not too far from a locally good block is also good.

Lemma 5.B.2. *Let $l, r \in [n']$ be such that $r - l \geq 18(\beta + \gamma)\tau$. Let $\mathcal{I}[L, R-1]$ be the closure of $[l, r)$. Set $m_1 = (1-\rho)l + \rho r$ and $m_2 = \rho l + (1-\rho)r$ and let $\mathcal{I}[M_1, M_2 - 1]$ be the closure of $[m_1, m_2)$. Suppose for some $L \leq x \leq M_1$ block x is (θ, γ)-locally-good. Then we have*

1. *$M_1 \leq L + (R-L)/3$, $M_2 \geq L + 2(R-L)/3$.*
2. *$\mathcal{I}[M_1, M_2 - 1]$ is a $(2\theta, 2\gamma)$-good interval.*

Proof. Since $L \leq x \leq R-1$ and block x is (θ, γ)-locally good, by definition $\mathcal{I}[L, R-1]$ is a (θ, γ)-good interval. From the inclusion $[l, r) \subseteq \mathcal{I}[L, R-1]$ we know that

$$(R-L)(\beta + \alpha\gamma)\tau \geq |\mathcal{I}(L, R-1)| \geq r - l \geq 18(\beta + \alpha\gamma)\tau,$$

which implies $R - L \geq 18$.

We begin by proving item (1).

Claim 5.B.3. *$M_1 \leq L + (R-L)/3$.*

Proof. Suppose $M_1 > L + (R-L)/3$. From the inclusion $\mathcal{I}[L+1, M_1 - 1] \subseteq [l, m_1)$, we have

$$\begin{aligned}\rho(r-l) = m_1 - l \geq |\mathcal{I}[L+1, M_1 - 1]| &\geq (M_1 - L - 1)(\beta - \alpha\gamma)\tau \\ &> \frac{1}{4}(R-L) \cdot (\beta - \alpha\gamma)\tau.\end{aligned}$$

The last inequality holds as long as $R - L \geq 12$. Similarly, from the inclusion $[l, r) \subseteq \mathcal{I}[L, R-1]$, we have that

$$r - l \leq |\mathcal{I}[L, R-1]| \leq (R-L)(\beta + \alpha\gamma)\tau.$$

This implies $\rho > \frac{1}{4} \cdot \frac{\beta - \gamma}{\beta + \gamma}$ which is a contradiction. □

Claim 5.B.4. *$M_2 \geq L + 2(R-L)/3$.*

Proof. Suppose $M_2 < L + 2(R-L)/3$. From the inclusion $[l, m_2) \subseteq \mathcal{I}[L, M_2 - 1]$, we have

$$(1-\rho)(r-l) = m_2 - l \leq |\mathcal{I}[L, M_2 - 1]| \leq (M_2 - L)(\beta + \alpha\gamma)\tau$$

$$< \frac{3}{4}(R-L-2)\cdot(\beta+\alpha\gamma)\tau.$$

The last inequality holds as long as $R-L \geq 18$. Similarly, from the inclusion $\mathcal{I}[L+1, R-2] \subseteq [l, r)$, we have that

$$r - l \geq |\mathcal{I}[L+1, R-2]| \geq (R-L-2)(\beta-\alpha\gamma)\tau.$$

This implies $1-\rho < \frac{3}{4}\cdot\frac{\beta+\gamma}{\beta-\gamma}$ which is a contradiction. □

An immediate consequence of item (1) is that $M_2 - L \leq 2(M_2 - M_1)$. Therefore, by (θ,γ)-locally-goodness of x, we have

$$\begin{aligned}\sum_{j=M_1}^{M_2-1} \mathsf{ED}\left(c'[\phi^{-1}(j)], \tilde{Y}^{(j)}\right) &\leq \sum_{j=x}^{M_2-1} \mathsf{ED}\left(c'[\phi^{-1}(j)], \tilde{Y}^{(j)}\right)\\ &\leq \gamma\cdot(M_2-L)\tau\\ &\leq 2\gamma\cdot(M_2-M_1)\tau.\end{aligned}$$

Similarly, the number of 2γ-bad blocks among $\{M_1,\cdots,M_2-1\}$ is at most the number of γ-bad blocks among $\{L,\cdots,M_2-1\}$, which is upper bounded by $\theta(M_2-L) \leq 2\theta(M_2-M_1)$. Therefore the interval $\mathcal{I}[M_1, M_2-1]$ is $(2\theta, 2\gamma)$-good. □

The following is the main lemma we use to prove Theorem 5.6.6.

Lemma 5.B.5. *Assume $j \in [d]$ is a (θ,γ)-locally-good block. Denote by $l^{(t)}$, $r^{(t)}$ the values of l, r at beginning of the t-th iteration when running Figure 5.1 on input $(c', 1, n'+1, j)$. Suppose $r^{(t)} - l^{(t)} \geq 36(\beta+\gamma)\tau$. Then we have*

$$\Pr\left[\phi^{-1}(j) \subseteq \left[l^{(t+1)}, r^{(t+1)}\right) \;\middle|\; \phi^{-1}(j) \subseteq \left[l^{(t)}, r^{(t)}\right)\right] \geq 1 - \mathsf{negl}(n'),$$

where the probability is taken over the randomness of the algorithm.

Proof. Let m_1 and m_2 be defined as in Figure 5.1. Let $\mathcal{I}[L, R-1]$ be closure of $[l^{(t)}, r^{(t)}]$, and let $\mathcal{I}[M_1, M_2-1]$ be the closure of $[m_1, m_2)$. Since we always have $[m_1, m_2) \subseteq [l^{(t+1)}, r^{(t+1)})$,

$\phi^{-1}(j) \subseteq [m_1, m_2)$ would immediately imply $\phi^{-1}(j) \subseteq [l^{(t+1)}, r^{(t+1)})$. In the rest of the proof, we assume $\phi^{-1}(j) \not\subseteq [m_1, m_2)$, which means $L \le j \le M_1$ or $M_2 - 1 \le j \le R$.

We may assume that $L \le j \le M_1$ since the other case $M_2 - 1 \le j \le R$ is completely symmetric. The condition $r^{(t)} - l^{(t)} \ge 36(\beta+\gamma)\tau$ implies that $R - L \ge 36$, and that $m_2 - m_1 \ge \left(r^{(t)} - l^{(t)}\right)/2 \ge 18(\beta+\gamma)\tau$. Therefore we can apply Lemma 5.B.2 to m_1, m_2 and get that (1) $M_2 - M_1 \ge (R-L)/3 \ge 12$, and (2) $\mathcal{I}[M_1, M_2 - 1]$ is a $(2\theta, 2\gamma)$-good interval. Since $\mathcal{I}[M_1, M_2 - 1]$ is the closure of $[m_1, m_2)$, Lemma 5.B.1 gives

$$\Pr_{i \in [m_1, m_2)} [\varphi(i) \ne \phi(i)] \le 2\gamma + 2\theta + \frac{2\gamma}{\beta} < \frac{1}{4} + \frac{1}{25} < \frac{1}{3},$$

where the second last inequality is because $\theta < 1/50$, $\gamma \le 1/12$ (i.e., item (1) of Proposition 5.6.4) and $\beta \ge 2$.

Let $i_1, i_2, \cdots, i_N$ be the samples drawn by Figure 5.1, which are independent and uniform samples from $[m_1, m_2)$. Define X_j to be the indicator random variable of the event $\{\varphi(i_j) = \perp\} \cup \{\varphi(i_j) < x\}$, and define Y_j to be the indicator random variable of the event $\{\varphi(i_j) \ne \phi(i_j)\}$. It follows that $\mathrm{E}[Y_j] < 1/3$, and $\phi^{-1}(x) \not\subseteq \left[l^{(t+1)}, r^{(t+1)}\right)$ if and only if $\sum_{j=1}^{N} X_j \ge N/2$. Therefore it suffices to upper bound the probability of the latter event.

We observe that if $i \in [m_1, m_2) \subseteq \mathcal{I}[M_1, M_2]$, then $\phi(i) \ge M_1 \ge j$. Therefore $\varphi(i) = \phi(i)$ implies $\varphi(i) \ge j$, or in other words $X_j \le Y_j$. An application of Chernoff bound gives

$$\begin{aligned}\Pr\left[\sum_{j=1}^{N} X_j \ge \frac{N}{2}\right] &\le \Pr\left[\sum_{j=1}^{N} Y_j \ge \frac{N}{2}\right] \le \Pr\left[\sum_{j=1}^{N} Y_j \ge \left(1 + \frac{1}{2}\right) \sum_{j=1}^{N} \mathrm{E}[Y_j]\right] \\ &\le \exp\left(-\frac{N}{36}\right).\end{aligned}$$

Taking $N = \Theta(\log^2 n')$ gives $\Pr\left[\sum_{j=1}^{N} X_j \ge \frac{N}{2}\right] \le \exp\left(-\Theta\left(\log^2 n'\right)\right) = \mathsf{negl}(n')$. □

We are now ready to prove Theorem 5.6.6.

Proof of Theorem 5.6.6. Let $C = 36(\beta+\gamma)\tau$ be defined as in Figure 5.1, and $T = O(\log n')$ be the number of iterations until $r - l \le C$. Denote by $l^{(t)}$, $r^{(t)}$ the values of l, r at beginning

of the t-th iteration. Let $\mathbf{b}$ be the random variable denoting the output of Figure 5.1. We have

$$\Pr[\mathbf{b} = b_j] \geq \Pr\left[\phi^{-1}(j) \subseteq [l^{(T)}, r^{(T)})\right] \cdot \Pr\left[\mathbf{b} = b_j \;\middle|\; \phi^{-1}(j) \subseteq [l^{(T)}, r^{(T)})\right]. \tag{5.2}$$

We are going to lower bound both probabilities in the right-hand-side of (Equation (5.2)).

According to the algorithm, we have the following inclusion chain

$$[l^{(T)}, r^{(T)}) \subseteq [l^{(T-1)}, r^{(T-1)}) \subseteq \cdots \subseteq [l^{(1)}, r^{(1)}),$$

where $l^{(1)} = 1$ and $r^{(1)} = n' + 1$. By Lemma 5.B.5, it holds that

$$\begin{aligned}\Pr\left[\phi^{-1}(j) \subseteq [l^{(T)}, r^{(T)})\right] &= \prod_{t=1}^{T-1} \Pr\left[\phi^{-1}(j) \subseteq \left[l^{(t+1)}, r^{(t+1)}\right) \;\middle|\; \phi^{-1}(j) \subseteq \left[l^{(t)}, r^{(t)}\right)\right] \\ &\geq (1 - \mathsf{negl}(n'))^T \geq 1 - T \cdot \mathsf{negl}(n') = 1 - \mathsf{negl}(n').\end{aligned}$$

For the second term in Equation (5.2), let $\mathcal{I}[L, R-1]$ be the closure of $[l^{(T)}, r^{(T)})$. Then $R - L \leq 2 + 36(\beta + \gamma)/(\beta - \gamma) \leq 50$. Conditioned on $\phi^{-1}(j) \subseteq [l^{(T)}, r^{(T)}) \subseteq \mathcal{I}[L, R-1]$, the interval $\mathcal{I}[L, R-1]$ is (θ, γ)-good. Therefore every block in $\mathcal{I}[L, R-1]$ is γ-good since the number of γ-bad blocks is bounded by $(R-L)\theta \leq 50\theta < 1$. Due to item (2) of Theorem 5.6.7, we have

$$\Pr\left[\mathbf{b} = b_j \;\middle|\; \phi^{-1}(j) \subseteq [l^{(T)}, r^{(T)})\right] = 1.$$

Hence the theorem follows. □

6. RESOURCE-BOUNDED LOCALLY DECODABLE CODES: USING CRYPTOGRAPHIC PUZZLES FOR CONSTRUCTIONS WITHOUT RANDOM ORACLES

A portion of this chapter appears in The International Association for Cryptologic Research Cryptology ePrint Archive [13], available https://ia.cr/2021/801. A portion of this chapter is to appear in the 13th Conference on Security and Cryptography for Networks – SCN 2022 https://scn.unisa.it/scn22/.

In this chapter, we turn to one explicit use of cryptographic primitives in the construction of new alternative notions of locally decodable codes. Recall that a $(\ell, \delta, p, \text{dist})$-locally decodable code (LDC) (Definition 2.7.5) is a coding scheme $C[K, k, q_1, q_2]$ such that the decoder $\mathsf{Dec} : [k] \to \Sigma_1$ is a probabilistic oracle algorithm with the following properties. For any message $x \in \Sigma_1^k$, when the decoder is given oracle access to any $\tilde{y} \in \Sigma_2^*$ such that $\text{dist}(\tilde{y}, \mathsf{Enc}(x)) \leqslant \delta$, the decoder makes at most ℓ queries to its oracle and outputs $x_i \in \Sigma_1$ with probability at least p. Here ℓ is the *locality* (also known as *query complexity*), δ is the *error-tolerance*, p is the *success probability*, and dist is some fractional distance (e.g., fractional Hamming distance). Informally, locally decodable codes are codes which allow for super-efficient decoding of individual bits (or symbols) of the original encoded message, *without* decoding the entire message. In particular, the decoder for such codes makes a bounded number of queries to the code word (given as an oracle to the decoder), and outputs the desired symbol with good probability. Locally decodable codes have strong cryptographic motivations and influences, such as hard-core bits [128], program testing [22, 63, 189], arithmetization [21, 22, 192, 236], and private information retrieval schemes [12, 25, 92, 127, 170, 173]. For example, the proof of the Goldreich-Levin Theorem [128] immediately yields a local list decoder for the Walsh-Hadamard code [17].

We turn our focus to LDCs which are resilient against Hamming errors; i.e., dist is the fractional Hamming distance. Classical Hamming LDC constructions fall into three locality-complexity regimes: constant locality, poly-logarithmic (in K) locality, and sub-polynomial (but super logarithmic) locality. For the constant locality regime, the gap between the best constructions and lower bounds is large. The best known constructions are via matching-vector codes which have block length $K = \exp(\exp(\sqrt{\log(k) \cdot \text{loglog}(k)}))$ [108,

109, 265], while the best known lower bounds are slightly sub-quadratic with block length $K = \Omega(k^{1-2/\ell-1})/\log(k)$ for locality $\ell \geqslant 3$ [260, 261]. In the poly-logarithmic locality regime, Reed-Muller codes are Hamming LDCs with locality $\log^c(k)$ for some constant $c > 1$ and have block length $K = O(k^{1/(c-1)+o(1)})$ [264]. Finally, in the sub-polynomial, super-logarithmic locality regime, there exists Hamming LDCs with constant rate (i.e., block length $K = O(k)$) [182, 183].

While classical Hamming LDCs exhibit large gaps within and between locality regimes, some research has been directed towards some alternative models for LDCs in order to circumvent the above limitations. For example, Ben-Sasson et al. [37] introduce and define *relaxed* LDCs, or rLDCs in short. An rLDC allows the decoding algorithm to output $\perp$ on a small (constant) fraction of indices $i \in [k]$. This simple relaxation yields rLDC constructions with constant locality and block length nearly linear at $K = k^{1+\varepsilon}$ for small positive constant $\varepsilon > 0$ [37]. Recently, Blocki et al. [57] study rLDCs in the context of *computationally bounded channels* (or PPT channels) [188]. Again, as in the case of rLDCs, this further relaxation allows for constructions that achieve better rate-locality tradeoffs than classical Hamming LDCs [57]. Relaxed LDCs targeting computationally bounded channels are also heavily motivated by both modern cryptography, where adversaries are assumed to be PPT, and the real world where Lipton [188] argues that all channels can be reasonably modeled as PPT.

Turing back to explicit use of cryptographic primitives in LDCs, we examine the work of Ostrovsky, Pandey, and Sahai [214]. They introduce the notion of *private locally decodable codes*, or private LDCs. These LDCs are secure against computationally bounded channels and equip both the encoding and decoding algorithm with a shared *secret key* generated via standard cryptographic techniques (e.g., one-way functions, pseudo-random generators).[1] Key to their construction is that the key shared by the encoder and decoder *remains a secret from the channel*. The secret key assumption yields a construction of a private LDC with constant rate and slightly super-logarithmic query complexity $\omega(\log(\lambda))$, and success probability $1 - \mathsf{negl}(\lambda)$, where λ is the security parameter. Follow-up works of Hemenway and Ostrovsky [155] and Hemenway et al. [156] use public-key cryptographic assumptions to

[1]↑If the secret key is generated uniformly at random, then these codes are secure against computationally unbounded channels; i.e., the classical channel model for codes.

construct *public-key LDCs*; here, the decoding algorithm has the secret key and the encoding algorithm uses the public key when generating codewords. These LDCs also achieve better rate-locality tradeoffs than classical Hamming LDCs at the cost of public-key cryptographic assumptions.

6.1 Resource-Bounded Locally Decodable Codes

Recently, Blocki, Kulkarni, and Zhou [60] introduce and study *resource-bounded locally decodable codes*. For these LDCs, the channel is assumed to belong to some algorithm class $\mathbb{C}$ which is resource-constrained in some way; e.g., the channel is a low-depth circuit, the channel is PPT, or the channel is memory-constrained. This is a generalization of Lipton's computational channel model [188] and (arguably) captures many channels encountered in nature (i.e., the real world). Informally, a code $C[K, k, q_1, q_2]$ is a $(\ell, \delta, p, \varepsilon, \delta, \mathbb{C})$-*LDC against class* $\mathbb{C}$ if on any input $i \in [k]$, the decoder makes at most ℓ queries to its (possibly δ-corrupt) codeword oracle and outputs the i^{th} symbol with probability at least p. The $(\mathbb{C}, \varepsilon)$-security property informally states that any adversary cannot produce a δ-corrupt codeword oracle that causes the decoder to output some symbol x_i of the original encoded message with probability less than p (see Definition 6.5.3 for formal definition).

Blocki, Kulkarni, and Zhou construct a resource-bounded LDC, in the random oracle model, with constant rate, locality polylog(λ), and success probability $1 - \mathsf{negl}(\lambda)$ for security parameter λ. Their construction is provably secure for any channel class $\mathbb{C}$ admitting a *safe function*. A function f is ρ-*safe* against class $\mathbb{C}$ of algorithms if for all $\mathcal{A} \in \mathbb{C}$ we have $\Pr[\mathcal{A}(x) = f(x)] \leqslant \rho$, where the probability is taken over the random coins of $\mathcal{A}$ and $x \xleftarrow{\$} \{0,1\}^*$. They prove that, in the random oracle model, there exists a safe-function against the class $\mathbb{C}$ of parallel random access machines, and thus are able to construct provably secure resource-bounded LDCs against the class $\mathbb{C}$. However, they conjecture that such constructions exist under standard cryptographic assumptions without the random oracles.

6.2 Our Results

We resolve the open problem of Blocki, Kulkarni, and Zhou [60]. We construct resource-bounded LDCs against any class $\mathbb{C}$ for which there exists a *cryptographic puzzle* secure against the class $\mathbb{C}$. Informally, a *cryptographic puzzle* [43] consists of two algorithms $(\mathsf{Puz.Gen}, \mathsf{Puz.Sol})$, and has the properties that $\mathsf{Puz.Gen}$ should be "easy" to compute and $\mathsf{Puz.Sol}$ should be "hard" to compute, capturing the intuition that puzzles should be "easy" to generate and "difficult" to solve. A $\mathbb{C}$-*hard puzzle* informally states that any adversary $\mathcal{A} \in \mathbb{C}$ cannot solve a randomly generated puzzle. Given such puzzles, we can construct Hamming LDCs secure against the class $\mathbb{C}$.

Theorem 6.2.1 (Informal, see Theorem 6.6.2 [13])**.** *Let $\mathbb{C}$ be a class of algorithms such that there exists a $\mathbb{C}$-hard puzzle. Then there exists a construction of a locally decodable code code for Hamming errors that is secure against the class $\mathbb{C}$.*

Our LDC can be instantiated with any (concretely secure) cryptographic puzzle. In particular, the (concretely secure versions of) *time-lock puzzles* of Bitansky et al. [43] directly give us LDCs which are secure against small-depth channels. Informally, a time-lock puzzle is a cryptographic puzzle that any adversary represented by a low-depth circuit cannot solve a randomly generated puzzle. Additionally, in [13] we introduce the notion of *memory-hard puzzles*: cryptographic puzzles where any parallel adversary with "small memory" cannot solve a randomly generated puzzle.[2] We also construct such puzzles assuming the existence of succinct randomized encodings and the additional minimal assumption that a memory-hard language exists. In this report, we do not focus on memory-hard puzzles, but mention them here as an example of the types of resource-bounded LDCs we can construct using Theorem 6.6.2. See [13] for details.

6.3 Technical Overview

For sake of presentation, we formally define $(\mathbb{C}, \varepsilon)$-hard puzzles.

[2]↑This is a very large oversimplification. See [13] for details.

Definition 6.3.1 ($(\mathbb{C}, \varepsilon)$-hard Puzzle)**.** *A puzzle* $\mathsf{Puz} = (\mathsf{Puz.Gen}, \mathsf{Puz.Sol})$ *is a* $(\mathbb{C}, \varepsilon)$*-hard puzzle for algorithm class* $\mathbb{C}$ *there exists a polynomial* t' *such that for all polynomials* $t > t'$ *and every algorithm* $\mathcal{A} \in \mathbb{C}$*, there exists* λ_0 *such that for all* $\lambda > \lambda_0$ *and every* $s_0, s_1 \in \{0,1\}^\lambda$ *we have*

$$\left|\Pr\left[\mathcal{A}(Z_b, Z_{1-b}, s_0, s_1)\right] - \frac{1}{2}\right| \leqslant \varepsilon(\lambda),$$

where the probability is taken over $b \xleftarrow{\$} \{0,1\}$ *and* $Z_i \leftarrow \mathsf{Puz.Gen}(1^\lambda, t(\lambda), s_i)$ *for* $i \in \{0,1\}$*.*

Intuitively, the definition states that any adversary in the class $\mathbb{C}$ cannot distinguish between two randomly generated puzzles, even when given the respective solutions to those puzzles.

With the definition of $(\mathbb{C}, \varepsilon)$-hard puzzles in hand, recall that a resource-bounded LDC is a locally decodable code that is secure against some class $\mathbb{C}$ of adversaries, assumed to have some resource constraint. For example, $\mathbb{C}$ can be a class of adversaries that are represented by low-depth circuits, or have small (amortized) area-time complexity. In more detail, security of resource-bounded LDCs requires that any adversary in the class $\mathbb{C}$ cannot corrupt an encoding $y = \mathsf{Enc}(x)$ to some $\tilde{y}$ such that (1) The distance between y and $\tilde{y}$ is small; and (2) There exists an index i such that the decoder, when given $\tilde{y}$ as its oracle, outputs x_i with probability less than p.

We construct our resource-bounded LDC by modifying the construction of [60] to use cryptographic puzzles in place of random oracles. In particular, for algorithm class $\mathbb{C}$, if there exists a cryptographic puzzle that is unsolvable by any algorithm in $\mathbb{C}$, then we use this puzzle to construct a LDC secure against $\mathbb{C}$. Our construction, mirroring [60], relies on another relaxed LDC: a *private LDC* [214]. Private LDCs are LDCs that are additionally parameterized by a key generation algorithm that on input 1^λ for security parameter λ outputs a shared secret key sk to *both* the encoding and decoding algorithm. Crucially, this secret key is hidden from the adversarial channel.

We construct our Hamming LDC as follows. Let $(\mathsf{Gen}, \mathsf{Enc_p}, \mathsf{Dec_p})$ be a private Hamming LDC. The encoder, on input message x, samples random coins $s \in \{0,1\}^\lambda$ then generates cryptographic puzzle Z with solution s. The encoder then samples a secret key $\mathsf{sk} \leftarrow \mathsf{Gen}(1^\lambda; s)$ using the OPS key generation algorithm, where Gen uses random coins s, and encodes the

message x as $Y_1 = \mathsf{Enc}_{\mathsf{p}}(x; \mathsf{sk})$. The puzzle Z is then encoded as Y_2 via some repetition code. The encoder then outputs $Y = Y_1 \circ Y_2$. This codeword is corrupted to some $\widetilde{Y}$, which can be parsed as $\widetilde{Y} = \widetilde{Y}_1 \circ \widetilde{Y}_2$. The local decoder, on input index i and given oracle access to $\widetilde{Y}$, first recovers puzzle Z by querying $\widetilde{Y}_2$ (e.g., via random sampling with majority vote). The decoder then solves puzzle Z and recovers solution s. Given s, the local decoder is able to generate the same secret key $\mathsf{sk} \leftarrow \mathsf{Gen}(1^\lambda; s)$ and now runs the local decoder $\mathsf{Dec}_{\mathsf{p}}(i; \mathsf{sk})$. All queries made by $\mathsf{Dec}_{\mathsf{p}}(i; \mathsf{sk})$ are answered by querying $\widetilde{Y}_1$, and the decoder outputs $\mathsf{Dec}_{\mathsf{p}}(i; \mathsf{sk})$. The construction is secure against any class $\mathbb{C}$ for which there exist cryptographic puzzles that are secure against this class.

Security is established via a reduction to the cryptographic puzzle. In particular, if there exists an adversary A in the class $\mathbb{C}$ which can violate the security of our construction, then we construct another adversary in the class $\mathbb{C}$ which can break the security of the cryptographic puzzle. In particular, the reduction relies on a two-phase hybrid distinguishing argument [60]. Let Enc and Dec be the encoder and local decoder constructed above. Define $\mathsf{Enc}_0 := \mathsf{Enc}$ and define Enc_1 identically as Enc_0, except additionally Enc_1 takes as input a secret key sk_1 (whereas Enc_0 samples a secret key sk_0) that is given to the encoder $\mathsf{Enc}_{\mathsf{p}}(\cdot; \mathsf{sk}_1)$ (whereas Enc_0 gives secret key sk_0). Phase one of the argument samples $b \xleftarrow{\$} \{0, 1\}$ uniformly at random to encode a message x with sk_b, and obtains corrupted codeword $\widetilde{Y}_b \leftarrow A(x, \mathsf{Enc}_b(x; \mathsf{sk}_b))$. Let $\widetilde{Y}_b = \widetilde{Y}_{0,b} \circ \widetilde{Y}_{1,b}$ where $\widetilde{Y}_{0,b} = \mathsf{Enc}_{\mathsf{p}}(x; \mathsf{sk}_b)$. Phase two of the argument consists of constructing a distinguisher $\mathcal{D}$ which is given message x, secret key sk_b, and $\widetilde{Y}_{0,b}$; more importantly, $\mathcal{D}$ is *not* given access to the cryptographic puzzle, or the encoding of the cryptographic puzzle. The distinguisher then must output the choice bit b. Given this distinguisher, we construct an adversary $B \in \mathbb{C}$ such that B on input (Z_b, Z_{1-b}, s_0, s_1) for uniformly random bit b, where Z_i is a puzzle with solution s_i, outputs b with probability proportional to the distinguisher, breaking the cryptographic puzzle assumption.

6.4 Additional Related Work

Cryptographic puzzles are functions which require some specified amount of resources (e.g., time or space) to compute. Time-lock puzzles, introduced by Rivest, Shamir, and

Wagner [227] extending the study of timed-released cryptography of May [200], are puzzles which require large sequential time to solve: any circuit solving the puzzle has large depth. [227] proposed a candidate time-lock puzzle based on the conjectured sequential hardness of exponentiation in RSA groups, and the proposed schemes of [66, 120] are variants of this scheme. Mahmoody, Moran, and Vadhan [194] give a construction of *weak* time-lock puzzles in the random oracle model, where "weak" says that both a puzzle generator and puzzle solver require (roughly) the same amount of computation, whereas the standard definition of puzzles requires the puzzle generation algorithm to be much more efficient than the solving algorithm. Closer to our work, Bitansky et al. [43] construct time-lock puzzles using succinct randomized encodings, which can be instantiated from one-way functions, indistinguishability obfuscation, and other assumptions [122]. Recently, Malavolta and Thyagarajan [196] introduce and construct homomorphic time-lock puzzles: puzzles where one can compute functions over puzzle solutions without solving them. Continued exploration of indistinguishability obfuscation has pushed it closer and closer to being instantiated from well-founded cryptographic assumptions such as learning with errors [161].

We also mention that in [13], we construct a new notion of cryptographic puzzles known as *memory-hard puzzles*. Such puzzles are cryptographic puzzles which, intuitively, cannot be solved by algorithms with insufficient *space-time complexity*. We use these memory-hard puzzles to give the first construction of a (one-time secure) memory-hard function in the standard model, without relying on idealized assumptions such as random oracles.

6.5 Preliminaries

Our construction utilizes a number of alternative Hamming locally decodable codes (LDCs), such as the private LDCs of Ostrovsky, Pandey, and Sahai [214] and a relaxed notion called a LDC^* [60]. A LDC^* is a LDC that is required to decode the entire original message while making as few queries as possible to its provided oracle. We formally give the definitions of these codes along with the formal definition of a $\mathbb{C}$-secure LDC. We also recall the definition of a $(\mathbb{C}, \varepsilon)$-hard puzzle from Section 6.3.

$\texttt{priv-LDC-Sec-Game}(\mathcal{A}, x, \lambda, \delta, p)$:

1. The challenger generates a secret key $\mathsf{sk} \leftarrow \mathsf{Gen}(1^\lambda)$, computes the codeword $y \leftarrow \mathsf{Enc}_{\mathsf{sk}}(x, \lambda)$ for the message x and sends the codeword y to the attacker.
2. The attacker outputs a corrupted codeword $y' \leftarrow \mathcal{A}(x, y, \lambda, \delta, p, k, K)$ where $y' \in \Sigma_2^K$ should have fractional Hamming distance at most δ from y.
3. The output of the experiment is determined as follows:

$$\texttt{priv-LDC-Sec-Game}(\mathcal{A}, x, \lambda, \delta, p) = \begin{cases} 1 & \text{if } \mathsf{HAM}(y, y') \leqslant \delta K \text{ and } \exists i \in [k] \text{ s.t. } \Pr\left[\mathsf{Dec}_{\mathsf{sk}}^{y'}(i, \lambda) = x_i\right] < p \\ 0 & \text{otherwise} \end{cases}$$

If the output of the experiment is 1 (resp. 0), the attacker $\mathcal{A}$ is said to *win* (resp. *lose*) against C.

Figure 6.1. Definition of `priv-LDC-Sec-Game`, which defines the security of the a one-time private Hamming LDC against the class $\mathbb{C}$ of algorithms.

Definition 6.5.1 ([60])**.** *A coding scheme $C[K, k, q_1, q_2] = (\mathsf{Enc}, \mathsf{Dec})$ is an (ℓ, δ, p)-LDC^* if Dec, with oracle access to a word y' such that $\mathsf{HAM}(\mathsf{Enc}(x), y') \leqslant \delta$, makes at most ℓ queries to y' and outputs x with probability at least p.*

We given an equivalent definition of *one-time private LDCs* which are secure with respect to a particular class of algorithms $\mathbb{C}$.

Definition 6.5.2 (One-Time Private Key LDC)**.** *Let $C[K, k, \lambda, q_1, q_2] = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be a triple of probabilistic algorithms. We say C is a $(\ell, \delta, p, \varepsilon, \mathbb{C})$-*private Hamming locally decodable code *(private LDC) against the class of algorithms $\mathbb{C}$ if*

1. *$\mathsf{Gen}(1^\lambda)$ is the key generation algorithm that takes as input 1^λ and outputs secret key $\mathsf{sk} \in \{0,1\}^*$ for security parameter λ;*
2. *$\mathsf{Enc}\colon \Sigma_1^k \times \{0,1\}^* \to \Sigma_2^K$ is the encoding algorithm that takes as input message $x \in \Sigma_1^k$ and secret key sk and outputs a codeword $y \in \Sigma_2^K$;*
3. *$\mathsf{Dec}^{y'} : [k] \times \{0,1\}^* \to \Sigma_1$ is the decoding algorithm that takes as input index $i \in [k]$ and secret key sk, is additionally given query access to a corrupted codeword $y' \in \Sigma^{K'}$, and outputs $b \in \Sigma_1$ after making at most ℓ queries to y'; and*

$\texttt{LDC-Sec-Game}(\mathcal{A}, x, \lambda, \delta, p)$:

1. The challenger computes $Y \leftarrow \mathsf{Enc}(x, \lambda)$ encoding the message x and sends $Y \in \Sigma_2^K$ to the attacker.
2. The channel $\mathcal{A}$ outputs a corrupted codeword $Y' \leftarrow \mathcal{A}(x, Y, \lambda, \delta, p, k, K)$ where $Y' \in \Sigma_2^K$ has Hamming distance at most δK from Y.
3. The output of the experiment is determined as follows:

$$\texttt{LDC-Sec-Game}(\mathcal{A}, x, \lambda, \delta, p) = \begin{cases} 1 & \text{if } \mathsf{HAM}(Y, Y') \leqslant \delta K \text{ and } \exists i \leqslant k \text{ s.t. } \Pr\left[\mathsf{Dec}^{Y'}(i, \lambda) = x_i\right] < p \\ 0 & \text{otherwise} \end{cases}$$

If the output of the experiment is 1 (resp. 0), the channel is said to *win* (resp. *lose*).

Figure 6.2. `LDC-Sec-Game` defining the interaction between an attacker and an honest party.

4. *For all algorithms* $\mathcal{A} \in \mathbb{C}$ *and all messages* $x \in \Sigma_1^k$ *we have* $\Pr\left[\texttt{priv-LDC-Sec-Game}(\mathcal{A}, x, \lambda, \delta, p) = 1\right] \leqslant \varepsilon$*, where the probability is taken over the random coins of* $\mathcal{A}$*,* Gen*, and* `priv-LDC-Sec-Game` *defined in Figure 6.1.*

Note that setting $\mathbb{C}$ to be the class of computationally unbounded adversaries and assuming that Gen outputs long enough random strings, Definition 6.5.2 is equivalent to the definition given in Definition 5.5.1.

We give another equivalent definition of $\mathbb{C}$-secure LDCs, noting that it is equivalent to Definition 5.5.2.

Definition 6.5.3 ($\mathbb{C}$-Secure LDC)**.** *Let* $\mathbb{C}$ *be a class of algorithms. A coding scheme* $C[K, k, q_1, q_2]$ *is an* $(\ell, \delta, p, \varepsilon, \mathbb{C})$-locally decodable code *if*

1. $\mathsf{Enc}\colon \Sigma_1^k \to \Sigma_2^K$ *is the encoding algorithm that takes as input message* $x \in \{0,1\}^k$ *and outputs a codeword* $y \in \{0,1\}^K$*;*
2. $\mathsf{Dec}^{y'}\colon [k] \to \Sigma_1$ *is the decoding algorithm that takes as input index* $i \in [k]$*, is additionally given query access to a corrupted codeword* $y' \in \{0,1\}^{K'}$*, and outputs* $b \in \Sigma_1$ *after making at most* ℓ *queries to* y'*; and*

3. *For all algorithms* $\mathcal{A} \in \mathbb{C}$ *and all messages* $x \in \{0,1\}^k$ *we have*

$$\Pr\left[\texttt{LDC-Sec-Game}(\mathcal{A}, x, \lambda, \delta, p) = 1\right] \leqslant \varepsilon,$$

where the probability is taken over the random coins of $\mathcal{A}$ *and* `LDC-Sec-Game`*, defined in Figure 6.2.*

We conclude by recalling the definition of a $\mathbb{C}$-hard puzzle from Section 6.1.

Definition 6.3.1 ($(\mathbb{C}, \varepsilon)$-hard Puzzle)**.** *A puzzle* $\mathsf{Puz} = (\mathsf{Puz.Gen}, \mathsf{Puz.Sol})$ *is a* $(\mathbb{C}, \varepsilon)$*-hard puzzle for algorithm class* $\mathbb{C}$ *there exists a polynomial* t' *such that for all polynomials* $t > t'$ *and every algorithm* $\mathcal{A} \in \mathbb{C}$*, there exists* λ_0 *such that for all* $\lambda > \lambda_0$ *and every* $s_0, s_1 \in \{0,1\}^\lambda$ *we have*

$$\left|\Pr\left[\mathcal{A}(Z_b, Z_{1-b}, s_0, s_1)\right] - \frac{1}{2}\right| \leqslant \varepsilon(\lambda),$$

where the probability is taken over $b \xleftarrow{\$} \{0,1\}$ *and* $Z_i \leftarrow \mathsf{Puz.Gen}(1^\lambda, t(\lambda), s_i)$ *for* $i \in \{0,1\}$*.*

6.6 Resource-Bounded Locally Decodable Codes from Cryptographic Puzzles

We present the construction of our resource-bounded LDCs from $(\mathbb{C}, \varepsilon)$-hard puzzles.

Construction 6.6.1. *Let* $C_{\mathsf{p}}[K_{\mathsf{p}}, k_{\mathsf{p}}, \lambda, q_1, q_2] = (\mathsf{Gen}, \mathsf{Enc}_{\mathsf{p}}, \mathsf{Dec}_{\mathsf{p}})$ *be a private Hamming LDC, let* $C_*[K_*, k_*, 2, q_2] = (\mathsf{Enc}_*, \mathsf{Dec}_*)$ *be a Hamming* LDC^**, and let* $\mathsf{Puz} = (\mathsf{Puz.Gen}, \mathsf{Puz.Sol})$ *be a* $(\mathbb{C}, \varepsilon')$*-hard puzzle. Let* t' *be the polynomial guaranteed by Definition 6.3.1. Then we construct* $C[K, k, q_1, q_2] = (\mathsf{Enc}, \mathsf{Dec})$ *as follows.*

$\mathsf{Enc}(x,\lambda)[C_{\mathsf{p}}, C_*, \mathsf{Puz}]$:	$\mathsf{Dec}^{Y'_{\mathsf{p}} \circ Y'_*}(i,\lambda)[C_{\mathsf{p}}, C_*, \mathsf{Puz}]$:
1. *Sample random seed* $s \xleftarrow{\$} \{0,1\}^\lambda$.	1. *Decode* $Z \leftarrow \mathsf{Dec}_*^{Y_*}$.
2. *Choose polynomial* $t > t'$ *and compute* $Z \leftarrow \mathsf{Puz.Gen}(1^\lambda, t(\lambda), s)$, *where* $Z \in \{0,1\}^{k_*}$.	2. *Compute* $s \leftarrow \mathsf{Puz.Sol}(Z)$.
3. *Set* $Y_* \leftarrow \mathsf{Enc}_*(Z)$.	3. *Compute* $\mathsf{sk} \leftarrow \mathsf{Gen}_{\mathsf{p}}(1^\lambda; s)$.
4. *Set* $\mathsf{sk} \leftarrow \mathsf{Gen}_{\mathsf{p}}(1^\lambda; s)$.	4. *Output* $\mathsf{Dec}_{\mathsf{p}}^{Y'_{\mathsf{p}}}(i; \mathsf{sk})$.
5. *Set* $Y_{\mathsf{p}} \leftarrow \mathsf{Enc}_{\mathsf{p}}(x, \lambda; \mathsf{sk})$.	
6. *Output* $Y_{\mathsf{p}} \circ Y_*$.	

We prove that if there exists a $\mathbb{C}$-hard puzzle, then Construction 6.6.1 is a $\mathbb{C}$-secure Hamming LDC.

Theorem 6.6.2 ([13])**.** *Let* $\mathbb{C}$ *be a class of algorithms. Let* $C_{\mathsf{p}}[K_{\mathsf{p}}, k_{\mathsf{p}}, \lambda]$ *be a* $(\ell_{\mathsf{p}}, \delta_{\mathsf{p}}, p_{\mathsf{p}}, \varepsilon_{\mathsf{p}})$*-private Hamming LDC and let* $C_*[K_*, k_*]$ *be a* (ℓ_*, δ_*, p_*)*-*LDC^**. Further assume that* $\mathsf{Enc}_{\mathsf{p}}$*,* $\mathsf{Dec}_{\mathsf{p}}$*, and* Enc_* *are contained in* $\mathbb{C}$*. If there exists a* $(\mathbb{C}, \varepsilon')$*-hard puzzle, then Construction 6.6.1 is a* $(\ell, \delta, p, \varepsilon, \mathbb{C})$*-locally decodable code* $C[K, k] = (\mathsf{Enc}, \mathsf{Dec})$ *with* $k = k_{\mathsf{p}}$*,* $K = K_{\mathsf{p}} + K_*$*,* $\ell = \ell_{\mathsf{p}} + \ell_*$*,* $\delta = (1/K) \cdot \min\{\delta_* \cdot K_*, \delta_{\mathsf{p}} \cdot K_{\mathsf{p}}\}$*,* $p \geqslant 1 - k_{\mathsf{p}}(2 - p_{\mathsf{p}} - p_*)$*, and* $\varepsilon = (\varepsilon_{\mathsf{p}} \cdot p + 2\varepsilon')/(1-p)$*.*

Ostrovsky, Pandey, and Sahai [214] give a construction of a one-time private Hamming LDC with linear block length $K_{\mathsf{p}} = \theta(k_{\mathsf{p}})$, constant error-tolerance $\delta_{\mathsf{p}} = \Theta(1)$, locality $\log^{1+\epsilon}(\lambda)$ for any constant $\epsilon > 0$, success probability $p_{\mathsf{p}} = 1 - \vartheta_1(\lambda)$, and security $\varepsilon = \vartheta_2(\lambda)$, where λ is the security parameter and ϑ_1, ϑ_2 are fixed negligible functions. Blocki, Kulkarni, and Zhou [60] give a construction of a LDC^* via a repetition code using Justesen codes [162] with the following parameters: the LDC^* has constant rate, constant error tolerance $\delta_* = \Theta(1)$, constant success probability $p_* = \Theta(1)$, and locality $\ell_* = \Theta(k_*)$. By suitably choosing parameters of the above constructions, we have the following corollary.

Corollary 6.6.3. *Let* $\mathbb{C}$ *be a class of algorithms, let* $C_{\mathsf{p}}[K_{\mathsf{p}}, k_{\mathsf{p}}, \lambda]$ *be the one-time private Hamming LDC of [214], and let* $C_*[K_*, k_*]$ *be the* LDC^* *of [60]. Assume that* $\mathsf{Enc}_{\mathsf{p}}$*,* $\mathsf{Dec}_{\mathsf{p}}$*, and* Enc_* *are contained in* $\mathbb{C}$*. Then if there exists a* $(\mathbb{C}, \varepsilon')$*-hard puzzle, then for security parameter*

λ and every $k = \text{poly}(\lambda)$, Construction 6.6.1 is a $(\ell, \delta, p, \varepsilon, \mathbb{C})$-$\mathsf{LDC}$ *$C[K, k] = (\mathsf{Enc}, \mathsf{Dec})$ with constant rate $K = \Theta(k)$, $\ell = \Theta(\log^{1+\epsilon}(\lambda))$, success probability $p = \Theta(1)$, error-tolerance $\delta = \Theta(1)$, and security $\varepsilon = \mathsf{negl}(\lambda)$, where negl is an unspecified negligible function.*

6.6.1 Efficiency

The efficiency of the scheme is directly given by the efficiency of C_{p}, C_*, and puz. In particular, if all of the algorithms defined by $C_{\mathsf{p}}, C_*, \mathsf{Puz}$ are polynomial time, then Enc and Dec both run in polynomial time. We also remark that our LDC encoder Enc can be resource bounded: the encoder Enc only needs to be able to compute $\mathsf{Puz.Gen}$, $\mathsf{Enc}_{\mathsf{p}}$, Enc_*, and $\mathsf{Gen}_{\mathsf{p}}$. Crucially, the encoder does not need to compute $\mathsf{Puz.Sol}$. This is in contrast with Blocki, Kulkarni, and Zhou [60], where their encoding function could not be resource-bounded.

6.6.2 Security

We begin with a high-level overview of the security proof. In the same vein as Blocki, Kulkarni, and Zhou [60], we employ the use of a two-phased hybrid distinguisher. This distinguishing argument proceeds as follows. In phase one, we consider two encoders Enc_0 and Enc_1. The encoder Enc_0 is exactly the encoder for our LDC in Construction 6.6.1. The encoder Enc_1 is the hybrid encoder and differs as follows: (1) Enc_1 is given additionally as input a secret key sk to be used with the private LDC C_{p}, rather than generating this key; and (2) the part of the codeword Y_* is constructed by sampling some s' independently and uncorrelated with sk, and then encoding $\mathsf{Enc}_*(\mathsf{Puz.Gen}(s'))$. Phase one takes as input a message x, flips a bit $b \xleftarrow{\$} \{0, 1\}$, obtains codeword $Y_b \leftarrow \mathsf{Enc}_b(x)$, and then obtains corrupted codeword $Y'_b \leftarrow \mathcal{A}(Y_b)$, where $\mathcal{A} \in \mathbb{C}$.

In the second phase, an algorithm $\mathcal{D}$ is given the original message x, the secret key sk_b used in Enc_b, and the corrupted codeword $Y'_{\mathsf{p},b}$. We note that $Y'_{\mathsf{p},b}$ is a substring of Y'_b that corresponds to the corruption of the codeword $Y_{\mathsf{p},b} \leftarrow \mathsf{Enc}_{\mathsf{p}}(x, \mathsf{sk}_b)$. Further, the algorithm $\mathcal{D}$ is *not given access* to the puzzle $\mathsf{Puz.Sol}$. The algorithm $\mathcal{D}$ is asked to output bit b', and wins this game if $b' = b$.

Now if the adversary $\mathcal{A}$ is able to break LDC-Sec-Game with probability at least ε, we want to construct an algorithm $\mathcal{B} \in \mathbb{C}$ that uses this distinguishing argument to break the security of Puz. This is done as follows. Suppose $\mathcal{B}$ is given as input (Z_b, Z_{1-b}, s_0, s_1) for some $b \xleftarrow{\$} \{0,1\}$ that is unknown to $\mathcal{B}$ and where s_0, s_1 are uniformly random. Then $\mathcal{B}$ uses s_0 to generate sk, encodes $Y_* \leftarrow \mathsf{Enc}_*(Z_b)$, and encodes $Y_{\mathsf{p}} \leftarrow \mathsf{Enc}_{\mathsf{p}}(x, \mathsf{sk})$ for some fixed message x. We observe that if $b = 0$, then s_0 is the solution to $Z_0 = Z_b$, and thus Y_* is correlated with the secret key sk. Further, if $b = 1$, then s_0 is uncorrelated with $Z_b = Z_1$. Corrupted codeword $Y' \leftarrow \mathcal{A}(Y_{\mathsf{p}} \circ Y_*)$ is then obtained. Next, given x, secret key sk, and substring Y'_{p}, the algorithm simulates $\mathsf{Dec}_{\mathsf{p}}$ using sk and attempts to decode x_i for some arbitrary $i \in [|x|]$, obtaining x'_i. If $x'_i \neq x_i$, then $\mathcal{B}$ outputs $b' = 0$; otherwise it outputs $b' = 1$.

Now $\mathcal{B}$ is able to break the security of Puz as follows. If $b = 0$, then sk is correlated with Y_*. This implies that $\mathcal{A}$ is able to win LDC-Sec-Game with probability at least ε by assumption; in particular, it forces $\mathsf{Dec}_{\mathsf{p}}$ to output an incorrect bit for some index i with probability at least $(1-p)$. In this case, $b' = 0$ with probability at least $\varepsilon \cdot (1-p)$. If $b = 1$, then sk is completely uncorrelated with Y_*, so information theoretically $\mathcal{A}$ cannot win LDC-Sec-Game except with probability at most ε_{p}. This implies that with probability at most $\varepsilon_{\mathsf{p}} \cdot p$ the decoder fails to output correctly on some index i, which implies that with probability at least $1 - \varepsilon_{\mathsf{p}} \cdot p$ the decoder outputs correctly on every bit. In this case, $b' = 1$ with probability at least $1 - \varepsilon_{\mathsf{p}} \cdot p$. This allows $\mathcal{B} \in \mathbb{C}$ to distinguish (Z_b, Z_{1-b}, s_0, s_1) with noticeable advantage $\Omega(\varepsilon \cdot (1-p) - \varepsilon_{\mathsf{p}} \cdot p)$ thus breaking the security of the puzzle.

Proof of Theorem 6.6.2. We first remark that definitions of k, K, ℓ, δ, and p follow directly by construction. We now turn to arguing the security of our scheme under the game LDC-Sec-Game, defined in Figure 6.2. To prove security, we assume that if there exists an adversary $\mathcal{A} \in \mathbb{C}$ that, given the puzzle Puz, can win LDC-Sec-Game with probability at least ε, then we can construct an adversary $\mathcal{B} \in \mathbb{C}$ which breaks the $(\mathbb{C}, \varepsilon')$-hard puzzle.

To prove this, we employ a two-phase hybrid distinguishing argument. In the two-phase distinguishing argument, the first phase defines two encoders: Enc_0 and Enc_1. The encoder Enc_0 is exactly identical to the encoding function of Construction 6.6.1, which we denote as Enc. The encoder Enc_1 is our hybrid encoder, and is defined as follows.

$\mathsf{Enc}_1(x, \lambda, \mathsf{sk})$:

1. Sample $s' \xleftarrow{\$} \{0,1\}^{k_\mathsf{p}}$.
2. Choose polynomial $t > t'$ and compute $Z' \leftarrow \mathsf{Puz.Gen}(1^\lambda, t(\lambda), s')$.
3. Set $Y_* \leftarrow \mathsf{Enc}_*(Z')$.
4. Set $Y_\mathsf{p} \leftarrow \mathsf{Enc}_\mathsf{p}(x, \lambda; \mathsf{sk})$.
5. Output $Y_\mathsf{p} \circ Y_*$.

Phase one of the argument then consists of randomly selecting Enc_b for $b \xleftarrow{\$} \{0,1\}$, encoding a message $Y_b \leftarrow \mathsf{Enc}_b(x, \lambda, \mathsf{sk}_b)$, and obtaining $Y_b' \leftarrow \mathcal{A}(x, Y_b, \lambda, \delta, p)$.

Phase two of the argument consists of constructing a distinguisher $\mathcal{D}$ which is given the original message x, sk_b, and the codeword $Y'_{\mathsf{p},b}$ which is the corrupted substring of Y_b' that corresponds to corrupting the string $Y_{\mathsf{p},b}$. Further, the distinguisher $\mathcal{D}$ is *not* given access to the puzzle Puz. The distinguisher is then supposed to output bit b.

We formally give our two-phase distinguisher which breaks the $(\mathbb{C}, \varepsilon')$-hard puzzle if there exists a channel $\mathcal{A} \in \mathbb{C}$ which wins `LDC-Sec-Game` with probability at least ε. Suppose such an adversary $\mathcal{A}$ exists. For puzzle solutions s_0, s_1 (viewed as independent random strings), we want to construct an adversary $\mathcal{B} \in \mathbb{C}$ which distinguishes (Z_b, Z_{1-b}, s_0, s_1) with probability at least ε' for $b \xleftarrow{\$} \{0,1\}$. Fix a message x and security parameter λ. Our adversary $\mathcal{B}$ is constructed as follows: suppose $\mathcal{B}$ is given as input (Z_b, Z_{1-b}, s_0, s_1) for some $b \xleftarrow{\$} \{0,1\}$ unknown to $\mathcal{B}$.

1. Fix message x.
2. Encode the message x as follows:
   (a) Obtain $\mathsf{sk} \leftarrow \mathsf{Gen}_\mathsf{p}(1^\lambda; s_0)$.
   (b) Set $Y_* \leftarrow \mathsf{Enc}_*(Z_b)$.
   (c) Set $Y_\mathsf{p} \leftarrow \mathsf{Enc}_\mathsf{p}(x, \lambda; \mathsf{sk})$.
   (d) Set $Y = Y_\mathsf{p} \circ Y_*$.
3. Obtain $Y' \leftarrow \mathcal{A}(x, Y, \lambda, \delta, p, k, K)$.

4. Set Y'_{p} to be the substring of Y' that corresponds to the corruption of Y_{p} above.

5. Simulate $x'_i \leftarrow \mathsf{Dec}_{\mathsf{p}}^{Y'_{\mathsf{p}}}(i; \mathsf{sk})$ for some $i \in [|x|]$.

6. If $x_i \neq x'_i$ output $b' = 0$. Else output $b' = 1$.

Notice that by assumption, $\mathcal{B} \in \mathbb{C}$ since $\mathcal{A}, \mathsf{Enc}_{\mathsf{p}}, \mathsf{Dec}_{\mathsf{p}}, \mathsf{Enc}_* \in \mathbb{C}$. Now we argue that our adversary distinguishes (Z_b, Z_{1-b}, s_0, s_1) with noticeable probability. Observe that sk is always generated as $\mathsf{Gen}_{\mathsf{p}}(1^\lambda, s_0)$. Notice that for $b = 1$ the puzzle Z_1 is encoded as Y_*, and the secret key sk is unrelated to the solution s_1 of puzzle Z_1. In this case, the adversary $\mathcal{A}$ wins the `LDC-Sec-Game` with probability at most ε_{p}; this holds information theoretically since sk and Y_* are completely unrelated and uncorrelated. With probability at most ε_{p}, $\mathcal{A}$ introduces an error pattern such that $\mathsf{HAM}(Y, Y') \leqslant \delta$ and there exists $i \leqslant k$ such that the decoder outputs x_i with probability less than p. For the case $b = 0$, puzzle Z_0 is encoded as Y_* and has solution s_0, which is used to generate sk. Thus in this case, the probability that the decoder outputs an incorrect x_i for some $i \leqslant k$ with at most probability p is at least ε since we assume $\mathcal{A}$ wins `LDC-Sec-Game` with probability at least ε.

We analyze the probability $\mathcal{B}$ outputs bit b'. First consider the case where $b = 0$. Then the probability that $b' = 0$ is at least $\varepsilon \cdot (1 - p)$ by the argument above. Now for $b = 1$, the probability that $b' = 0$ is at most $\varepsilon_{\mathsf{p}} \cdot p$, which implies that $b' = 1$ is at least $1 - \varepsilon_{\mathsf{p}} \cdot p$. Therefore

$$\Pr_{b \xleftarrow{\$} \{0,1\}} [\mathcal{B}(Z_b, Z_{1-b}, s_0, s_1) = b] \geqslant \frac{1}{2}(\varepsilon \cdot (1 - p) + 1 - \varepsilon_{\mathsf{p}} \cdot p)$$

which implies that

$$\Pr_{b \xleftarrow{\$} \{0,1\}} [\mathcal{B}(Z_b, Z_{1-b}, s_0, s_1) = b] - \frac{1}{2} \geqslant \frac{\varepsilon \cdot (1 - p) - \varepsilon_{\mathsf{p}} \cdot p}{2} = \varepsilon'.$$

Thus $\mathcal{B}$ breaks Puz with probability at least ε', which contradicts the hardness of Puz. $\square$

7. COMPUTATIONALLY RELAXED LOCALLY DECODABLE CODES FOR EDIT ERRORS: USING DIGITAL SIGNATURES FOR DIRECT CONSTRUCTIONS

In the final chapter of this dissertation, we examine another alternative notion of locally decodable code, namely that of *relaxed locally decodable codes*. A relaxed LDC, or rLDC in short, mirrors the definition of a LDC (Definition 2.7.5), with the following change: the decoding algorithm is additionally allowed to output a symbol $\perp \notin \Sigma$ indicating that the decoder "does not know" the correct symbol. Additionally, it is required that the decoder output *either* the correct symbol or $\perp$ with probability at least p, the decoder *always* output the correct symbol if its oracle is a correct codeword, and in some cases it is required that given any corrupt codeword a large fraction of the original message can still be recovered correctly (i.e., $\perp$ is not output too often).

Relaxed LDCs were originally introduced by Ben-Sasson et al. [37] and have proved to be a powerful variation of LDCs with constructions achieving *constant* locality, constant error-tolerance, and block length $K = k^{1+\varepsilon}$ for any small constant $\varepsilon > 0$ [37]. Further relaxing rLDCs, Blocki et al. [57] consider *computationally relaxed LDCs*, or crLDCs. These rLDCs are only resilient to errors introduced by adversarial channels that are computationally bounded (i.e., PPT). Similar to resource-bounded LDCs, this relaxation is inspired by the work of Lipton [188] which considers Hamming codes in the context of PPT adversarial channels instead of the classical unbounded adversarial channel. Assuming the existence of collision-resistant hash functions, Blocki et al. [57] construct crLDCs with constant error-tolerance, constant rate, and locality that is poly-logarithmic in the block length.

7.1 Our Results

In this work, we revisit the notion of computationally relaxed locally decodable codes, or crLDCs, with respect to both Hamming and insertion-deletion (InsDel) errors. To begin, a *relaxed locally decodable code* (rLDC) is defined analogously to a standard LDC (as given in Section 2.7.1), but the decoder is additionally allowed to output the symbol $\perp \notin \Sigma$ which

signifies that the decoder does not know the correct answer. It is also required that the decoder does not output $\perp$ "too often". *Computationally relaxed locally decodable codes* are rLDCs with the additional property that the adversarial channel introducing errors into codewords is assumed to be some probabilistic polynomial time (PPT) algorithm, rather than a computationally unbounded algorithm (as in the definitions of rLDCs and LDCs). We formally define crLDCs in the definition below.

Definition 7.1.1 (Computationally Relaxed Locally Decodable Codes)**.** *Let $\mathcal{C}$ be a family of coding schemes $\{C_\lambda[K, k, q_1, q_2]\}_{\lambda\in\mathbb{N}}$ for $k := k(\lambda)$ with (randomized) encoding algorithms $\{\mathsf{Enc}_\lambda \colon \Sigma_1^k \to \Sigma_2^K\}_{\lambda\in\mathbb{N}}$ where $|\Sigma_i| = q_i$. We say $\mathcal{C}$ is a $(\ell, \rho, p, \delta, \mathrm{dist})$-*computationally relaxed locally decodable code *(*crLDC*) if there exists a family of randomized oracle decoding algorithms $\{\mathsf{Dec}_\lambda \colon \{1, \ldots, k\} \to \Sigma_1\}_{\lambda\in\mathbb{N}}$ satisfying the following properties.*

1. *For all $\lambda \in \mathbb{N}$ and any word $\widetilde{y} \in \Sigma_2^*$, the algorithm $\mathsf{Dec}_\lambda^{\widetilde{y}}(i)$ makes at most ℓ queries to $\widetilde{y}$ for any $i \in [k]$;*

2. *For all $\lambda \in \mathbb{N}$ and any message $x \in \Sigma_1^k$, we have that $\Pr\left[\mathsf{Dec}_\lambda^{\mathsf{Enc}_\lambda(x)}(i) = x_i\right] = 1$ for all $i \in [k]$;*

3. *Define binary predicate $\mathsf{Fool}(y', \rho, p, x, y, \lambda) = 1$ if and only if (1) $\mathrm{dist}(y, y') \leqslant \rho$; and (2) $\exists i \in [k]$ such that $\Pr\left[\mathsf{Dec}_\lambda^{y'}(i) \in \{x_i, \perp\}\right] < p$, where the probability is taken over the random coins of Dec_λ; otherwise $\mathsf{Fool}(y', \rho, p, x, y) = 0$. We require that for all probabilistic polynomial time (PPT) adversaries $\mathcal{A}$ there exists a negligible function $\varepsilon_{\mathsf{F}}(\cdot)$ such that for all $\lambda \in \mathbb{N}$ and all $x \in \Sigma_1^k$, we have*

$$\Pr\left[\mathsf{Fool}(\mathcal{A}(y), \rho, p, x, y, \lambda) = 1\right] \leqslant \varepsilon_{\mathsf{F}}(\lambda),$$

   where the probability is taken over the random coins of $\mathcal{A}$ and $y = \mathsf{Enc}_\lambda(x)$.

4. *Define binary predicate $\mathsf{Limit}(y', \rho, \delta, x, y, \lambda) = 1$ if and only if the following hold: (1) $\mathrm{dist}(y, y') \leqslant \rho$; and (2) for set $\mathsf{Good}(y') := \{i \in [k] \colon \Pr\left[\mathsf{Dec}_\lambda^{y'}(i) = x_i\right] > 2/3\}$, we have $|\mathsf{Good}(y')| < \delta \cdot k$, where the probability is taken over the random coins of Dec_λ;*

otherwise $\mathsf{Limit}(y', \rho, \delta, x, y, \lambda) = 0$. *We require that for all PPT adversaries* $\mathcal{A}$ *there exists a negligible function* $\varepsilon_{\mathsf{L}}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$ *and all* $x \in \Sigma_1^k$, *we have*

$$\Pr\left[\mathsf{Limit}(\mathcal{A}(y), \rho, \delta, x, y, \lambda) = 1\right] \leqslant \varepsilon_{\mathsf{L}}(\lambda),$$

where the probability is taken over the random coins of $\mathcal{A}$ *and* $y = \mathsf{Enc}_\lambda(x)$.

If dist *is the normalized Hamming distance* HAM, *we say the code is a* Hamming crLDC*; if* dist *is the normalized edit distance* ED, *we say the code is a* InsDel crLDC. *Here,* ℓ *is the* locality *of* $\mathcal{C}$, ρ *is the* error-tolerance *of* $\mathcal{C}$, *and* k/K *is the* rate *of* $\mathcal{C}$. *If* $q_2 = 2$, *we say that* $\mathcal{C}$ *is a family of* binary crLDCs. *For convenience, if* $q_1 = q_2$, *we simply write* $C_\lambda[K, k, q_1]$.

Note that if we require Definition 7.1.1 to hold with respect to $p = 2/3$, $\varepsilon_{\mathsf{F}}(\lambda) = \varepsilon_{\mathsf{L}}(\lambda) = 0$ for all λ, for all computationally unbounded adversaries, and for dist $= \mathsf{HAM}$, we recover the original definition of a relaxed locally decodable code [37]. Our definition captures the notion of asymptotic security when interacting with arbitrary PPT adversarial channels; this differs from the standard definition of (relaxed) locally decodable codes, as we are concerned with worst-case errors with respect to these restricted channels. Definition 7.1.1 closely follows the definition of Blocki et al. [57] with a few modifications. First, the constructions of [57] utilize a public random seed for a collision-resistant hash function, so the crLDC definition of [57] is quantified over the randomness of the seed generation algorithm Gen. Since our constructions do not require a public random seed, we omit Gen from our definition and instead define crLDCs over a family of codes $\mathcal{C} = \{C_\lambda\}_{\lambda \in \mathbb{N}}$. We note that the definition of crLDC of [57] requires the public random seed to be generated in an honest (i.e., trusted) way, and our definitions and constructions circumvent this requirement. Second, we slightly strengthen the security definition by tweaking the predicate Fool: in our definition, the adversary wins if there *exists* an index i such that the probability the decoder outputs correctly on input i is less than p. In contrast, the security definition of [57] requires the adversary to output corrupt codeword y' and a target index i such that the probability the decoder outputs correctly on index i is less than p.

Our first contribution is the construction of a family of binary Hamming crLDCs $\mathcal{C}_{\mathsf{HAM}} = \{C_\lambda[K, k, 2]\}$ satisfying Definition 7.1.1. Our construction is conceptually simpler than the Hamming crLDC of Blocki et al. [57], does not require a trusted setup, and achieves (asymptotically) the same rate, error-tolerance, and locality. Our construction borrows from the idea of concatenation codes [118]. Briefly, code concatenation techniques utilize an outer code $C_{out} = (\mathsf{Enc}_{out}, \mathsf{Dec}_{out})$ and an inner code $C_{in} = (\mathsf{Enc}_{in}, \mathsf{Dec}_{in})$ and encode a message x via the following process: (1) encode x as $y = \mathsf{Enc}_{out}(x)$; (2) partition y into some number d of blocks $y^{(1)} \circ \ldots \circ y^{(d)}$; (3) compute $Y^{(i)} = \mathsf{Enc}_{in}(y^{(i)})$ for all $i \in \{1, \ldots, d\}$; and (4) output codeword $Y = Y^{(1)} \circ \ldots \circ Y^{(d)}$. In place of an outer code C_{out}, we utilize a suitable *digital signature scheme* in conjunction with a classical Hamming code as our inner code C_{in} to obtain our final construction. We discuss the details of our construction in Section 7.2.1. Briefly, a digital signature scheme with signatures of length $r(\cdot)$ is a tuple of PPT algorithms $\Pi = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ that satisfy the following properties. (1) The algorithm Gen takes as input security parameter $\lambda \in \mathbb{N}$ (in unary) and outputs a pair of keys $(\mathsf{pk}, \mathsf{sk})$, where pk is the *public key* or *verification key* and sk is the *private key* or *signing key*; (2) The algorithm Sign takes as input a message m of arbitrary length and the signing key sk and outputs a signature σ of message m. (3) The algorithm Verify is a deterministic algorithm that takes as input a message m, some signature σ, and a verification key pk decides if σ is a valid signature of message m. A signature scheme Π is said to be *secure* if for all PPT adversaries $\mathcal{A}$, for $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$, if $\mathcal{A}$ is given pk as input and given oracle access to the function $\mathsf{Sign}_{\mathsf{sk}}(\cdot)$, then except with negligible probability in the security parameter $\mathcal{A}$ cannot output a pair $(\widetilde{m}, \widetilde{\sigma})$ such that $\mathsf{Verify}_{\mathsf{pk}}(\widetilde{m}, \widetilde{\sigma})$ accepts and $\mathcal{A}$ never queried its oracle at input $\widetilde{m}$.

Given a digital signature scheme and any binary Hamming code, we obtain our first main result.

Theorem 7.1.1. *Suppose that $\Pi = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ is a digital signature scheme with signatures of length $r := r(\lambda)$. Let $C_{in} = (\mathsf{Enc}_{in}, \mathsf{Dec}_{in})$ be a Hamming code with rate β_{in} and error-tolerance ρ_{in}. Then for every positive polynomial $k(\cdot)$ and positive constant $c < 1/2$, there exists code family $\mathcal{C}_{\mathsf{HAM}} := \{C_{\mathsf{H},\lambda}[n, k(\lambda), 2] = (\mathsf{Enc}_{\mathsf{H},\lambda}, \mathsf{Dec}_{\mathsf{H},\lambda})\}_{\lambda \in \mathbb{N}}$ such that for all $\varepsilon > 0$ the family $\mathcal{C}_{\mathsf{HAM}}$ is a (ℓ, ρ, p, δ)-Hamming crLDC with block length $n = O((1/\beta_{in}) \max\{k(1 +$*

$\log(k)/r), r\})$, *error-tolerance* $\rho = c \cdot \rho_{in}$, *locality* $\ell = O((\log^{1+\varepsilon}(\lambda)/\beta_{in}) \cdot (r + \log(k)))$, *success probability* $p = 1 - \mathsf{negl}(\lambda) > 2/3$, *and* $\delta = 1/2$, *where* $k := k(\lambda)$ *and* $\mathsf{negl}(\cdot)$ *is a negligible function.*

Our code family $\mathcal{C}_{\mathsf{HAM}}$ is constant rate whenever $\beta_{in} = \Theta(1)$ and $r(\lambda) = \Omega(\log(k(\lambda)))$ and $r(\lambda) \leqslant k(\lambda)$. Our construction allows for $r(\lambda) > k(\lambda)$, but this results in locality $\ell \geqslant n$, which is counter to the goal of LDCs; moreover, in this case, it is simply more efficient to use a standard optimal Hamming code.

We can instantiate Theorem 7.1.1 using a constant rate, constant error-tolerance Hamming code C_{in} (e.g., [162]) and an appropriate digital signature scheme to achieve a constant rate construction with poly-logarithmic locality. While there are many signature scheme to choose from, our construction shines when $r(\lambda) = \text{polylog}(\lambda)$. Under standard idealized models, there exist secure digital signature schemes with signature lengths as small as $\Theta(\log^{1+\epsilon}(\lambda))$ for small constant $\epsilon > 0$ [61, 229] under the assumption that these schemes satisfy the following notion of concrete security: for security parameter λ, any adversary running in time $2^{\lambda/2}$ can violate the security (i.e., unforgability) of the scheme with probability at most $2^{-\lambda/2}$ for signatures of length $r(\lambda) = \lambda$. Plugging in $\lambda' = \Theta(\log^{1+\epsilon}(\lambda))$, the above schemes are secure against super-polynomial time adversaries with negligible security in λ, which implies they are asymptotically secure against PPT adversaries. Using such schemes in conjunction with $\varepsilon = \epsilon$ and a constant rate, constant error-tolerance code C_{in}, we obtain the following corollary.

Corollary 7.1.2. *Suppose that* $\Pi = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ *is a digital signature scheme with signatures of length* $r(\lambda) = \Theta(\log^{1+\epsilon}(\lambda))$ *for small constant* $\epsilon > 0$. *Then for all positive polynomials* $k(\cdot)$, *there exists a code family* $\mathcal{C}_{\mathsf{HAM}} = \{C_{\mathsf{H},\lambda}[n, k(\lambda), 2]\}_{\lambda \in \mathbb{N}}$ *that is a* (ℓ, ρ, p, δ)*-Hamming* crLDC *with parameters* $n = O(k)$, $\rho = \Theta(1)$, $\ell = O(\log^{2(1+\epsilon)}(\lambda))$, $p = 1 - \mathsf{negl}(\lambda) > 2/3$, *and* $\delta = 1/2$, *where* $k := k(\lambda)$ *and* $\mathsf{negl}(\cdot)$ *is a negligible function.*

The parameters of our construction in Corollary 7.1.2 are comparable to the construction of Blocki et al. [57]. Blocki et al. construct a Hamming crLDC using local expander graphs and assuming the existence of a collision-resistant hash function family. This Hamming

crLDC achieves constant rate k/n, constant error-rate ρ, locality $\ell = \text{polylog}(n)$, constant $\delta = \Theta(1)$, and $p = 1 - \mathsf{negl}(\lambda)$.

7.1.1 Extension to Insertion-Deletion Errors

Our second contribution is extending the construction of Theorem 7.1.1 to handle insertion-deletion errors. Prior constructions of insertion-deletion LDCs, or InsDel LDCs, utilized a so-called "Hamming-to-InsDel" compiler [48, 215]. This compiler also borrows from the notion of concatenation codes, using a suitable Hamming LDC as the outer code C_{out} and a suitable InsDel code as the inner code (i.e., a non-local code). This new InsDel LDC has asymptotically the same rate and error-tolerance as the underlying Hamming LDC at the cost of a poly-logarithmic blow-up in the locality. Key to this compiler is the usage of a *noisy binary search* algorithm, which intuitively allows one to search an almost sorted list of integers and find most entries with high probability.[1]

We use the noisy binary search tools of Block et al. [48] in their "Hamming-to-InsDel" compiler to extend the construction of Theorem 7.1.1 to the insertion-deletion setting. We additionally modify the family $\mathcal{C}_{\mathsf{HAM}}$ to make use of a suitable inner code C_{in} that is resilient to insertion-deletion errors. For our purposes, we let C_{in} be the well-known Schulman-Zuckerman insertion-deletion code [230]. This code has both constant rate and constant error-tolerance, and additionally has properties that are required by the noisy binary search algorithm of [48].

Given this noisy binary search algorithm, the Schulman-Zuckerman insertion-deletion code, and a digital signature scheme, we obtain our second main result.

Theorem 7.1.3. *Suppose that* $\Pi = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ *is a digital signature scheme with signatures of length* $r := r(\lambda)$. *For every positive polynomial* $k(\cdot)$ *and positive constant* $\rho^* < 1/3$, *there exists a code family* $\mathcal{C}_{\mathsf{Ins}} := \{C_\lambda[n, k(\lambda), 2] = (\mathsf{Enc}_{\mathsf{I},\lambda}, \mathsf{Dec}_{\mathsf{I},\lambda})\}_{\lambda\in\mathbb{N}}$ *such that for all* $\varepsilon > 0$, *the family* $\mathcal{C}_{\mathsf{Ins}}$ *is a* (ℓ, ρ, p, δ)*-InsDel* crLDC *with block length* $n = O(\max\{k(1 + \log(k)/r), r\})$, *error-tolerance* $\rho = \Theta(1)$, *locality* $\ell = O(\log^{3+\varepsilon}(\lambda) \cdot (r + \log(k)))$, *success probability* $p = 1 - \rho^* - \mathsf{negl}(\lambda) > 2/3$, *and* $\delta = 1 - \Theta(\rho)$, *where* $k := k(\lambda)$ *and* $\mathsf{negl}(\cdot)$ *is a negligible function.*

[1] ↑Looking ahead, noisy binary search is used to find blocks of codewords that are "not too corrupt"; see Sections 7.2 and 7.5 for more discussion.

Like with Theorem 7.1.1, our family $\mathcal{C}_{\mathsf{Ins}}$ is constant rate whenever $r(\lambda) = \Omega(\log(k(\lambda)))$ and $r(\lambda) \leqslant k(\lambda)$, and additionally has the same downside whenever $r(\lambda) > k(\lambda)$, in which case it is more efficient to directly encode with an (asymptotically) optimal insertion-deletion code. Moreover, under the same set of assumptions on the underlying digital signature scheme as with our Hamming crLDC [61, 229], given any sufficiently large polynomial and setting $\varepsilon = \epsilon$ for small constant $\epsilon > 0$, we obtain the following corollary.

Corollary 7.1.4. *Suppose that* $\Pi = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ *is a digital signature scheme with signatures of length* $r(\lambda) = \Theta(\log^{1+\epsilon}(\lambda))$ *for small constant* $\epsilon > 0$*. Then for every positive polynomial* $k(\cdot)$ *there exists a code family* $\mathcal{C}_{\mathsf{Ins}} := \{C_{\mathsf{I},\lambda}[n, k(\lambda), 2]\}_{\lambda \in \mathbb{N}} = \{(\mathsf{Enc}_\lambda, \mathsf{Dec}_\lambda)\}$ *that is a* (ℓ, ρ, p, δ)*-computationally relaxed locally decodable code with parameters* $n = O(k)$*,* $\ell = O(\log^{4+2\epsilon}(\lambda))$*,* $\rho = \Theta(1)$*,* $p = 1 - \mathsf{negl}(\lambda)$*, and* $\delta = 1 - \Theta(\rho)$*, where* $k := k(\lambda)$ *and* $\mathsf{negl}(\cdot)$ *is a negligible function.*

To the best of our knowledge, our InsDel crLDCs are the first of their kind. Our constructions compare favorably to the prior InsDel LDCs (i.e., non-relaxed and computationally unbounded errors) of Block et al. [48]. In the poly-logarithmic locality regime, we achieve constant rate and constant error-tolerance, while Block et al. achieve slightly sub-linear rate (i.e., $n/k = o(k)$). Moreover, the construction of Block et al. has locality $\log(k)^{O(\log\log(k))}$ with constant rate and constant error-tolerance.

7.2 Technical Overview

The main technical ingredients for both our Hamming crLDC and InsDel crLDC constructions is the use of a r-length digital signature scheme $\Pi = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ along with a suitable inner code C_{in}. The encoding algorithms for both of our codes are nearly identical, with the main difference being the choice of the inner code C_{in} (i.e., Hamming vs. InsDel). The decoding algorithms are also similar; in particular, as we will see, the InsDel decoder is a modification of the Hamming decoder to handle insertion-deletion error patterns. We begin by discussing our Hamming construction followed by our InsDel construction.

7.2.1 Hamming crLDC Construction

In this section, let C_{in} be an appropriate Hamming code (i.e., non-local), and let $\Pi = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ be an r-length digital signature scheme. Our encoding algorithm borrows from the notion of concatenation codes [118]. Concatenation codes consist of an outer code and an inner code, and (roughly) operate as follows: first, encode a message with the outer encoder; second, partition the outer code word in some way; third, encode each partition with the inner encoder; finally, output the concatenation each inner codeword. In place of an outer code, we utilize the digital signature scheme Π, and we use C_{in} as our inner code. We give an overview of both our Hamming and InsDel code constructions.

The Hamming Encoder $\mathsf{Enc}_{\mathsf{H},\lambda}$.

We present our formal Hamming encoding algorithm in Figure 7.1 and present an overview here. Let $\lambda \in \mathbb{N}$ be the security parameter. For any message $x \in \{0,1\}^k$, our encoder partitions x into $d = \lceil k/r(\lambda) \rceil$ blocks $x^{(1)}, \ldots, x^{(d)}$ such that $x^{(1)} \circ \cdots \circ x^{(d)} = x$; here "$\circ$" is string concatenation and $x^{(i)} \in \{0,1\}^{r(\lambda)}$ for every $i \in \{1, \ldots, d\}$.[2] Now in place of using an outer code to encode each block $x^{(i)}$, we utilize our digital signature scheme. The encoder first generates a key pair $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$ then signs the message $x^{(i)} \circ i$ as $\sigma^{(i)} \leftarrow \mathsf{Sign}_{\mathsf{sk}}(x^{(i)} \circ i)$ using the generated secret key. The encoder then computes the encoding of $x^{(i)} \circ \sigma^{(i)} \circ \mathsf{pk} \circ i$ using the code C_{in} to obtain codeword $c^{(i)}$, where pk is the public verification key generated by the encoder. The encoder then outputs a final codeword $C = c^{(1)} \circ \cdots \circ c^{(d)} \in \{0,1\}^n$. Note that whenever $r(\lambda) \geqslant k$, we sign and encode a single block. Moreover, the locality of the decoder will be larger than the final codeword length n since there is only a single block to decode. At that point, it is more efficient to either choose a larger k or simply use a classical Hamming code with similar rate and error-tolerance (i.e., just use C_{in} without signing anything).

[2] ↑For simplicity in this overview, assume $k/r(\lambda)$ is an integer.

Input : A message $x \in \{0,1\}^k$.
Output : A codeword $C \in \{0,1\}^n$.
Hardcoded : An inner Hamming code $C_{in} = (\mathsf{Enc}_{in}, \mathsf{Dec}_{in})$; $r(\cdot)$-length signature scheme $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$; and security parameter $\lambda \in \mathbb{N}$ in unary.

Sample $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$.
Set $d = \lceil k/r(\lambda) \rceil$.
Partition x into blocks $x^{(1)} \circ \cdots \circ x^{(d)}$ where $x^{(j)} \in \{0,1\}^{r(\lambda)}$ for every $j \in \{1, \ldots, d\}$ (padding the last block as necessary).
foreach $j \in \{1, \ldots, d\}$ **do**
    Compute the following values:

$$\sigma^{(j)} \leftarrow \mathsf{Sign}_{\mathsf{sk}}(x^{(j)} \circ j) \tag{7.1}$$

$$C^{(j)} = \mathsf{Enc}_{in}(x^{(j)} \circ \sigma^{(j)} \circ \mathsf{pk} \circ j) \tag{7.2}$$

Define $C := C^{(1)} \circ \cdots \circ C^{(d)} \in \{0,1\}^n$.
return C.

Figure 7.1. Encoding algorithm $\mathsf{Enc}_{\mathsf{H},\lambda}(x)$ for Hamming errors.

Strawman Decoding Algorithm.

Given $\mathsf{Enc}_{\mathsf{H},\lambda}$ in Figure 7.1, there is a natural (but insecure) decoding algorithm one may use. However, as we will see, this natural decoder cannot hope to satisfy our definition of crLDCs (Definition 7.1.1). The strawman decoder proceeds as follows. Let $x \in \{0,1\}^k$, $C = \mathsf{Enc}_{\mathsf{H},\lambda}(x) \in \{0,1\}^n$, and let $\widetilde{C} \in \{0,1\}^n$ such that $\mathsf{HAM}(\widetilde{C}, C) \leqslant \rho$. Let $i \in \{1, \ldots, k\}$ be the input given to the strawman decoder and let $\widetilde{C}$ be its oracle. Since the goal is to recover bit x_i from string $\widetilde{C}$, the strawman decoder first calculates index $j \in \{1, \ldots, d\}$ such that bit x_i resides in block $x^{(j)}$. Now since we are in the Hamming error setting, the strawman decoder views its oracle $\widetilde{C}$ as blocks $\widetilde{C}^{(1)} \circ \cdots \circ \widetilde{C}^{(d)}$ and recovers block $\widetilde{C}^{(j)}$. The strawman decoder then runs the decoder of C_{in} with input $\widetilde{C}^{(j)}$ to obtain some string $\widetilde{m}^{(j)}$ which can be viewed as some (potentially corrupt) string $\widetilde{x}^{(j)} \circ \widetilde{\sigma}^{(j)} \circ \widetilde{\mathsf{pk}} \circ \widetilde{j}$. The strawman decoder then proceeds to use the digital signature scheme to verify the contents of this decoded message by checking if $\mathsf{Verify}_{\widetilde{\mathsf{pk}}}(\widetilde{x}^{(j)} \circ j, \widetilde{\sigma}^{(j)}) \stackrel{?}{=} 1$. If verification fails, then the decoder outputs $\perp$; otherwise, the decoder outputs $\widetilde{x}^{(j)}_{i^*}$, where i^* is the index of $x^{(j)}$ that corresponds to bit x_i.

Notice that if $\widetilde{C} = C$, then this strawman decoder outputs the correct bit x_i with probability 1, satisfying Item 2 of Definition 7.1.1. However, this strawman decoder can never satisfy Item 3 if we desire error-tolerance $\rho = \Theta(1)$. Consider the following simple attack. Let $\mathcal{A}$ be a PPT adversary that operates as follows: (1) Given codeword C, the adversary $\mathcal{A}$ decodes block $C^{(1)}$ to obtain $x^{(1)} \circ \sigma^{(1)} \circ \mathsf{pk} \circ 1$. (2) $\mathcal{A}$ then generates its own key pair $(\mathsf{pk}', \mathsf{sk}')$, a message $x' = 1 - x^{(1)}$, and computes $\sigma' = \mathsf{Sign}_{\mathsf{sk}'}(x' \circ 1)$. (3) $\mathcal{A}$ then computes $C' = \mathsf{Enc}_{in}(x' \circ \sigma' \circ \mathsf{pk}' \circ 1)$ and outputs $\widetilde{C} = C' \circ C^{(2)} \circ \cdots \circ C^{(d)}$. Intuitively, this attack succeeds for two reasons. The first reason is that corruption of $C^{(1)}$ to C' is a small fraction of the total amount of corruptions allotted to transform C to $\widetilde{C}$. The second reason is that the strawman decoder relies on the public key $\mathsf{pk}'/\widetilde{\mathsf{pk}}$ to perform verification. The key to preventing this attack is addressing the second reason: the recovery of the public key. Notice that if the decoder recovered the true public key pk used by $\mathsf{Enc}_{\mathsf{H},\lambda}$, then this attack fails since the verification procedure fails and the decoder outputs bot. Thus we modify the strawman decoder to recover the true public key to obtain our final Hamming decoder.

The Hamming Decoder $\mathsf{Dec}_{\mathsf{H},\lambda}$.

We present our formal Hamming decoding algorithm in Figure 7.2 and present an overview here. Recall that our goal is to recover the public key used by $\mathsf{Enc}_{\mathsf{H},\lambda}$. While we cannot recover this key with probability 1, we can recover pk with sufficiently high probability. To do so, we utilize random sampling and with majority vote. Let $\mu \in \mathbb{N}$ be a parameter of our choice, let $x \in \{0,1\}^k$, and $C = \mathsf{Enc}_{\mathsf{H},\lambda}(x)$. Let $\widetilde{C} \in \{0,1\}^n$ be a corrupt codeword such that $\mathsf{HAM}(C, \widetilde{C}) \leqslant \rho$. Then on input $i \in \{1, \ldots, k\}$ and given oracle access to $\widetilde{C}$, the decoder $\mathsf{Dec}_{\mathsf{H},\lambda}$ attempts to recover bit x_i. It proceeds in two steps. In step 1, $\mathsf{Dec}_{\mathsf{H},\lambda}$ attempts to recover the true public key pk. Given parameter μ, $\mathsf{Dec}_{\mathsf{H},\lambda}$ uniformly samples block indexes $j_1, \ldots, j_\mu \xleftarrow{\$} \{1, \ldots, d\}$. Parsing $\widetilde{C}$ as $\widetilde{C}^{(1)} \circ \cdots \circ \widetilde{C}^{(d)}$, for each $\kappa \in \{1, \ldots, \mu\}$, the decoder $\mathsf{Dec}_{\mathsf{H},\lambda}$: (1) recovers some string $\widetilde{m}^{(j_\kappa)} \leftarrow \mathsf{Dec}_{in}(\widetilde{C}^{(j_\kappa)})$; (2) parses $\widetilde{m}^{(j_\kappa)}$ as $\widetilde{x}^{(j_\kappa)} \circ \widetilde{\sigma}^{(j_\kappa)} \circ \widetilde{\mathsf{pk}}^{(j_\kappa)} \circ \widetilde{j}$; and (3) recovers key $\widetilde{\mathsf{pk}}^{(j_\kappa)}$. The decoder then sets $\mathsf{pk}^* = \mathsf{majority}(\widetilde{\mathsf{pk}}^{(j_1)}, \ldots, \widetilde{\mathsf{pk}}^{(j_\mu)})$. In step 2, $\mathsf{Dec}_{\mathsf{H},\lambda}$ now proceeds identically to the strawman decoder, except it will use pk^* for verification. In more details, $\mathsf{Dec}_{\mathsf{H},\lambda}$ computes block index j such that x_i resides in $x^{(j)}$.

Input : An index $i \in \{1, \ldots, k\}$.
Oracle : Bitstring $\widetilde{C} \in \{0,1\}^n$.
Output : A symbol $\widetilde{x} \in \{0,1\}$ or $\perp$.
Hardcoded : An inner Hamming code $C_{in} = (\mathsf{Enc}_{in}, \mathsf{Dec}_{in})$; $r(\cdot)$-length signature scheme $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$; security parameter $\lambda \in \mathbb{N}$ in unary; and parameter $\mu \in \mathbb{N}$ (to be determined later).

Set $d = \lceil k/r(\lambda) \rceil$ and $\mathsf{bl} = n/d$. `/* Compute number of blocks` d `and block length` bl. `*/`
Sample $j_1, \ldots, j_\mu \xleftarrow{\$} \{1, \ldots, d\}$ uniformly at random.
Initialize $\mathsf{pk}^*, \mathsf{pk}_1, \ldots, \mathsf{pk}_\mu = 0$.
foreach $\kappa \in \{1, \ldots, \mu\}$ **do** `/* Public key pk recovery through majority sampling. */`
    Obtain string $\widetilde{m}^{(j_\kappa)} \leftarrow \mathsf{Dec}_{in}\left(\widetilde{C}[(j_\kappa - 1) \cdot \mathsf{bl} + 1, j_\kappa \cdot \mathsf{bl}]\right)$.
    Parse $\widetilde{m}^{(j_\kappa)}$ as $\widetilde{x}^{(j_\kappa)} \circ \widetilde{\sigma}^{(j_\kappa)} \circ \widetilde{\mathsf{pk}}^{(j_\kappa)} \circ \widetilde{j}$.
    Set $\mathsf{pk}_\kappa = \widetilde{\mathsf{pk}}^{(j_\kappa)}$.
Set $\mathsf{pk}^* = \mathsf{majority}(\mathsf{pk}_1, \ldots, \mathsf{pk}_\mu)$.
Compute $j \in \{1, \ldots, d\}$ such that $(j-1) \cdot r(\lambda) < i \leqslant j \cdot r(\lambda)$.
Obtain string $\widetilde{m}^{(j)} \leftarrow \mathsf{Dec}_{in}\left(\widetilde{C}[(j-1) \cdot \mathsf{bl} + 1, j \cdot \mathsf{bl}]\right)$. `/* Recover block where` x_i `should reside. */`
Parse $\widetilde{m}^{(j)}$ as $\widetilde{x}^{(j)} \circ \widetilde{\sigma}^{(j)} \circ \widetilde{\mathsf{pk}}^{(j)} \circ \widetilde{j}$.
if $\mathsf{Verify}_{\mathsf{pk}^*}(\widetilde{x}^{(j)} \circ j, \widetilde{\sigma}^{(j)}) = 0$ **then**
    return $\perp$.
return $\widetilde{x}^{(j)}_{i^*}$ *for* $i^* = i - (j-1) \cdot r(\lambda)$.

Figure 7.2. Decoding algorithm $\mathsf{Dec}^{\widetilde{C}}_{\mathsf{H},\lambda}(i)$ for Hamming errors.

Then the decoder obtains $\widetilde{m}^{(j)} \leftarrow \mathsf{Dec}_{in}(\widetilde{C}^{(j)})$, parses this string as $\widetilde{x}^{(j)} \circ \widetilde{\sigma}^{(j)} \circ \widetilde{\mathsf{pk}}^{(j)} \circ \widetilde{j}$, then checks if $\mathsf{Verify}_{\mathsf{pk}^*}(\widetilde{x}^{(j)} \circ j, \widetilde{\sigma}^{(j)}) \stackrel{?}{=} 1$. If verification fails, $\mathsf{Dec}_{\mathsf{H},\lambda}$ outputs $\perp$; else it outputs bit $\widetilde{x}^{(j)}_{i^*}$, where i^* is the index of $x^{(j)}$ that corresponds to bit x_i. Note that $\mathsf{Dec}_{\mathsf{H},\lambda}$ using its own computed value j here is crucial, otherwise it is possible for an adversary to simply swap two blocks $C^{(j_1)}$ and $C^{(j_2)}$ such that $x^{(j_1)} \neq x^{(j_2)}$, which would violate Item 3 of Definition 7.1.1.

Hamming crLDC Security Proof Overview

The main technical challenge of proving Theorem 7.1.1 is showing that $(\mathsf{Enc}_{\mathsf{H},\lambda}, \mathsf{Dec}_{\mathsf{H},\lambda})$ satisfies Items 3 and 4 of Definition 7.1.1. We give a high level overview of the proof here and present the formal proof in Section 7.4.

Towards Item 3, we begin by analyzing the probability that $\mathsf{Dec}^{\widetilde{C}}_{\mathsf{H},\lambda}(i) \in \{x_i, \perp\}$ for any $i \in \{1, \ldots, k\}$ and $\widetilde{C} \in \{0,1\}^n$ such that $\mathsf{HAM}(\widetilde{C}, C) \leqslant \rho$. If $\mathsf{Dec}^{\widetilde{C}}_{\mathsf{H},\lambda}(i) = x_i$, then $\mathsf{Verify}_{\mathsf{pk}^*}(\widetilde{x}^{(j)} \circ j, \widetilde{\sigma}^{(j)}) = 1$ and $\widetilde{x}^{(j)}_{i^*} = x_i$. Conditioning on $(\widetilde{x}^{(j)}, \widetilde{\sigma}^{(j)})$ not being a signature forgery, then the verification succeeding implies that $\widetilde{x}^{(j)} = x^{(j)}$ and $\widetilde{\sigma}^{(j)} = \sigma^{(j)}$. Now, the verification succeeds whenever $\mathsf{pk}^* = \mathsf{pk}$. By Chernoff, we can ensure that $\mathsf{pk}^* = \mathsf{pk}$ with high probability (depending on μ) as long as more than half of the (possibly corrupt) blocks $\widetilde{C}^{(1)}, \ldots, \widetilde{C}^{(d)}$ have less than ρ_{in}-fraction of Hamming errors. Setting $\rho = c \cdot \rho_{in}$ for any positive constant $c < 1/2$ ensure this is the case. Here we conditioned on the fact that $\widetilde{\sigma}^{(i)}$ was not a forgery for $\widetilde{x}^{(j)} \circ j$. Appealing to the security of the digital signature scheme, the probability that $\widetilde{\sigma}^{(i)}$ is not a signature is at least $1 - \varepsilon_{\Pi}(\lambda)$, where $\varepsilon_{\Pi}(\lambda)$ is a negligible function for the security of the digital signature scheme Π. As index i was arbitrary here, by union bound over each index i, we establish Item 3 of Definition 7.1.1 for $p = 1 - \exp(-\mu(1/2 - c)^2 \big/ 2(1 - c))$ and $\varepsilon_{\mathsf{F}}(\lambda) = k \cdot \varepsilon_{\Pi}(\lambda)$, noting that $\varepsilon_{\mathsf{F}}(\lambda)$ is negligible in λ since k is a polynomial in λ.

Towards Item 4, by setting $\rho = c \cdot \rho_{in}$ for positive constant $c < 1/2$, we ensure that the number of corrupt blocks $\widetilde{C}^{(1)}, \ldots, \widetilde{C}^{(d)}$ with less than ρ_{in}-fraction of corruptions is more than half. Let $\mathcal{J} = \{j \colon \mathsf{HAM}(\widetilde{C}^{(j)}, C^{(j)}) \leqslant \rho_{in}\}$. Then we have $|\mathcal{J}| \geqslant d/2$. Moreover, for any $j \in \mathcal{J}$, we have that $\mathsf{Dec}_{in}(\widetilde{C}^{(j)}) = x^{(j)} \circ \sigma^{(j)} \circ \mathsf{pk} \circ j$; i.e., we can recover the uncorrupt message, signature, public key, and index. Now for any $i \in \{1, \ldots, k\}$, if bit x_i lies in $x^{(j)}$ for some $j \in \mathcal{J}$, the probability $\mathsf{Dec}_{\mathsf{H},\lambda}$ outputs x_i depends on the probability that $\mathsf{Dec}_{\mathsf{H},\lambda}$ correctly recovers $\mathsf{pk}^* = \mathsf{pk}$. We can suitably choose parameter μ to ensure that $\Pr[\mathsf{pk}^* = \mathsf{pk}] > 2/3$. This along with $|\mathcal{J}| \geqslant d/2$ implies that $|\mathsf{Good}(\widetilde{C})| \geqslant (1/2) \cdot k$, and thus we can set $\delta = 1/2$. Notice here that $|\mathcal{J}| \geqslant d/2$ holds *for any* corrupt $\widetilde{C}$ by our choice of ρ. This implies that $|\mathsf{Good}(\widetilde{C})| \geqslant (1/2) \cdot k$ holds for any corrupt $\widetilde{C}$, not just ones produced by a PPT adversary $\mathcal{A}$. Thus Item 4 holds with $\delta = 1/2$ and $\varepsilon_{\mathsf{L}}(\lambda) := 0$.

7.2.2 InsDel crLDC Construction

Like our Hamming code construction for Theorem 7.1.1, the main technical ingredients of Theorem 7.1.3 are a digital signature scheme and a suitable inner code. In our case, we choose the Schulman-Zuckerman InsDel code, or SZ code, as our inner code. Our construction additionally requires the use of the noisy binary search algorithm due to Block et al. [48]. In particular, simply replacing C_{in} in $\mathsf{Enc}_{\mathsf{H},\lambda}$ of Figure 7.1 with the SZ code and using $\mathsf{Dec}_{\mathsf{H},\lambda}$ does not yield an interesting InsDel crLDC.

Challenges to Decoding Insertion-Deletion Errors.

While Hamming errors correspond to bit flips in the final codeword, insertion-deletion errors allow an adversary to insert symbols into and delete symbols from any codeword given to it. Given $C = \mathsf{Enc}'_{\mathsf{H},\lambda}(x)$, where $\mathsf{Enc}'_{\mathsf{H},\lambda}$ is identical to $\mathsf{Enc}_{\mathsf{H},\lambda}$ except we use the SZ code as the code C_{in}, there is a simple attack to ensure that $\mathsf{Dec}_{\mathsf{H},\lambda}$ always outputs $\perp$. The adversary simply transforms $C = C^{(1)} \circ \ldots \circ C^{(d)}$ to corrupt codeword $\widetilde{C} = C_1^{(d)} \circ C^{(1)} \circ \cdots \circ C^{(d-1)} \circ C_0^{(d)}$, where $C_0^{(d)}$ is the first half of $C^{(d)}$ and $C_1^{(d)}$ is the second half of $C^{(d)}$. Now recovery of any block j and the public key pk is impossible. Note that this implies that $(\mathsf{Enc}_{\mathsf{H},\lambda}, \mathsf{Dec}_{\mathsf{H},\lambda})$ is an InsDel crLDC with $\delta = 0$; however, we can achieve larger δ by leveraging noisy binary search.

Noisy Binary Search Overview.

We leverage the noisy binary search algorithm of Block et al. [48] to overcome the above challenges. To begin, our insertion-deletion encoder is nearly identical to $\mathsf{Enc}_{\mathsf{H},\lambda}$ with the following changes: (1) we use the SZ code as our inner code C_{in}; and (2) we additionally pad each block encoded by the SZ code with a suitable number of 0s before and after each codeword, before concatenating all of them together to yield the final codeword. To differentiate from $\mathsf{Enc}_{\mathsf{H},\lambda}$, we let $c^{(j)} = \mathsf{SZ.Enc}(x^{(j)} \circ \sigma^{(j)} \circ \mathsf{pk} \circ j)$ denote the encoded signed message block and let $C^{(j)}$ denote $c^{(j)}$ padded with an appropriate number of 0s before and after the codeword. This padding is necessary for the noisy binary search algorithm.

Now given some (possibly corrupt) codeword $\widetilde{C}$ and for any desired index $i \in \{1, \ldots, k\}$, our decoding algorithm leverages the noisy binary search algorithm to search C' for some (possibly corrupt) block $\widetilde{c}^{(j)}$ which contains the desired bit x_i. So long as both $\widetilde{C}$ and $\widetilde{c}^{(j)}$ are "not too corrupt", then the noisy binary search algorithm outputs $\widetilde{c}^{(j)}$ with high probability. Then we can decode the string $\widetilde{c}^{(j)}$ as $\widetilde{x}^{(j)} \circ \widetilde{\sigma}^{(j)} \circ \widetilde{\mathsf{pk}} \circ \widetilde{j}$ and use the signature scheme verification algorithm to ensure that $\widetilde{x}^{(j)}$ is correct.

To understand the noisy binary search (NBS) algorithm and its guarantees, we require the notion of γ-*goodness*. For two bit strings $x, y \in \{0,1\}^*$, we say that y is γ-*good* with respect to x if $\mathsf{ED}(x, y) \leqslant \gamma$, where ED is the normalized edit distance. The notion of γ-goodness—albeit under different formal definitions—has been useful in the design and analysis of depth-robust graphs, which are a combinatorial object used extensively in the study of so-called memory-hard functions [4, 7, 112], and it is essential to the success of the NBS algorithm. Intuitively, for a fixed "correct" ordered list of strings $A = (a_1, \ldots, a_n)$ and some other list of strings $B = (b_1, \ldots, b_{n'})$, the NBS algorithm finds any string b_j that is γ-good with respect to the string a_j for $j \in \{1, \ldots, n\}$, except with negligible probability. In our context, these bit strings b_j correspond to (possibly corrupt) blocks in the (possibly corrupt) codeword. Given a tolerance parameter $\rho^* \in (0, 1/2)$, the NBS algorithm on input $j \in \{1, \ldots, n\}$ outputs the string b_j for at least $(1 - \rho^*) \cdot n$ fraction of the γ-good indices j, except with negligible probability. Moreover, the algorithm runs in time that is poly-logarithmic in n'; note that this is only possible by allowing NBS to fail on some small fraction of γ-good indices, otherwise the algorithm requires $\Omega(n')$ time. When translating NBS to operate encodings $\widetilde{c}^{(j)}$ rather than arbitrary bit strings, the algorithm utilizes a so-called block decoding algorithm $\mathsf{BlockDecode}$ to find the string $\widetilde{c}^{(j)}$ within the larger string $\widetilde{C}$. The algorithm $\mathsf{BlockDecode}$ takes as input an index $i \in \{1, \ldots, |\widetilde{C}|\}$, and as long as $\widetilde{c}^{(j)}$ is a γ-good block and the index i falls within a ball around $\widetilde{C}^{(j)}$ in $\widetilde{C}$, then $\mathsf{BlockDecode}$ outputs the block $\widetilde{c}^{(j)}$, except with probability at most γ. Our construction leverages both the $\mathsf{BlockDecode}$ and NBS algorithms along with a suitable digital signature scheme Π. We now formally present our encoding and decoding algorithms.

The Encoder $\mathsf{Enc}_{\mathsf{I},\lambda}$.

We present our formal encoding algorithm $\mathsf{Enc}_{\mathsf{I},\lambda}$ in Figure 7.3. As our insertion-deletion encoder is a modification of our Hamming encoder, we highlight the differences in blue text and with comments in Figure 7.3. On input message $x \in \{0,1\}^k$ and given security parameter $\lambda \in \mathbb{N}$ (in unary) along with a constant α (specified by the NBS and $\mathsf{BlockDecode}$ algorithms), the encoder $\mathsf{Enc}_{\mathsf{I},\lambda}$ first samples a public/private key pair $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$. Next, the message x is partitioned into $d = \lceil k/r(\lambda) \rceil$ blocks $x^{(1)} \circ \cdots \circ x^{(d)}$, each of size $r(\lambda)$ bits. For every block $j \in \{1, \ldots, d\}$, the signature $\sigma^{(j)} \leftarrow \mathsf{Sign}_{\mathsf{sk}}(x^{(j)} \circ j)$ is computed, and a new block $m^{(j)} := x^{(j)} \circ \sigma^{(j)} \circ \mathsf{pk} \circ j$ is formed. This block is then encoded as $c^{(j)} = \mathsf{SZ.Enc}_\lambda(m^{(j)})$, and computes a zero-buffered block $C^{(j)} = (0^{\alpha \cdot r(\lambda)} \circ c^{(j)} \circ 0^{\alpha \cdot r(\lambda)})$, where α is a suitably chosen constant given by [48] (see Lemma 7.5.3). The final codeword C is the concatenation of all buffered blocks; i.e., $C := C^{(1)} \circ \cdots \circ C^{(d)}$. As with our Hamming encoder, whenever $r(\lambda) \geqslant k$, it is efficient to encode x with the SZ code, as we lose all locality benefits.

The Decoder $\mathsf{Dec}_{\mathsf{I},\lambda}$.

We present the formal decoding algorithm $\mathsf{Dec}_{\mathsf{I},\lambda}$ in Figure 7.4 and give a high-level overview here. As our insertion-deletion decoder is a modification of our Hamming decoder, we highlight the differences in blue text and with comments in Figure 7.4. Let $x \in \{0,1\}^k$ and $C = \mathsf{Enc}_{\mathsf{I},\lambda}(x)$, and suppose that $\widetilde{C} \in \{0,1\}^{n'}$ is a corrupt codeword such that $\mathsf{ED}(C, \widetilde{C}) \leqslant \rho$ for some n'. Given oracle access to $\widetilde{C}$ and an index $i \in \{1, \ldots, k\}$ as input, the decoder attempts to recover bit x_i of the original message x. Calling back to the encoding algorithm, bit x_i resides in block $x^{(j)}$ for j satisfying $(j-1) \cdot r(\lambda) < i \leqslant j \cdot r(\lambda)$ (Line 3 of Figure 7.3). Thus recovering x_i is done by recovering block $x^{(j)}$ from corrupt codeword $\widetilde{C}$. Assuming $\mathsf{Dec}_{\mathsf{I},\lambda}$ can recover $x^{(j)}$, the decoder simply outputs $x_i = x^{(j)}_{i^*}$ for $i^* = i - (j-1) \cdot r(\lambda)$; otherwise, our goal will be for $\mathsf{Dec}_{\mathsf{I},\lambda}$ to output $\perp$.

Same line as the Hamming decoder, our insertion-deletion decoder proceeds in two steps: first, the decoder attempts to recover the public key pk used to encode message x as C by

Input : A message $x \in \{0,1\}^k$.
Output : A codeword $C \in \{0,1\}^n$.
Hardcoded : The SZ InsDel code $(\mathsf{SZ.Enc}, \mathsf{SZ.Dec})$; $r(\cdot)$-length signature scheme $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$; constant $\alpha \in \mathbb{N}$; and security parameter $\lambda \in \mathbb{N}$ in unary.

Sample $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$.
Set $d = \lceil k/r(\lambda) \rceil$.
Partition x into blocks $x^{(1)} \circ \cdots \circ x^{(d)})$ where $x^{(j)} \in \{0,1\}^{r(\lambda)}$ for every $j \in \{1, \ldots, d\}$ (padding the last block as necessary).
foreach $j \in \{1, \ldots, d\}$ **do**
    Compute the following values: `/* Differs from` $\mathsf{Enc}_{\mathsf{H},\lambda}$`. */`

$$\sigma^{(j)} \leftarrow \mathsf{Sign}_{\mathsf{sk}}(x^{(j)} \circ j) \tag{7.3}$$

$$c^{(j)} = \mathsf{SZ.Enc}(x^{(j)} \circ \sigma^{(j)} \circ \mathsf{pk} \circ j) \tag{7.4}$$

$$C^{(j)} = (0^{\alpha \cdot \mathsf{ml}} \circ c^{(j)} \circ 0^{\alpha \cdot \mathsf{ml}}), \tag{7.5}$$

    where $\mathsf{ml} = |x^{(j)} \circ \sigma^{(j)} \circ \mathsf{pk} \circ j|$. `/* ml is the same for every` j`. */`
Define $C := C^{(1)} \circ \cdots \circ C^{(d)} \in \{0,1\}^n$.
return C

Figure 7.3. Encoding algorithm $\mathsf{Enc}_{\mathsf{I},\lambda}(x)$ for insertion-deletion errors.

running $\mathsf{Enc}_{\mathsf{I},\lambda}$; and second, the decoder attempts to recover some (possibly corrupt) block $\widetilde{m} = (\widetilde{x} \circ \widetilde{\sigma} \circ \widetilde{\mathsf{pk}} \circ \widetilde{j})$. The block $\widetilde{m}$ can then be parsed and verified using $\mathsf{Verify}_{\mathsf{pk}}$; that is, the decoder runs $\mathsf{Verify}_{\mathsf{pk}}(\widetilde{x} \circ j, \widetilde{\sigma})$ and outputs $\widetilde{x}_{i^*}$ if $\mathsf{Verify}_{\mathsf{pk}}(\widetilde{x} \circ j, \widetilde{\sigma}) = 1$ and outputs $\perp$ otherwise. Note here that (1) the public key pk recovered in the first step is not necessarily equal to string $\widetilde{\mathsf{pk}}$ parsed from $\widetilde{m}$ in the second step; and (2) pk and desired block index j are used to verify the string $\widetilde{x}$ rather than the parsed public key $\widetilde{\mathsf{pk}}$ and parsed index $\widetilde{j}$.

Towards the first goal of recovering the public key pk, let $\mu \in \mathbb{N}$ be a parameter (to be determined later). The decoder attempts to recover μ number of public keys, then take the majority of these recovered keys as the "true" public key pk. To do so, the decoder leverages the algorithm $\mathsf{BlockDecode}$. On input $\mathrm{j} \in [n]$, the algorithm $\mathsf{BlockDecode}$ queries the received word $\widetilde{C}$ in a large ball round index j for a block to decode. For example, if $\widetilde{C} = C = \mathsf{Enc}_{\mathsf{I},\lambda}(x)$, then $\mathsf{BlockDecode}(1)$ would return $c^{(1)}$ (as defined in Equation (7.4) of Figure 7.3). However, the algorithm $\mathsf{BlockDecode}$ is only guaranteed to return a correct block so long as the portion

of $\widetilde{C}$ it begins searching in is "not too corrupt". This intuition is captured by the notion of a γ-*good block*, which informally states that the edit distance between the uncorrupt block and the corrupt block is at most γ. Then, so long as the number of γ-good blocks is large (for some constant γ), the $\mathsf{BlockDecode}$ algorithm will succeed most of the time. We formalize the notion of γ-good blocks in Definition 7.5.2 and the guarantees of $\mathsf{BlockDecode}$ in Lemma 7.5.4. This guarantees that with good probability (which depends on γ and the value μ), the public key recovered by the decoding algorithm (Line 11) is the correct public key used by the encoder; in particular, we can suitably choose μ such that $\mathsf{pk}^* = \mathsf{pk}$ with high probability via a Chernoff bound so long as the number of γ-good blocks is greater than $d/2$.

Conditioned on the decoder obtaining some public key pk^* in Line 11 of Figure 7.4, the decoder proceeds in the next task of recovering the desired block $\widetilde{m}^{(j)}$. For this task, the decoder leverages the noisy binary search algorithm NBS. After running NBS on input j and obtaining some $\widetilde{m}^{(j)}$, assuming that $\widetilde{m}^{(j)} \neq \perp$, the decoder parses $\widetilde{m}^{(j)}$ as the string $(\widetilde{x}^{(j)} \circ \widetilde{\sigma}^{(j)} \circ \widetilde{\mathsf{pk}}^{(j)} \circ \widetilde{j})$. Then, as mentioned before, the decoder proceeds to verify that the signature $\widetilde{\sigma}^{(j)}$ is a valid signature for the string $\widetilde{x}^{(j)} \circ j$ using Verify and the recovered public key pk^*. It is important to again note that the index j used in the verification may not be the same as $\widetilde{j}$, and that pk^* may not be the same as $\widetilde{\mathsf{pk}}^{(j)}$. One could alternatively choose for the decoder to output $\perp$ if either $j \neq \widetilde{j}$ or $\mathsf{pk}^* \neq \widetilde{\mathsf{pk}}^{(j)}$; however, this would give the attacker an easy way to force the decoder to output $\perp$ on a large fraction of indices, something we want to avoid all together. Finally, if the verification succeeds, the decoder outputs bit $\widetilde{x}^{(j)}_{i^*}$ for $i^* = i - (j-1) \cdot r(\lambda)$.

InsDel crLDC Security Proof Overview

The main technical challenge of our construction is showing our construction satisfies Items 3 and 4 of Definition 7.1.1. We present the formal proof of security in Section 7.5 and give a high level overview in the remainder of this section.

Towards analyzing Item 3, our analysis proceeds in much the same way as for our Hamming crLDC. We directly analyze the probability that $\mathsf{Dec}^{\widetilde{C}}_{\mathsf{I},\lambda}(i) \in \{x_i, \perp\}$ for any $i \in \{1, \ldots, k\}$

Input : An index $i \in [k]$.
Oracle : Bitstring $\widetilde{C} \in \{0,1\}^{n'}$ for some n'.
Output : A symbol $\widetilde{x} \in \{0,1\}$ or $\perp$.
Hardcoded : The algorithms $\mathsf{BlockDecode}$ and NBS; $r(\cdot)$-length signature scheme $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$; block length n; security parameter $\lambda \in \mathbb{N}$ in unary; and parameter $\mu \in \mathbb{N}$ (to be determined later).

Set $d = \lceil k/r(\lambda) \rceil$.
Uniformly at random sample $i_1, \ldots, i_\mu \xleftarrow{\$} [n]$. /* Differs from $\mathsf{Dec}_{\mathsf{H},\lambda}$. */
Initialize $\mathsf{pk}^*, \mathsf{pk}_1, \ldots, \mathsf{pk}_\mu = 0$.
foreach $\kappa \in \{1, \ldots, \mu\}$ **do**
    Obtain string $\widetilde{m}^{(i_\kappa)} \leftarrow \mathsf{BlockDecode}^{\widetilde{C}}(i_\kappa)$. /* Differs from $\mathsf{Dec}_{\mathsf{H},\lambda}$. */
    if $\widetilde{m}^{(i_\kappa)} = \perp$ **then** /* Differs from $\mathsf{Dec}_{\mathsf{H},\lambda}$. */
        Set $\mathsf{pk}_\kappa = \perp$.
        continue
    Parse $\widetilde{m}^{(i_\kappa)}$ as $(\widetilde{x}^{(i_\kappa)} \circ \widetilde{\sigma}^{(i_\kappa)} \circ \widetilde{\mathsf{pk}}^{(i_\kappa)} \circ \widetilde{i})$.
    Set $\mathsf{pk}_\kappa = \widetilde{\mathsf{pk}}^{(i_\kappa)}$.
Set $\mathsf{pk}^* = \mathsf{majority}(\mathsf{pk}_1, \ldots, \mathsf{pk}_\mu)$.
Compute $j \in \{1, \ldots, d\}$ such that $(j-1) \cdot r(\lambda) < i \leqslant j \cdot r(\lambda)$.
Obtain string $\widetilde{m}^{(j)} \leftarrow \mathsf{NBS}^{C'}(j)$. /* Differs from $\mathsf{Dec}_{\mathsf{H},\lambda}$. */
if $\widetilde{m}^{(j)} = \perp$ *or* $\mathsf{pk}^* = \perp$ **then** /* Differs from $\mathsf{Dec}_{\mathsf{H},\lambda}$. */
    return $\perp$.
Parse $\widetilde{m}^{(j)}$ as $(\widetilde{x}^{(j)} \circ \widetilde{\sigma}^{(j)} \circ \widetilde{\mathsf{pk}}^{(j)} \circ \widetilde{j})$.
if $\mathsf{Verify}_{\mathsf{pk}^*}(\widetilde{x}^{(j)} \circ j, \widetilde{\sigma}^{(j)}) = 0$ **then**
    return $\perp$
return $\widetilde{x}^{(j)}_{i^*}$ *for* $i^* = i - (j-1) \cdot r(\lambda)$.

Figure 7.4. Decoding algorithm $\mathsf{Dec}^{\widetilde{C}}_{\mathsf{I},\lambda}(i)$ for insertion-deletion errors.

and $\widetilde{C} \in \{0,1\}^{n'}$ such that $\mathsf{ED}(\widetilde{C}, C) \leqslant \rho$ for $C = \mathsf{Enc}_{\mathsf{I},\lambda}(x)$. The proof proceeds identically to the Hamming crLDC with the following key changes. First, when recovering the public key, we additionally must consider the success probability of the algorithm $\mathsf{BlockDecode}$ to recover our corrupt codeword blocks. The success probability directly affects the parameters of our construction that are needed to apply a Chernoff bound to ensure that we recover $\mathsf{pk}^* = \mathsf{pk}$ with high probability. Second, when recovering block j where index i resides, we must use the noisy binary search algorithm to recover this block. Thus we additionally take into consideration the probability of success of our noisy binary search algorithm. As we

show in Section 7.5, the guarantees of both BlockDecode and NBS allow us to ensure that Item 3 is satisfied via careful selection of parameters.

Towards Item 4, the proof proceeds again nearly identically to the Hamming crLDC case, except again we must take into consideration the recovery of the public key $\mathsf{pk}^* = \mathsf{pk}$ via the algorithm BlockDecode, and the correct recovery of a block j with the algorithm NBS. Note that except with negligible probability, the noisy binary search algorithm will correctly recover any block that is γ-good with probability greater than $2/3$ (under suitable parameter choices). This then directly translates to the fraction δ of indices we are able to decode from for Item 4.

7.2.3 Related Work

Classical insertion-deletion codes were initially studied by Levenstein [185], and since then there has been extensive study into the construction of insertion-deletion codes; see the surveys of [205, 208, 245]. A recent line of work answered a long standing open in the construction of k-deletion correcting codes with optimal redundancy [241–243]. There has also been a line of work studying the random codes with positive information rate that are able to correct a large fraction of deletions [143, 179], and constant rate codes that are resilient to a constant fraction of insertion-deletion errors with efficient encoding and decoding (i.e., polynomial time) were studied extensively in [70, 85, 86, 88, 89, 142–144, 150, 151, 230]. Another direction in the study of insertion-deletion codes is extending the ideas of list decoding to this context. List decodable codes are error-correcting codes that are resilient to a larger fraction of errors at the cost of outputting a small list of potential codewords [144, 152, 190]. Haeupler and Shahrasbi [151] study constructions of explicit synchronization strings which can be "locally decoded" in the following sense. Each index of this string is computable using values which are located at a small number of other indices in the string. These explicit and locally decodable synchronization strings are used to imply near linear time interactive coding schemes for InsDel errors.

Lipton [188] initiated the study of codes which are resilient to errors introduced by computationally bounded channels; several follow-up works adopt this channel model which

allows for the construction of Hamming codes with better parameters than their classical counterparts [146, 188, 207, 233]. It has been argued that any real-world communication channel can be reasonably modeled as a computationally bounded channel [60, 188], and this notion is well-motivated by channels in the real-world, all of which have some sort of limitations on their computation. One can reasonably expect error patterns encountered in nature to be modeled by some (possibly unknown) probabilistic polynomial time algorithm. This channel model has also been extended to the locally decodable setting for both Hamming errors [57, 60, 155, 156, 214] and, more recently, the insertion-deletion setting [46].

Ben-Sasson et al. [37] introduced the notion of a relaxed locally decodable code. These codes admit local decoding algorithms with the additional property that the decoder is allowed to output a symbol $\perp$ which represents that the decoder does not know the correct value. This relaxation allows [37] to construct locally decodable codes with much better parameters than their classical counterparts; in particular, they achieve codes which are resilient to a constant fraction of Hamming errors, have constant locality, and have encoding length $k^{1+\epsilon}$ for small ϵ. In a follow-up work, Gur, Ramnarayan, and Rothblum [141] introduce and construct relaxed locally correctable codes (rLCC) for Hamming errors. These codes admit local correction algorithms which can correct corrupt symbols of an encoded message via querying a few locations into the received word. Their construction achieves significantly better parameters than classical Hamming LCC s: they achieve constant locality, constant error-tolerance, and polynomial encoding length. Furthermore, their rLCC is also a a rLDC since their encoding is systematic (i.e., the message is a substring of the codeword). Blocki, Gandikota, Grigorescu, and Zhou [57] study Hamming rLDCs and rLCC s in the context of computationally bounded channels (crLDC and crLCC). In particular, our work directly adapts their computation model but for insertion-deletion errors. The dual assumption of both a computationally bounded channel and a relaxed LDC allows [57] to construct crLCC s and crLDCs which achieve constant rate, constant error-tolerance, and poly-logarithmic locality, further improving on the results of Ben-Sasson et al. and Gur, Ramnarayan, and Rothblum [37, 141].

The study of insertion-deletion LDCs is scarce. To the best of our knowledge, all prior InsDel LDC results rely on the so-called "Hamming-to-InsDel" compiler of Ostrovsky and

Paskin-Cherniavsky [215]. This compiler transforms any Hamming LDC into an InsDel LDC in a manner that (approximately) preserves the rate and error-tolerance of the underlying Hamming LDC at the cost of a poly-logarithmic increase in the locality. Block et al. [48] reprove the result of [215] and give a conceptually simpler analysis using techniques borrowed from the study of a cryptographic object known as memory-hard functions [4–10, 39, 59, 62, 83, 117]. Cheng, Li and Zheng [90] propose the notion of locally decodable codes with randomized encoding, in both the Hamming and edit distance regimes. They study such codes in various settings, including where the encoder and decoder share randomness, or the channel is oblivious to the codeword, and hence adds error patterns non-adaptively. In the construction of their insertion-deletion codes, they directly invoke the compiler of [215] and obtain with block length $O(k)$ or $O(k \cdot \log(k))$ and polylog(k) locality for message length k. Recently, Block and Blocki [46] extend this compiler to the private-key setting of [214] where the encoder and decoder are assumed to share a secret key unknown to the channel, and to the resource-bounded setting of [60] where the channel is assumed to be resource constrained in some way (e.g., the channel is a low-depth circuit). We remark that it is likely that applying the "Hamming-to-InsDel" compiler to the crLDC of Blocki et al. [57] would yield an InsDel crLDC, though this has not been formally claimed or proven in prior work.

Finally, there has been recent progress in deriving lower bounds for insertion-deletion LDCs. Blocki et al. [56] proved that InsDel LDCs with constant locality require exponential block length, also showing that linear 2-query InsDel LDCs do not exist. This makes it all the more surprising that one can achieve a constant rate InsDel crLDC in the poly-logarithmic locality regime.

7.3 Preliminaries

We let $\lambda \in \mathbb{N}$ denote the security parameter. For any $n \in \mathbb{Z}^+$ we let $[n] := \{1, \ldots, n\}$. A function $\mu \colon \mathbb{N} \to \mathbb{R}_{\geqslant 0}$ is said to be negligible if $\mu(n) = o(1/|p(n)|)$ for any fixed non-zero polynomial p. We write PPT to denote probabilistic polynomial time. For any randomized algorithm A, we let $y \leftarrow A(x)$ denote the process of obtaining output y from algorithm A

on input x. For a finite set S, we let $s \xleftarrow{\$} S$ denote the process of sampling elements of S uniformly at random.

We use "$\circ$" to denote the string concatenation operation. For bitstring $x \in \{0,1\}^*$, we use subscripts to denote individual bits of x; e.g., $x_i \in \{0,1\}$ is the i-th bit of x. Additionally, we often partition a bitstring $x \in \{0,1\}^k$ into some number of blocks d of equal length; e.g., $x = (x^{(1)} \circ \cdots \circ x^{(d)})$ where $x^{(j)} \in \{0,1\}^{k/d}$ for all $j \in [d]$. We also utilize array notation when convenient: e.g., for bitstring $x \in \{0,1\}^k$ and indices $a, b \in [k]$ such that $a \leqslant b$, we let $x[a,b] := (x_a \circ x_{a+1} \circ \cdots \circ x_b)$. For two strings $x \in \{0,1\}^k$ and $y \in \{0,1\}^*$, we define $\mathsf{ED}(x,y)$ as the minimum number of insertions and deletions required to transform x into y (or vice versa), normalized by $2k$.

In this work, we utilize *digital signatures* and give the formal definition below.

Definition 7.3.1 (Digital Signature Scheme)**.** *A* digital signature scheme *with signatures of length $r(\cdot)$ is a tuple of PPT algorithms $\Pi = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ satisfying the following properties:*

1. Gen *is the key generation algorithm and takes as input a security parameter 1^λ and outputs a pair of keys $(\mathsf{pk}, \mathsf{sk}) \in \{0,1\}^* \times \{0,1\}^*$, where pk is the public key and sk is the secret/private key. It is assumed that $|\mathsf{pk}|, |\mathsf{sk}| \geqslant \lambda$ are polynomial in λ, and that λ can be efficiently determined from pk or sk. Without loss of generality, we assume that $|\mathsf{pk}| = r(\lambda)$.*

2. Sign *is the signing algorithm and takes as input secret key sk and message $m \in \{0,1\}^*$ of arbitrary length and outputs a signature $\sigma \leftarrow \mathsf{Sign}_{\mathsf{sk}}(m) \in \{0,1\}^{r(\lambda)}$, where Sign runs in time $\mathrm{poly}(|\mathsf{sk}|, |m|)$.*

3. Verify *is the deterministic verification algorithm that takes as input public key pk, message m, and signature σ, and outputs a bit $b = \mathsf{Verify}_{\mathsf{pk}}(m, \sigma) \in \{0,1\}$. Moreover Verify run in time $\mathrm{poly}(r(\lambda), |m|)$.*

Additionally, we require the following two properties:

Signature experiment $\mathsf{Sig\text{-}forge}_{\Pi,\mathcal{A}}(\lambda)$

1. Obtain $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$.
2. Adversary $\mathcal{A}$ is given pk as input and given oracle access to the algorithm $\mathsf{Sign}_{\mathsf{sk}}(\cdot)$. We denote this as $\mathcal{A}^{\mathsf{Sign}_{\mathsf{sk}}(\cdot)}(\mathsf{pk})$. Let Q denote the set of all queries made by $\mathcal{A}$ to its oracle. The adversary outputs pair (m, σ).
3. The adversary $\mathcal{A}$ wins if and only if (a) $\mathsf{Verify}_{\mathsf{pk}}(m, \sigma) = 1$; and (b) $m \notin Q$. If $\mathcal{A}$ wins, we define $\mathsf{Sig\text{-}forge}_{\Pi,\mathcal{A}}(\lambda) := 1$; otherwise we define $\mathsf{Sig\text{-}forge}_{\Pi,\mathcal{A}}(\lambda) := 0$.

Figure 7.5. Description of the signature forgery experiment Sig-forge.

1. ***Completeness:*** *For all messages* $m \in \{0,1\}^*$ *and all* $(\mathsf{pk}, \mathsf{sk}) \in \mathrm{supp}(\mathsf{Gen}(1^\lambda))$*, we have*[3]

$$\mathsf{Verify}_{\mathsf{pk}}(m, \mathsf{Sign}_{\mathsf{sk}}(m)) = 1.$$

2. ***Security:*** *For all PPT adversaries* $\mathcal{A}$*, there exists a negligible function* $\varepsilon_\Pi(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$ *we have*

$$\Pr\left[\mathsf{Sig\text{-}forge}_{\Pi,\mathcal{A}}(\lambda) = 1\right] \leqslant \varepsilon_\Pi(\lambda),$$

where the experiment Sig-forge *is defined in Figure 7.5.*

For completeness, we also include the classical definition of an error-correcting code.

Definition 7.3.2. *A* coding scheme $C[K, k, q_1, q_2] = (\mathsf{Enc}, \mathsf{Dec})$ *is a pair of encoding and decoding algorithms* $\mathsf{Enc}\colon \Sigma_1^k \to \Sigma_2^K$ *and* $\mathsf{Dec}\colon \Sigma_2^* \to \Sigma_1^k$*, where* $|\Sigma_i| = q_i$*. A code* $C[K, k, q_1, q_2]$ *is a* (ρ, dist) error-correcting code *for* $\rho \in [0,1]$ *and fractional distance* dist *if for all* $x \in \Sigma_1^k$ *and* $y \in \Sigma_2^*$ *such that* $\mathrm{dist}(\mathsf{Enc}(x), y) \leqslant \rho$*, we have that* $\mathsf{Dec}(y) = x$*. Here,* ρ *is the* error rate *of* C*. If* $q_1 = q_2$*, we simply denote this by* $C[K, k, q_1]$*. If* $\mathrm{dist} = \mathsf{HAM}$*, then* C *is a* Hamming code*; if* $\mathrm{dist} = \mathsf{ED}$*, then* C *is an* insertion-deletion code *(InsDel code).*

[3] ↑Other definitions (e.g., [169]) require this condition to hold except with negligible probability over $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$.

Key to our construction is the so-called "SZ-code", which is an insertion-deletion error-correcting code with constant rate and constant error-tolerance.

Lemma 7.3.1 (SZ-code [230])**.** *There exists positive constants $\beta_{\mathsf{sz}} \leqslant 1$ and $\rho_{\mathsf{sz}} > 0$ such that for large enough values of $t \in \mathbb{Z}^+$, there exists a $(\rho_{\mathsf{sz}}, \mathsf{ED})$ code $\mathsf{SZ}(t) = (\mathsf{SZ.Enc}, \mathsf{SZ.Dec})$ where $\mathsf{SZ.Enc}\colon \{0,1\}^t \to \{0,1\}^{(1/\beta_{\mathsf{sz}})\cdot t}$ and $\mathsf{SZ.Dec}\colon \{0,1\}^* \to \{0,1\}^t \cup \{\perp\}$ with the following properties:*

1. $\mathsf{SZ.Enc}$ *and* $\mathsf{SZ.Dec}$ *run in time* $\mathrm{poly}(t)$*; and*
2. *For all $x \in \{0,1\}^t$, every interval of length $2\log(t)$ in $\mathsf{Enc}(x)$ has fractional Hamming weight $\geqslant 2/5$.*

We omit the parameter t when it is clear from context.

7.4 Proof of Theorem 7.1.1

We dedicate this section to showing that $\mathcal{C}_{\mathsf{HAM},\lambda} = \{(\mathsf{Enc}_{\mathsf{H},\lambda}, \mathsf{Dec}_{\mathsf{H},\lambda})\}_{\lambda\in\mathbb{N}}$ satisfies Theorem 7.1.1, where $\mathsf{Enc}_{\mathsf{H},\lambda}$ and $\mathsf{Dec}_{\mathsf{H},\lambda}$ are defined in Figures 7.1 and 7.2, respectively. We recall the theorem below.

Theorem 7.1.1. *Suppose that $\Pi = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ is a digital signature scheme with signatures of length $r := r(\lambda)$. Let $C_{in} = (\mathsf{Enc}_{in}, \mathsf{Dec}_{in})$ be a Hamming code with rate β_{in} and error-tolerance ρ_{in}. Then for every positive polynomial $k(\cdot)$ and positive constant $c < 1/2$, there exists code family $\mathcal{C}_{\mathsf{HAM}} := \{C_{\mathsf{H},\lambda}[n, k(\lambda), 2] = (\mathsf{Enc}_{\mathsf{H},\lambda}, \mathsf{Dec}_{\mathsf{H},\lambda})\}_{\lambda\in\mathbb{N}}$ such that for all $\varepsilon > 0$ the family $\mathcal{C}_{\mathsf{HAM}}$ is a (ℓ, ρ, p, δ)-Hamming crLDC with block length $n = O((1/\beta_{in})\max\{k(1 + \log(k)/r), r\})$, error-tolerance $\rho = c\cdot\rho_{in}$, locality $\ell = O((\log^{1+\varepsilon}(\lambda)/\beta_{in})\cdot(r + \log(k)))$, success probability $p = 1 - \mathsf{negl}(\lambda) > 2/3$, and $\delta = 1/2$, where $k := k(\lambda)$ and $\mathsf{negl}(\cdot)$ is a negligible function.*

Proof. For the block length n, note that $x^{(j)}, \sigma^{(j)} \in \{0,1\}^{r(\lambda)}$, and $j \in \{0,1\}^{\log(d)}$. Furthermore, without loss of generality we assume that $\mathsf{pk} \in \{0,1\}^\lambda$ and $r(\lambda) \geqslant \lambda$. Note that $\log(d) = O(\log(k))$. If rate of C_{in} is β_{in}, then block length of our code is $d\cdot(1/\beta_{in})\cdot(2r(\lambda)+\lambda+\log(d)) = (\lceil k/r(\lambda)\rceil)\cdot(1/\beta_{in})\cdot(2r(\lambda)+\lambda+\log(\lceil k/r(\lambda)\rceil)) = O((1/\beta_{in})\cdot k\cdot(3+\log(k)/r(\lambda))) =$

$O((1/\beta_{in}) \cdot k(1 + \log(k)/r(\lambda)))$ whenever $k \geqslant r(\lambda)$. When $r(\lambda) > k$, then we pad the input message x with $k - r(\lambda)$ number of 0's at the end to get a string of length $r(\lambda)$, which gives a single codeword block of length $(1/\beta_{in}) \cdot (2r(\lambda) + \lambda + 1) = O((1/\beta_{in}) \cdot r(\lambda))$. Thus we have our block length as $O((1/\beta_{in}) \cdot \max\{k \cdot (1 + \log(k)/r(\lambda)), r(\lambda)\})$. For the locality ℓ, we know that any block j has length $(1/\beta_{in}) \cdot (2r(\lambda) + \lambda + \log(d)) = (1/\beta_{in}) \cdot (2r(\lambda) + \lambda + \log(k) - \log(r(\lambda))) = O((1/\beta_{in}) \cdot (r(\lambda) + \log(k)))$. Since we decode $\mu + 1$ blocks, our overall locality is $\ell = O((\mu/\beta_{in}) \cdot (r(\lambda) + \log(k)))$. Moreover, $\mathsf{Dec}_{\mathsf{H},\lambda}$ makes $O((\mu/\beta_{in}) \cdot (r(\lambda) + \log(k)))$ queries to its oracle on any input i, satisfying Item 1 of Definition 7.1.1.

For item Item 2, assume that $\mathsf{Dec}_{\mathsf{H},\lambda}$ is given oracle access to $\widetilde{C} = C$ for some $C = \mathsf{Enc}_{\mathsf{H},\lambda}(x)$ and $x \in \{0,1\}^k$. We then analyze the probability that $\mathsf{Dec}^{C}_{\mathsf{H},\lambda}(i) = x_i$ for any $i \in [k]$. First, since C is a correct codeword, recovery of the public key succeeds with probability 1. That is, for every $\kappa \in [\mu]$, the string $\widetilde{m}^{(j_\kappa)} \leftarrow \mathsf{Dec}_{in}\left(C[(j_\kappa - 1) \cdot \mathsf{bl} + 1, j_\kappa \cdot \mathsf{bl}]\right)$ recovered in Line 5 is equal to $x^{(j_\kappa)} \circ \sigma^{(j_\kappa)} \circ \mathsf{pk} \circ j_\kappa$ (i.e., everything is correct). Here, $\mathsf{bl} = n/d$ is the length of each block $C^{(j)}$ in $C = C^{(1)} \circ \cdots \circ C^{(d)}$. Thus $\mathsf{pk}^* = \mathsf{pk}$ with probability 1. Now fixing $j \in [d]$ to be the block such that bit x_i resides in $x^{(j)}$, by the above discussion we know that $\widetilde{m}^{(j)}$ recovered in Line 10 is correct and is parsed as $x^{(j)} \circ \sigma^{(j)} \circ \mathsf{pk} \circ j$. This along with the fact that $\mathsf{pk}^* = \mathsf{pk}$ implies that Line 12 is true with probability 0 (i.e., $\mathsf{Verify}_{\mathsf{pk}^*}(x^{(j)} \circ j, \sigma^{(j)}) = 1$ with probability 1). This implies that $\mathsf{Dec}^{C}_{\mathsf{H},\lambda}(i) = x_i$ with probability 1, as desired.

For error-tolerance, let $\rho_{in} \in (0,1)$ be the error-tolerance of C_{in}. Intuitively, we want to set our final error-tolerance ρ such that for any ρ-fraction of corruptions, less than half of the d blocks contain more than ρ_{in} faction of errors each. Equivalently, more than half of the d blocks contain at most ρ_{in} fraction of errors. Let K be the bitlength of each block. To corrupt a single block $j \in [d]$ such that decoding is incorrect, the block j must contain at least $\rho_{in} \cdot K + 1$ errors (bitflips). Let $\widetilde{J} \subset [d]$ be the set of indices such that block $j \in \widetilde{J}$ has at least $\rho_{in} \cdot K + 1$ Hamming errors. Then we have

$$
\begin{aligned}
|\widetilde{J}| \cdot (\rho_{in} \cdot K + 1) &\leqslant \rho \cdot K \cdot d \\
|\widetilde{J}| &\leqslant \frac{\rho \cdot K \cdot d}{(\rho_{in} \cdot K + 1)} < \frac{\rho \cdot K \cdot d}{\rho_{in} \cdot K} \\
|\widetilde{J}| &< \frac{\rho}{\rho_{in}} \cdot d.
\end{aligned}
$$

We want $|\widetilde{J}| < (1/2) \cdot d$ to ensure that more than half of the blocks contain at most $\rho_{in} \cdot K$ errors. Therefore we have

$$\frac{\rho}{\rho_{in}} \cdot d < \frac{1}{2} \cdot d \implies \rho < \frac{\rho_{in}}{2}.$$

Thus we set $\rho = c \cdot \rho_{in} = \Theta(\rho_{in})$ for any positive constant $c < 1/2$.

Next we turn to analyzing the success probability p and Item 3 of Definition 7.1.1. For the predicate Fool, fix message x and let $C = \mathsf{Enc}_{\mathsf{H},\lambda}(x)$. Let $\widetilde{C} \in \{0,1\}^n$ such that $\mathsf{HAM}(C, \widetilde{C}) \leqslant \rho$. We want to show that for all PPT $\mathcal{A}$ there exists a negligible function ε_{F} such that for all λ and $x \in \{0,1\}^k$ we have $\Pr[\mathsf{Fool}(\mathcal{A}(C), \rho, p, x, C, \lambda) = 1] \leqslant \varepsilon_{\mathsf{F}}(\lambda)$ for $C = \mathsf{Enc}_{\mathsf{H},\lambda}(x)$. Equivalently, we want to show that

$$\Pr\left[\exists i \in [k] \colon \Pr\left[\mathsf{Dec}^{\widetilde{C}}_{\mathsf{H},\lambda}(i) \in \{x_i, \perp\}\right] < p\right] \leqslant \varepsilon_{\mathsf{F}}(\lambda),$$

where $\mathsf{HAM}(C, \widetilde{C}) \leqslant \rho$. Restated again, we want to show that

$$\Pr\left[\forall i \in [k] \colon \Pr\left[\mathsf{Dec}^{\widetilde{C}}_{\mathsf{H},\lambda}(i) \in \{x_i, \perp\}\right] \geqslant p\right] \geqslant 1 - \varepsilon_{\mathsf{F}}(\lambda).$$

We begin by analyzing the probability that $\mathsf{Dec}^{\widetilde{C}}_{\mathsf{H},\lambda}(i) \in \{x_i, \perp\}$, and let E_i denote this event, and also let Forge_i denote the event that $\mathcal{A}$ produces a signature forgery $(\widetilde{x}^{(j)} \circ j, \widetilde{\sigma}^{(j)})$, where $j \in [d]$ such that $(j-1) \cdot r(\lambda) < i \leqslant j \cdot r(\lambda)$ and $\widetilde{x}^{(j)}$ and $\widetilde{\sigma}^{(j)}$ are recovered in Line 10 of Figure 7.2. Then we have that

$$\Pr[E_i] = \Pr[E_i \mid \overline{\mathsf{Forge}_i}].$$

Note that the decoder can never output $\perp$ and x_i simultaneously. Letting $E_i(x)$ be the event that $\mathsf{Dec}^{\widetilde{C}}_{\mathsf{H},\lambda}(i) = x$ for symbol x, we have that

$$\Pr[E_i \mid \overline{\mathsf{Forge}_i}] = \Pr[E_i(x_i) \mid \overline{\mathsf{Forge}_i}] + \Pr[E_i(\perp) \mid \overline{\mathsf{Forge}_i}].$$

We analyze the first probability $\Pr[E_i(x_i) \mid \overline{\mathsf{Forge}_i}]$. For this case, since we assume that $(\widetilde{x}^{(j)} \circ j, \widetilde{\sigma}^{(j)})$ is not a forgery, it must be the case that (1) $\widetilde{x}^{(j)} = x^{(j)}$ and $\widetilde{\sigma}^{(j)} = \sigma^{(j)}$; and (2) $\mathsf{Verify}_{\mathsf{pk}^*}(\widetilde{x}^{(j)} \circ j, \widetilde{\sigma}^{(j)}) = 1$. Now the verification in this case only succeeds if $\mathsf{pk}^* = \mathsf{pk}$. This implies

$$\Pr[E_i(x_i) \mid \overline{\mathsf{Forge}_i}] = \Pr[\mathsf{pk}^* = \mathsf{pk}].$$

Next we analyze the probability $\Pr[E_i(\perp) \mid \overline{\mathsf{Forge}_i}]$. Note that $\mathsf{Dec}_{\mathsf{H},\lambda}$ only outputs $\perp$ if the verification procedure of Line 12 fails. We can then lower bound this probability as

$$\Pr[E_i(\perp) \mid \overline{\mathsf{Forge}_i}] \geqslant$$
$$\Pr[\mathsf{Verify}_{\mathsf{pk}^*}(\widetilde{x}^{(j)} \circ j, \widetilde{\sigma}^{(j)}) = 0 \mid \overline{\mathsf{Forge}_i} \ \wedge \ (\widetilde{x}^{(j)} \neq x^{(j)} \ \vee \ \widetilde{\sigma}^{(j)} \neq \sigma^{(j)}) \ \wedge \ \mathsf{pk}^* = \mathsf{pk}] \cdot \Pr[\mathsf{pk}^* = \mathsf{pk}].$$

Since we assume $\overline{\mathsf{Forge}_i}$ is true, we know that $\Pr[\mathsf{Verify}_{\mathsf{pk}^*}(\widetilde{x}^{(j)} \circ j, \widetilde{\sigma}^{(j)}) = 0 \mid \overline{\mathsf{Forge}_i} \ \wedge \ (\widetilde{x}^{(j)} \neq x^{(j)} \ \vee \ \widetilde{\sigma}^{(j)} \neq \sigma^{(j)}) \ \wedge \ \mathsf{pk}^* = \mathsf{pk}] = 1$. This implies that

$$\Pr[E_i(\perp) \mid \overline{\mathsf{Forge}_i}] \geqslant \Pr[\mathsf{pk}^* = \mathsf{pk}],$$

which in turn implies

$$\Pr[E_i \mid \overline{\mathsf{Forge}_i}] \geqslant 2 \cdot \Pr[\mathsf{pk}^* = \mathsf{pk}] \geqslant \Pr[\mathsf{pk}^* = \mathsf{pk}].$$

We turn to analyzing the probability that $\mathsf{pk}^* = \mathsf{pk}$. For completeness, let pk be the public key sampled by $\mathsf{Enc}_{\mathsf{H},\lambda}(x)$ to generate y. Then by our parameter choice $c \in (0, 1/2)$, at least $(1-c) \cdot d > (1/2)d$ blocks of y' contain at most ρ_{in}-fraction of Hamming errors. Let $\mathsf{bl} = n/d$ denote the length of any codeword block and let $\mathcal{J}_{\mathsf{Good}}$ denote the set of blocks with at most ρ_{in}-fraction of Hamming errors. This implies that $x^{(j)} \circ \sigma^{(j)} \circ \mathsf{pk} \circ j = \mathsf{Dec}_{in}(\widetilde{C}[(j-1) \cdot \mathsf{bl}+1, j \cdot \mathsf{bl}])$

for any $j \in \mathcal{J}_{\mathsf{Good}}$; i.e., it is a correct decoding since $\widetilde{C}^{(j)} = \widetilde{C}[(j-1)\cdot \mathsf{bl}+1, j\cdot \mathsf{bl}]$ is within the unique decoding radius of C_{in}. Define random variable X_κ for $\kappa \in [\mu]$ as

$$X_\kappa := \begin{cases} 1 & x^{(j)} \circ \sigma^{(j)} \circ \mathsf{pk} \circ j = \mathsf{Dec}_{in}(\widetilde{C}[(j-1)\cdot \mathsf{bl}+1, j\cdot \mathsf{bl}]) \\ 0 & \text{otherwise} \end{cases}.$$

We know that

$$\begin{aligned} \Pr[X_\kappa = 1] &= \Pr[x^{(j)} \circ \sigma^{(j)} \circ \mathsf{pk} \circ j = \mathsf{Dec}_{in}(\widetilde{C}[(j-1)\cdot \mathsf{bl}+1, j\cdot \mathsf{bl}])] \\ &= \Pr[j_\kappa \in \mathcal{J}_{\mathsf{Good}}] \\ &\geqslant (1-c) > 1/2. \end{aligned}$$

Let $q = (1-c) > 1/2$. Then by a Chernoff bound we have that

$$\Pr\left[\sum_{\kappa\in[\mu]} X_\kappa > \frac{\mu}{2}\right] \geqslant 1 - \exp(-\mu\cdot(q-1/2)^2/(2q)).$$

This implies that with probability at least $p := 1-\exp(-\mu\cdot(q-1/2)^2/(2q))$, we have $\mathsf{pk}^* = \mathsf{pk}$. Thus we have

$$\Pr[E_i \mid \overline{\mathsf{Forge}_i}] \geqslant p.$$

Throughout the above analysis, we only assumed that no forgery occurred. For any arbitrary i, the probability that $\overline{\mathsf{Forge}_i}$ occurs is at least $1-\varepsilon_\Pi(\lambda)$, where $\varepsilon_\Pi(\cdot)$ is a negligible function that depends on the security of the digital signature scheme Π. Thus by union bound, we have that

$$\Pr[\exists i \in [k]\colon \Pr[\mathsf{Dec}^{\widetilde{C}}_{\mathsf{H},\lambda}(i) \in \{x_i, \perp\}] < p] \geqslant k\cdot\varepsilon_\Pi(\lambda).$$

Setting $\varepsilon_{\mathsf{F}}(\lambda) := k\cdot\varepsilon_\Pi(\lambda)$, we have that $\varepsilon_{\mathsf{F}}(\lambda)$ is negligible in λ since $k(\lambda)$ is a polynomial.

Finally, we turn to analyzing the parameter δ and Item 4 of Definition 7.1.1. By our choice of $\rho = c\cdot\rho_{in}$ for positive constant $c < 1/2$, we know that for *any* $\widetilde{C} \in \{0,1\}^n$ such that

$\mathsf{HAM}(\widetilde{C}, C) \leqslant \rho$, at least $(1-c)\cdot d$ blocks of $\widetilde{C}$ contain at most ρ_{in}-fraction of Hamming errors. Again letting $\mathcal{J}_{\mathsf{Good}} \subset [d]$ denote the indices of these blocks, we have $|\mathcal{J}_{\mathsf{Good}}| > d/2$. For $\mathsf{bl} = n/d$, this again implies that $x^{(j)} \circ \sigma^{(j)} \circ \mathsf{pk} \circ j = \mathsf{Dec}_{in}(\widetilde{C}[(j-1)\cdot\mathsf{bl}+1, j\cdot\mathsf{bl}])$ for any $j \in \mathcal{J}_{\mathsf{Good}}$. This in turn implies that for any $j \in \mathcal{J}_{\mathsf{Good}}$, we have that $\Pr\left[\mathsf{Dec}^{\widetilde{C}}_{\mathsf{H},\lambda}(i) = x_i | \mathsf{pk}^* = \mathsf{pk}\right] = 1$ whenever $(j-1)\cdot r(\lambda) < i \leqslant j \cdot r(\lambda)$. Thus for any $j \in \mathcal{J}_{\mathsf{Good}}$ and $i \in [k]$ such that $(j-1)\cdot r(\lambda) < i \leqslant j \cdot r(\lambda)$, we have that

$$\Pr\left[\mathsf{Dec}^{\widetilde{C}}_{\mathsf{H},\lambda}(i) = x_i\right] = \Pr\left[\mathsf{pk}^* = \mathsf{pk}\right] \geqslant 1 - \exp(-\mu \cdot (1/2-c)^2 \big/ 2(1-c)).$$

By appropriately choosing $\mu(\lambda) := \Theta(\log^{1+\varepsilon}(\lambda))$ for $\varepsilon > 0$, we can ensure that $1 - \exp(-\mu \cdot (1/2-c)^2 \big/ 2(1-c)) = 1 - \mathsf{negl}(\lambda) > 2/3$; this also gives our claimed locality ℓ. Finally, we note that the set $\mathcal{I}_{\mathsf{Good}} := \{i \in [k] \colon j \in \mathcal{J}_{\mathsf{Good}} \ \wedge \ (j-1)\cdot r(\lambda) < i \leqslant j \cdot r(\lambda)\}$ has size $|\mathcal{I}_{\mathsf{Good}}| > k/2$. Thus we can set $\delta = 1/2$.

Here we have shown that for *any* $\widetilde{C} \in \{0,1\}^n$ such that $\mathsf{HAM}(\widetilde{C}, C) \leqslant \rho$, there exists a set $\mathsf{Good}(\widetilde{C}) := \mathcal{I}_{\mathsf{Good}}$ such that $|\mathcal{I}_{\mathsf{Good}}| \geqslant \delta \cdot k$. This is for any $\widetilde{C}$, and in particular, any corrupt codeword that a PPT adversary could produce. Thus we have that for any PPT adversary $\mathcal{A}$, any $x \in \{0,1\}^k$, and $C = \mathsf{Enc}_{\mathsf{H},\lambda}(x)$:

$$\Pr\left[\mathsf{Limit}(\mathcal{A}(C), \rho, \delta, x, C, \lambda) = 1\right] = 0.$$

□

7.5 Proof of Theorem 7.1.3

We dedicate this section to proving Theorem 7.1.3. We recall the theorem here.

Theorem 7.1.3. *Suppose that* $\Pi = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ *is a digital signature scheme with signatures of length* $r := r(\lambda)$. *For every positive polynomial* $k(\cdot)$ *and positive constant* $\rho^* < 1/3$, *there exists a code family* $\mathcal{C}_{\mathsf{Ins}} := \{C_\lambda[n, k(\lambda), 2] = (\mathsf{Enc}_{\mathsf{I},\lambda}, \mathsf{Dec}_{\mathsf{I},\lambda})\}_{\lambda\in\mathbb{N}}$ *such that for all* $\varepsilon > 0$, *the family* $\mathcal{C}_{\mathsf{Ins}}$ *is a* (ℓ, ρ, p, δ)*-InsDel* crLDC *with block length* $n = O(\max\{k(1 + \log(k)/r), r\})$, *error-tolerance* $\rho = \Theta(1)$, *locality* $\ell = O(\log^{3+\varepsilon}(\lambda) \cdot (r + \log(k)))$, *success*

probability $p = 1 - \rho^* - \mathsf{negl}(\lambda) > 2/3$*, and* $\delta = 1 - \Theta(\rho)$*, where* $k := k(\lambda)$ *and* $\mathsf{negl}(\cdot)$ *is a negligible function.*

Proof. To begin fix $x \in \{0,1\}^k$ and let $C = \mathsf{Enc}_{\mathsf{I},\lambda}(x)$. In the definition of Enc_λ, we know that $C = C^{(1)} \circ \cdots \circ C^{(d)}$, where $d = \lceil k/r(\lambda) \rceil$. First assume that $k \geqslant r(\lambda)$. For each $C^{(j)}$, note that it is the SZ encoding of $x^{(j)} \circ \sigma^{(j)} \circ \mathsf{pk} \circ j$, appended at the front and back with zero-buffers. Note that the bit-length of j is $\log(d) = \log(k) - \log(r(\lambda))$. For simplicity, assume that index $j \in \{0,1\}^{\log(k)}$ (we can pad to length $\log(k)$ otherwise). Then we have $\tau := |x^{(j)} \circ \sigma^{(j)} \circ \mathsf{pk} \circ j| = 3 \cdot r(\lambda) + \log(k)$. This gives us that $|c^{(j)}| = (1/\beta_{\mathsf{sz}}) \cdot \tau$, and that $|C^{(j)}| = 2\alpha\tau + (1/\beta_{\mathsf{sz}}) \cdot \tau = (2\alpha + (1/\beta_{\mathsf{sz}}))\tau$. Finally, this gives

$$\begin{aligned} n = |C| &= d \cdot (2\alpha + (1/\beta_{\mathsf{sz}})) \cdot \tau \\ &= \left(\frac{k}{r(\lambda)}\right) \cdot (2\alpha + (1/\beta_{\mathsf{sz}})) \cdot (3 \cdot r(\lambda) + \log(k)) \\ &= O\left(k \cdot \left(1 + \frac{\log(k)}{r(\lambda)}\right)\right), \end{aligned}$$

where the last equality holds since $\alpha, \beta_{\mathsf{sz}}$ are constants. Note here that k is sufficiently large whenever $\mathsf{SZ}(t)$ exists for all $t \geqslant \log(k)$. Now whenever $r(\lambda) > k$, we have that $d = 1$ and we simply sign and encode a single block of length τ, which yields a single block of length $(2\alpha + (1/\beta_{\mathsf{sz}}))\tau = (2\alpha + (1/\beta_{\mathsf{sz}})) \cdot (3r(\lambda) + \log(k)) = \Theta(r(\lambda))$ since $r(\lambda) > k$ and $\alpha, \beta_{\mathsf{sz}}$ are constants. Thus we have established $n = O(\max\{k(1 + \log(k)/r(\lambda)), r(\lambda)\})$.

For the remainder of this section, let $\beta := (2\alpha + (1/\beta_{\mathsf{sz}}))$. Next let $\widetilde{C} \in \{0,1\}^{n'}$ be some string such that $\mathsf{ED}(C, \widetilde{C}) \leqslant \rho$. We introduce some preliminary definitions and lemmas before continuing with the proof of Theorem 7.1.3. These definitions and lemmas come from the results of Block et al. [48]. First we define the notion of a *block decomposition*.

Definition 7.5.1 (Block Decomposition [48])**.** *A* block decomposition *of a (corrupt) codeword* $\widetilde{C} \in \{0,1\}^{n'}$ *is a non-decreasing map* $\phi\colon [n'] \to [d]$ *for* $n', d \in \mathbb{N}$.

For any block decomposition ϕ, since ϕ is a non-decreasing map we have that $\phi^{-1}(j)$ for any $j \in [d]$ is an interval. That is, $\phi^{-1}(j) = \{l_j, l_j + 1, \ldots, r_j\}$ for integers $l_j, r_j \in [n']$ and $l_j \leqslant r_j$. Thus ϕ induces a partition of $[n']$ into d intervals of the form $\{\phi^{-1}(j)\colon j \in [d]\}$.

Recall that $C = \mathsf{Enc}_\lambda(x)$ is of the form $C^{(1)} \circ \cdots \circ C^{(d)}$. Then the following holds.

Lemma 7.5.1 ([48]). *There exists a block decomposition $\phi_0 \colon [n'] \to [d]$ such that*

$$\sum_{j \in [d]} \mathsf{ED}\left(\widetilde{C}[\phi_0^{-1}(j)], C^{(j)}\right) \leqslant \rho \,. \tag{7.6}$$

Intuitively, Lemma 7.5.1 says there exists a block decomposition such that the total edit distance between C' and C' is exactly given by the sum of edit distances between the (possibly corrupt) blocks $\widetilde{C}[\phi_0^{-1}(j)]$ and blocks $C^{(j)}$.

Next we define the notion of a γ-good block.

Definition 7.5.2 (γ-good block [48]). *For $\gamma \in (0,1)$ and $j \in [d]$, we say that block j is γ-good with respect to a block decomposition ϕ if $\mathsf{ED}(\widetilde{C}[\phi^{-1}(j)], C^{(j)}) \leqslant \gamma$. Otherwise, we say that block j is γ-bad.*

With respect to block decomposition ϕ_0, the number of γ-bad blocks is bounded, and the length of the intervals $\phi_0^{-1}(j)$ is bounded for every γ-good block j.

Lemma 7.5.2 ([48]). *Let α be the constant given by Lemma 7.5.3, let β_{sz} be the constant given by Lemma 7.3.1, and let $\beta = (2\alpha + (1/\beta_{\mathsf{sz}}))$. Then the block decomposition ϕ_0 satisfies the following properties.*

1. *The total fraction of γ-bad blocks in $\widetilde{C}$ is at most $2 \cdot \beta \cdot \rho/(\gamma \cdot \alpha)$.*
2. *For any γ-good block j, we have that $(\beta - \alpha\gamma) \cdot \tau \leqslant |\phi_0^{-1}(j)| \leqslant (\beta + \alpha\gamma) \cdot \tau$, where $\tau = |x^{(j)} \circ \sigma^{(j)} \circ \mathsf{pk} \circ j|$.*

Given the notion of γ-good, we can now formally introduce the algorithms NBS and $\mathsf{BlockDecode}$ along with their guarantees.

Lemma 7.5.3 (Noisy-Binary Search [48]). *Let ρ_{sz} be the constant given by Lemma 7.3.1. There exists constant $\alpha = \Omega(\rho_{\mathsf{sz}})$ and a randomized oracle algorithm NBS with the following property. Let $\rho^* \in (0, 1/2)$ be a fixed constant, let t be sufficiently large, d be a parameter, let $b = (b^{(1)}, \ldots, b^{(d)}) \in \{0,1\}^t$ be any string where $b^{(i)} \in \{0,1\}^{t/d}$ for all $i \in [d]$, and let $c = (c^{(1)}, \ldots, c^{(d)})$ for $c^{(i)} = 0^{\alpha(t/d)} \circ \mathsf{SZ.Enc}(b^{(i)} \circ i) \circ 0^{\alpha(t/d)}$ for all $i \in [d]$. Then there exists*

a negligible function $\vartheta(\cdot)$ such that for any $c' \in \{0,1\}^{n'}$ satisfying $\mathsf{ED}(c,c') \leqslant \rho = \Theta(\rho^* \cdot \rho_{\mathsf{sz}})$, we have that

$$\Pr\left[\Pr_{j \xleftarrow{\$} [d]}\left[\mathsf{NBS}^{c'}(j) \neq b^{(j)} \mid j \text{ is } \gamma\text{-good}\right] \geqslant \rho^*\right] \leqslant \vartheta(n'), \tag{7.7}$$

where the probability is taken over the random coins of NBS*. Furthermore, the algorithm* NBS *makes $O(\log^3(n') \cdot (t/d + \log(d)))$ oracle queries for any input $j \in [d]$, and if $c = c'$ then the above probability is equal to 0.*

Lemma 7.5.4 ([48])**.** *Let ρ_{sz} be the constant given by Lemma 7.3.1. There exists constant $\alpha = \Omega(\rho_{\mathsf{sz}})$ and randomized oracle algorithm* BlockDecode *with the following properties. Let $\rho^* \in (0, 1/2)$ be a fixed constant, let t be sufficiently large, let d be a parameter, let $b = (b^{(1)}, \ldots, b^{(d)}) \in \{0,1\}^t$ be any string where $b^{(i)} \in \{0,1\}^{t/d}$ for all $i \in [d]$, and let $c = (c^{(1)}, \ldots, c^{(d)})$ for $c^{(i)} = 0^{\alpha(t/d)} \circ \mathsf{SZ.Enc}(b^{(i)} \circ i) \circ 0^{\alpha(t/d)}$ for all $i \in [d]$. Then for any $c' \in \{0,1\}^{n'}$ satisfying $\mathsf{ED}(c,c') \leqslant \rho = \Theta(\rho^* \cdot \rho_{\mathsf{sz}})$, we have that*

1. *For any γ-good block $j \in [d]$,*

$$\Pr_{i \in \phi_0^{-1}(j)}\left[\mathsf{BlockDecode}^{c'}(i) \neq b^{(j)}\right] \leqslant \gamma\,. \tag{7.8}$$

   Furthermore, if $c' = c$, then the probability above is equal to zero.

2. BlockDecode *has query complexity $O(t/d + \log(d))$.*

We now have the necessary components to prove Theorem 7.1.3. We begin by first showing Item 2 of Definition 7.1.1. Let $x \in \{0,1\}^k$, $C = \mathsf{Enc}_{\mathsf{I},\lambda}(x)$, and let $(\mathsf{pk}, \mathsf{sk})$ be the public and private key pair sampled by $\mathsf{Enc}_{\mathsf{I},\lambda}$ during the encoding of x as C. Suppose that $\widetilde{C} = C$ and let $i \in [k]$ be any index. We want to argue that $\Pr\left[\mathsf{Dec}_{\mathsf{I},\lambda}^{\widetilde{C}}(i) = x_i\right] = 1$. To see this, first observe that for $(j-1)r(\lambda) < i \leqslant jr(\lambda)$, we have

$$\Pr\left[\mathsf{Dec}_{\mathsf{I},\lambda}^{\widetilde{C}}(i) = x_i\right] = \Pr\left[\mathsf{pk}^* = \mathsf{pk}\right] \cdot \Pr\left[\widetilde{x}^{(j)} = x^{(j)}\right] \cdot \Pr\left[\widetilde{\sigma}^{(j)} = \sigma^{(j)}\right], \tag{7.9}$$

where $x^{(j)}$ is the j-th block of x (Line 3), $\sigma^{(j)}$ is computed as in Equation (7.3) of Figure 7.3, pk^* is computed as in Line 11 of Figure 7.4, and $\widetilde{x}^{(j)}, \widetilde{\sigma}^{(j)}$ are obtained from Line 16 of Figure 7.4. First notice that since $\widetilde{C} = C$, every block $j \in [d]$ of $\widetilde{C}$ is 0-good with respect to C. By Lemma 7.5.4, for every $j \in [d]$ we have that

$$\Pr_{i \in \phi_0(j)} \left[\mathsf{BlockDecode}^{\widetilde{C}}(i) = C^{(j)} \right] = 1.$$

This implies that for every $\kappa \in [\mu]$, it holds that $\mathsf{pk}^{(i_\kappa)} = \mathsf{pk}$ (Line 10) with probability 1, which implies that $\mathsf{pk}^* = \mathsf{pk}$ with probability 1. Next, by Lemma 7.5.3 because $\widetilde{C} = C$, we have that for $\widetilde{m}^{(j)} \leftarrow \mathsf{NBS}^{\widetilde{C}}(j)$, it holds that $\widetilde{m}^{(j)} = (x^{(j)} \circ \sigma^{(j)} \circ \mathsf{pk} \circ j)$ with probability 1. This implies that $\mathsf{Verify}_{\mathsf{pk}^*}(x^{(j)} \circ j, \sigma^{(j)}) = 1$ with probability 1, and thus $\mathsf{Dec}^{\widetilde{C}}_{\mathsf{I},\lambda}(i) = x_i$ with probability 1 as desired.

We now work towards proving Items 3 and 4 of Definition 7.1.1. Let $\mathcal{A}$ be a PPT adversary and let $\widetilde{C} = \mathcal{A}(C)$ such that $\widetilde{C} \in \{0,1\}^{n'}$ for some n' and $\mathsf{ED}(\widetilde{C}, C) \leqslant \rho$. We begin with Item 3. Let $D_i := \mathsf{Dec}^{\widetilde{C}}_{\mathsf{I},\lambda}(i)$ denote the random variable of running the decoder with input i and oracle $\widetilde{C}$. Our goal is to show that

$$\Pr\left[\exists i \in [k] \colon \Pr\left[D_i \in \{x_i, \perp\}\right] < p\right] \leqslant \varepsilon_{\mathsf{F}}(\lambda),$$

where $\varepsilon_{\mathsf{F}}(\lambda)$ is some negligible function. Equivalently stated:

$$\Pr\left[\forall i \in [k] \colon \Pr\left[D_i \in \{x_i, \perp\}\right] \geqslant p\right] \geqslant 1 - \varepsilon_{\mathsf{F}}(\lambda).$$

We directly analyze $\Pr\left[D_i \in \{x_i, \perp\}\right]$. The analysis here is almost identical to the analysis of Theorem 7.1.1, except now we must take into consideration the algorithms $\mathsf{BlockDecode}$ and NBS. Let Forge_i denote the event that $\mathcal{A}$ produces a signature forgery $(\widetilde{x}^{(j)} \circ j, \widetilde{\sigma}^{(j)})$ for $j \in [d]$ satisfying $(j-1) \cdot r(\lambda) < i \leqslant j \cdot r(\lambda)$ and $\widetilde{x}^{(j)}$ and $\widetilde{\sigma}^{(j)}$ are recovered in Line 16 of Figure 7.4. Then we have that

$$\Pr\left[D_i \in \{x_i, \perp\}\right] = \Pr\left[D_i \in \{x_i, \perp\} \mid \overline{\mathsf{Forge}_i}\right].$$

Since the decoder can never output $\perp$ and x_i simultaneously, we have that

$$\Pr\left[D_i \in \{x_i, \perp\} \mid \overline{\mathsf{Forge}_i}\right] = \Pr\left[D_i = x_i \mid \overline{\mathsf{Forge}_i}\right] + \Pr\left[D_i = \perp \mid \overline{\mathsf{Forge}_i}\right].$$

We analyze $\Pr\left[D_i = x_i \mid \overline{\mathsf{Forge}_i}\right]$. First notice that in this case, we have (1) $\mathsf{pk}^* \neq \perp$ and $\widetilde{m}^{(j)} \neq \perp$; and (2) $\mathsf{Verify}_{\mathsf{pk}^*}(\widetilde{x}^{(j)} \circ j, \widetilde{\sigma}^{(j)}) = 1$. Since we assume that $(\widetilde{x}^{(j)} \circ j, \widetilde{\sigma}^{(j)})$ is not a forgery, given the above it must be the case that (1) $\widetilde{x}^{(j)} = x^{(j)}$ and $\widetilde{\sigma}^{(j)} = \sigma^{(j)}$; and (2) $\mathsf{pk}^* = \mathsf{pk}$. Noting that NBS and $\mathsf{BlockDecode}$ are run independently, this implies that

$$\Pr\left[D_i = x_i \mid \overline{\mathsf{Forge}_i}\right] = \Pr\left[\mathsf{pk}^* = \mathsf{pk} \ \wedge \ \widetilde{m}^{(j)} \neq \perp\right] = \Pr\left[\mathsf{pk}^* = \mathsf{pk}\right] \cdot \Pr\left[\widetilde{m}^{(j)} \neq \perp\right].$$

Next we analyze $\Pr\left[D_i = \perp \mid \overline{\mathsf{Forge}_i}\right]$. The decoder outputs bot if $\mathsf{pk}^* = \perp$ or $\widetilde{m}^{(j)} = \perp$ or $\mathsf{Verify}_{\mathsf{pk}^*}(\widetilde{x}^{(j)} \circ j, \widetilde{\sigma}^{(j)}) = 0$, noting that the final verification is only checked conditioned on $\mathsf{pk}^* \neq \perp$ and $\widetilde{m}^{(j)} \neq \perp$. Thus we have

$$\begin{aligned}
&\Pr\left[D_i = \perp \mid \overline{\mathsf{Forge}_i}\right] = \Pr\left[\mathsf{pk}^* = \perp\right] + \Pr\left[\widetilde{m}^{(j)} = \perp\right] - \Pr\left[\mathsf{pk}^* = \perp \ \wedge \ \widetilde{m}^{(j)} = \perp\right] \\
&+ \Pr\left[\mathsf{Verify}_{\mathsf{pk}^*}(\widetilde{x}^{(j)} \circ j, \widetilde{\sigma}^{(j)}) = 0 \mid \mathsf{pk}^* \neq \perp \ \wedge \ \widetilde{m}^{(j)} \neq \perp \ \wedge \ \overline{\mathsf{Forge}_i}\right] \cdot \Pr\left[\mathsf{pk}^* \neq \perp \ \wedge \ \widetilde{m}^{(j)} \neq \perp\right].
\end{aligned}$$

Since we are assuming no forgery has occurred, the last probability can be lower bounded as

$$\begin{aligned}
&\Pr\left[\mathsf{Verify}_{\mathsf{pk}^*}(\widetilde{x}^{(j)} \circ j, \widetilde{\sigma}^{(j)}) = 0 \mid \mathsf{pk}^* \neq \perp \ \wedge \ \widetilde{m}^{(j)} \neq \perp \ \wedge \ \overline{\mathsf{Forge}_i}\right] \cdot \Pr\left[\mathsf{pk}^* \neq \perp \ \wedge \ \widetilde{m}^{(j)} \neq \perp\right] \\
&\geqslant \Pr\left[\mathsf{Verify}_{\mathsf{pk}^*}(\widetilde{x}^{(j)} \circ j, \widetilde{\sigma}^{(j)}) = 0 \mid \overline{\mathsf{Forge}_i} \ \wedge \ (\widetilde{x}^{(j)} \neq x^{(j)} \ \vee \ \widetilde{\sigma}^{(j)} \neq \sigma^{(j)}) \ \wedge \ \mathsf{pk}^* = \mathsf{pk}\right] \\
&\qquad \cdot \Pr\left[\widetilde{m}^{(j)} \neq \perp \ \wedge \ \mathsf{pk}^* = \mathsf{pk}\right] \\
&= \Pr\left[\widetilde{m}^{(j)} \neq \perp \ \wedge \ \mathsf{pk}^* = \mathsf{pk}\right],
\end{aligned}$$

where for the lower bound we ignore the case that $\mathsf{pk}^* \neq \mathsf{pk} \ \wedge \ \mathsf{pk}^* \neq \perp$. Thus we have that

$$\begin{aligned}
\Pr\left[D_i \in \{x_i, \perp\} \mid \overline{\mathsf{Forge}_i}\right] \geqslant \Pr\left[\mathsf{pk}^* = \perp\right] + \Pr\left[\widetilde{m}^{(j)} = \perp\right] - \Pr\left[\mathsf{pk}^* = \perp\right] \cdot \Pr\left[\widetilde{m}^{(j)} = \perp\right] \\
+ \Pr\left[\widetilde{m}^{(j)} \neq \perp\right] \cdot \Pr\left[\mathsf{pk}^* = \mathsf{pk}\right]
\end{aligned}$$

where the inequality holds since NBS and $\mathsf{BlockDecode}$ are run independently of each other.

With the above lower bound established, we turn to analyzing the probability that $\mathsf{pk}^* = \mathsf{pk}$. Let pk be the public key sampled by $\mathsf{Enc}_{\mathsf{I},\lambda}(x)$ to generate codeword C. Our recovery of pk is performed by sampling μ indices of $\widetilde{C}$ independently and uniformly at random, running $\mathsf{BlockDecode}$ on these indices, and taking the majority of the public keys we recover. Intuitively, we want to recover pk^* with high probability via a Chernoff bound. Define random variable X_κ for $\kappa \in [\mu]$ as

$$X_\kappa := \begin{cases} 1 & x^{(j)} \circ \sigma^{(j)} \circ \mathsf{pk} \circ j = \mathsf{BlockDecode}(i_\kappa) \text{ for some } j \in [d] \\ 0 & \text{otherwise} \end{cases}.$$

Thus we need to ensure that $\Pr[X_\kappa = 1] > 1/2$. By Lemma 7.5.4, we know that as long as the index i_κ lies within the bounds of some γ-good block j, then $X_\kappa = 1$ with probability at least $1 - \gamma$. Let $\mathcal{J}_{\mathsf{Good}} \subset [d]$ be the indices of the γ-good blocks in $\widetilde{C}$. Then we have

$$\Pr[X_\kappa = 1] \geqslant \Pr[i_\kappa \in \phi_0^{-1}(j) \mid j \in \mathcal{J}_{\mathsf{Good}}] \cdot (1 - \gamma)$$

By Lemma 7.5.2, we know that there are at least $1 - 2 \cdot \beta \cdot \rho/(\gamma \cdot \alpha)$-fraction of blocks which are γ-good, which implies $|\mathcal{J}_{\mathsf{Good}}| \geqslant d \cdot (1 - 2 \cdot \beta \cdot \rho/(\gamma \cdot \alpha))$. Since we want $\Pr[X_\kappa = 1] > 1/2$, by Lemma 7.5.1 we have

$$\begin{aligned} \frac{1}{n} \cdot d \cdot \left(1 - \frac{2 \cdot \beta \cdot \rho}{\gamma \cdot \alpha}\right) \cdot (\beta - \alpha \cdot \gamma) \cdot \tau \cdot (1 - \gamma) &> \frac{1}{2} \\ \frac{(\beta - \alpha \cdot \gamma) \cdot \tau}{\beta \cdot \tau} \cdot \left(1 - \frac{2 \cdot \beta \cdot \rho}{\gamma \cdot \alpha}\right) &> \frac{1}{2 \cdot (1 - \gamma)} \\ \left(1 - \frac{2 \cdot \beta \cdot \rho}{\gamma \cdot \alpha}\right) &> \frac{\beta}{2 \cdot (1 - \gamma) \cdot (\beta - \alpha \cdot \gamma)} \\ \frac{2 \cdot \beta \cdot \rho}{\gamma \cdot \alpha} &< 1 - \frac{\beta}{2 \cdot (1 - \gamma) \cdot (\beta - \alpha \cdot \gamma)} \\ \rho &< \frac{\gamma \cdot \alpha}{2 \cdot \beta} \cdot \left(1 - \frac{\beta}{2 \cdot (1 - \gamma) \cdot (\beta - \alpha \cdot \gamma)}\right) \end{aligned}$$

Borrowing the parameters of Block et al. [48, Proposition 22], we set $\gamma = 1/12$ and $\alpha = 2\gamma\rho_{in}/(\gamma+6)$. Recall also that $\beta = 2\alpha + (1/\beta_{\mathsf{sz}})$. Then we have

$$\begin{aligned}
\alpha &= (2/73)\cdot\rho_{in} \\
\beta &= (4/73)\cdot\rho_{in} + (1/\beta_{\mathsf{sz}}) \\
\frac{\gamma\cdot\alpha}{2\cdot\beta} &= \frac{(1/12)\cdot(2/73)\cdot\rho_{in}}{2\cdot(4/73)\cdot\rho_{in}+R} < 1 \\
\frac{\beta}{2\cdot(1-\gamma)\cdot(\beta-\alpha\cdot\gamma)} &= \frac{(4/73)\cdot\rho_{in}+(1/\beta_{\mathsf{sz}})}{(11/6)\cdot((23/438)\cdot\rho_{in}+(1/\beta_{\mathsf{sz}}))} < 1.
\end{aligned}$$

Given the above parameters, note that since $\rho_{in} < 1$, we have that

$$\frac{\gamma\cdot\alpha}{2\cdot\beta} \geqslant \frac{(2/876)\cdot\rho_{in}}{(8/73)+R}.$$

Thus setting

$$\rho := \frac{(2/876)\cdot\rho_{in}}{(8/73)+R}\cdot\left(1-\frac{(4/73)\cdot\rho_{in}+(1/\beta_{\mathsf{sz}})}{(11/6)\cdot((23/438)\cdot\rho_{in}+(1/\beta_{\mathsf{sz}}))}\right) = \Theta(1)$$

ensure that $\Pr[X_\kappa = 1] > 1/2$. Now let $q := \Pr[X_\kappa = 1] > 1/2$. By a Chernoff bound we have that

$$\Pr\left[\sum_{\kappa\in[\mu]} X_\kappa > \frac{\mu}{2}\right] \geqslant 1-\exp(-\mu\cdot(q-1/2)^2/(2q)).$$

This implies that with probability at least $1-\exp(-\mu\cdot(q-1/2)^2/(2q))$, we have that $\mathsf{pk}^* = \mathsf{pk}$. Note this implies that $\Pr[\mathsf{pk}^* = \perp] \leqslant \exp(-\mu\cdot(q-1/2)^2/(2q))$.

Now given that we recover $\mathsf{pk}^* = \mathsf{pk}$ with probability at least $1-\exp(-\mu\cdot(q-1/2)^2/(2q))$, we analyze the probability that $\widetilde{m}^{(j)} \neq \perp$. Note that by our noisy binary search algorithm in Lemma 7.5.3, if block j is γ-good, then we recover the correct block $x^{(j)}\circ\sigma^{(j)}\circ\mathsf{pk}\circ j$ with probability at least $(1-\rho^*)$, except with probability $\vartheta(n')$, where ρ^* is some constant and ϑ

is a negligible function. Conditioning on the case where we recover any γ-good block with probability at least $(1-\rho^*)$, we have

$$\Pr\left[\widetilde{m}^{(j)} \neq \perp \mid j \text{ is } \gamma\text{-good}\right] \geqslant (1-\rho^*).$$

This implies

$$\Pr\left[\widetilde{m}^{(j)} = \perp \mid j \text{ is } \gamma\text{-good}\right] \leqslant \rho^*.$$

Putting it all together we have that

$$\begin{aligned}\Pr\left[D_i \in \{x_i, \perp\} \mid \overline{\mathsf{Forge}_i}\right] &\geqslant (1-\rho^*)\cdot(1-\exp(-\mu\cdot(q-1/2)^2/(2q))) - \rho^*\cdot\exp(-\mu\cdot(q-1/2)^2/(2q)) \\ &= 1-\rho^*-\exp(-\mu\cdot(q-1/2)^2/(2q)),\end{aligned}$$

where $\rho^* \in (0, 1/2)$ is a fixed constant we are free to choose. In particular, we choose arbitrary positive constant $\rho^* < 1/3$.

Now note that for the above probability, we conditioned on $\overline{\mathsf{Forge}_i}$, as well as our noisy binary search algorithm succeeding. By the security of the digital signature scheme, $\overline{\mathsf{Forge}_i}$ happens with probability at least $1-\varepsilon_\Pi(\lambda)$, where $\varepsilon_\Pi(\lambda)$ is a negligible function for the security of the digital signature scheme. By Lemma 7.5.3, with probability at least $1-\vartheta(n')$ the noisy binary search algorithm gives us the guarantees we need. Finally, by Union bound over $i \in [k]$, we set $\varepsilon_{\mathsf{F}}(\lambda, n) := k\cdot\varepsilon_\Pi(\lambda)\cdot\vartheta(n)$, which is negligible in λ for any fixed k (note when k is fixed then n is a function of λ).

Next we work towards proving Item 4 of Definition 7.1.1. Again let $\widetilde{C} = \mathcal{A}(C)$ for a PPT adversary $\mathcal{A}$. Our goal is to show that there exists a negligible function $\varepsilon_{\mathsf{L}}(\cdot)$ such that for all $\lambda \in \mathbb{N}$ and all $x \in \{0,1\}^k$, we have

$$\Pr\left[\mathsf{Limit}(\mathcal{A}(y), \rho, \delta, x, y) = 1\right] \leqslant \varepsilon_{\mathsf{L}}(\lambda).$$

To do so, we directly analyze the size of the set $\mathsf{Good}(\widetilde{C})$. As before, let D_i be the random variable denoting the output of $\mathsf{Dec}_\lambda^{\widetilde{C}}(i)$. Then we are interested in lower bounding the

probability $\Pr[D_i = x_i]$ for a fixed $i \in [k]$. By definition of Dec_λ, the decoder only outputs a bit if (1) $\mathsf{pk}^* \neq \perp$; (2) $\widetilde{m}^{(j)} \neq \perp$; and (3) $\mathsf{Verify}_{\mathsf{pk}^*}(\widetilde{x}^{(j)} \circ j, \widetilde{\sigma}^{(j)}) = 1$. Thus we have

$$\Pr[D_i = x_i] \tag{7.10}$$

$$= \Pr[\mathsf{pk}^* \neq \perp] \cdot \Pr[\widetilde{m}^{(j)} \neq \perp \mid \mathsf{pk}^* \neq \perp] \cdot \Pr[\mathsf{Verify}_{\mathsf{pk}^*}(\widetilde{x}^{(j)} \circ j, \widetilde{\sigma}^{(j)}) = 1 \mid \mathsf{pk}^* \neq \perp \ \wedge \ \widetilde{m}^{(j)} \neq \perp] \tag{7.11}$$

$$= \Pr[\mathsf{pk}^* \neq \perp] \cdot \Pr[\widetilde{m}^{(j)} \neq \perp] \cdot \Pr[\mathsf{Verify}_{\mathsf{pk}^*}(\widetilde{x}^{(j)} \circ j, \widetilde{\sigma}^{(j)}) = 1 \mid \mathsf{pk}^* \neq \perp \ \wedge \ \widetilde{m}^{(j)} \neq \perp] \tag{7.12}$$

$$\geqslant \Pr[\mathsf{pk}^* = \mathsf{pk}] \cdot \Pr[\widetilde{m}^{(j)} = m^{(j)}] \cdot \Pr[\mathsf{Verify}_{\mathsf{pk}^*}(\widetilde{x}^{(j)} \circ j, \widetilde{\sigma}^{(j)}) = 1 \mid \mathsf{pk}^* = \mathsf{pk} \ \wedge \ \widetilde{m}^{(j)} = m^{(j)}]. \tag{7.13}$$

Here, $m^{(j)} := (x^{(j)} \circ \sigma^{(j)} \circ \mathsf{pk} \circ j)$, Equation (7.12) follows since pk^* and $\widetilde{m}^{(j)}$ are generated independently, and Equation (7.13) follows since these events are a subset of the events in Equation (7.12). Now we analyze each of the terms of Equation (7.13). First note that the final term $\Pr[\mathsf{Verify}_{\mathsf{pk}^*}(\widetilde{x}^{(j)} \circ j, \widetilde{\sigma}^{(j)}) = 1 \mid \mathsf{pk}^* = \mathsf{pk} \ \wedge \ \widetilde{m}^{(j)} = m^{(j)}] = 1$ by definition. Thus we analyze the other two probability terms. By our prior work, we know that $\Pr[\mathsf{pk}^* = \mathsf{pk}] \geqslant 1 - \exp(-\mu \cdot (q - 1/2)^2/(2q))$.

Next we lower bound $\Pr[\widetilde{m}^{(j)} = m^{(j)}]$. Recall that $\widetilde{m}^{(j)} = \mathsf{NBS}^{\widetilde{C}}(j)$. By Lemma 7.5.3 we have that

$$\Pr\left[\Pr_{j \xleftarrow{\$} [d]}\left[\widetilde{m}^{(j)} \neq C^{(j)} \mid j \text{ is } \gamma\text{-good}\right] \geqslant \rho^*\right] \leqslant \vartheta(n')$$

Fix j to be a γ-good block. Then we have

$$\Pr\left[\Pr[\widetilde{m}^{(j)} \neq C^{(j)}] \geqslant \rho^*\right] \leqslant \vartheta(n').$$

Or equivalently

$$\Pr\left[\Pr[\widetilde{m}^{(j)} = C^{(j)}] \leqslant 1 - \rho^*\right] \leqslant \vartheta(n').$$

This implies

$$\Pr\left[\Pr\left[\widetilde{m}^{(j)} = C^{(j)}\right] \geqslant 1 - \rho^*\right] \geqslant 1 - \vartheta(n').$$

Thus with overwhelming probability $1 - \vartheta(n')$, where $\vartheta(\cdot)$ is a negligible function, we have that $\widetilde{m}^{(j)} = C^{(j)}$ with probability at least $1 - \rho^*$. Now by Lemma 7.5.2, we know that the total fraction of γ-good blocks in $\widetilde{C}$ is at most $1 - (2 \cdot \beta \cdot \rho)/(\gamma \cdot \alpha)$. Note that every index $i \in [k]$ is uniquely associated with block $j \in [d]$ satisfying $(j-1) \cdot r(\lambda) < i \leqslant j \cdot r(\lambda)$. Then if $\mathcal{G} := \{i \in [k] \colon i \text{ is in a } \gamma\text{-good block}\}$, we have

$$\begin{aligned} |\mathcal{G}| &\geqslant r(\lambda) \cdot \left(1 - \frac{2\beta\rho}{\gamma\alpha}\right) \cdot d \\ &= \left(1 - \frac{2\beta\rho}{\gamma\alpha}\right) \cdot k. \end{aligned}$$

Therefore at least $1 - (2 \cdot \beta \cdot \rho)/(\gamma \cdot \alpha)$-fraction of indices $i \in [k]$ lie within a γ-good block.

Putting it all together, for any γ-good block we have that with probability at least $1 - \vartheta(n')$:

$$\Pr\left[D_i = x_i\right] \geqslant (1 - \exp(-\mu \cdot (q - 1/2)^2/(2q))) \cdot (1 - \rho^*).$$

Choosing μ and $\rho^* \in (0, 1/2)$ appropriately such that $(1 - \eta) \cdot (1 - \rho^*) > 2/3$, along with the fact that at least $1 - (2 \cdot \beta \cdot \rho)/(\gamma \cdot \alpha)$-fraction of indices $i \in [k]$ lie within a γ-good block, we have that with probability at least $1 - \vartheta(n')$

$$|\mathsf{Good}(\widetilde{C})| \geqslant \left(1 - \frac{2 \cdot \beta \cdot \rho}{\gamma \cdot \alpha}\right) \cdot k.$$

In other words, for $\delta = 1 - (2 \cdot \beta \cdot \rho)/(\gamma \cdot \alpha)$, we have that

$$\Pr\left[\mathsf{Limit}(\widetilde{C}, \rho, \delta, x, C) = 1\right] \leqslant \vartheta(n').$$

□

REFERENCES

[1] E. Abbe, "Polarization and randomness extraction," in *2011 IEEE International Symposium on Information Theory Proceedings*, 2011, pp. 184–188. DOI: 10.1109/ISIT.2011.6033870.

[2] N. Alon, O. Goldreich, J. Hastad, and R. Peralta, "Simple construction of almost k-wise independent random variables," in *Proceedings [1990] 31st Annual Symposium on Foundations of Computer Science*, 1990, 544–553 vol.2. DOI: 10.1109/FSCS.1990.89575.

[3] N. Alon and Y. Roichman, "Random cayley graphs and expanders," *Random Structures & Algorithms*, vol. 5, no. 2, pp. 271–284, 1994.

[4] J. Alwen, J. Blocki, and B. Harsha, "Practical graphs for optimal side-channel resistant memory-hard functions," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17, Dallas, Texas, USA: Association for Computing Machinery, 2017, pp. 1001–1017, ISBN: 9781450349468. DOI: 10.1145/3133956.3134031. [Online]. Available: https://doi.org/10.1145/3133956.3134031.

[5] J. Alwen and J. Blocki, "Efficiently computing data-independent memory-hard functions," in *Advances in Cryptology – CRYPTO 2016*, M. Robshaw and J. Katz, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 241–271, ISBN: 978-3-662-53008-5.

[6] J. Alwen, J. Blocki, and K. Pietrzak, "Depth-robust graphs and their cumulative memory complexity," in *Advances in Cryptology – EUROCRYPT 2017*, J.-S. Coron and J. B. Nielsen, Eds., Cham: Springer International Publishing, 2017, pp. 3–32, ISBN: 978-3-319-56617-7.

[7] J. Alwen, J. Blocki, and K. Pietrzak, "Sustained space complexity," in *Advances in Cryptology – EUROCRYPT 2018*, J. B. Nielsen and V. Rijmen, Eds., Cham: Springer International Publishing, 2018, pp. 99–130, ISBN: 978-3-319-78375-8.

[8] J. Alwen, B. Chen, K. Pietrzak, L. Reyzin, and S. Tessaro, "Scrypt is maximally memory-hard," in *Advances in Cryptology – EUROCRYPT 2017*, J.-S. Coron and J. B. Nielsen, Eds., Cham: Springer International Publishing, 2017, pp. 33–62, ISBN: 978-3-319-56617-7.

[9] J. Alwen and V. Serbinenko, "High parallel complexity graphs and memory-hard functions," in *Proceedings of the Forty-Seventh Annual ACM Symposium on Theory of Computing*, ser. STOC '15, Portland, Oregon, USA: Association for Computing Machinery, 2015, pp. 595–603, ISBN: 9781450335362. DOI: 10.1145/2746539.2746622. [Online]. Available: https://doi.org/10.1145/2746539.2746622.

[10] J. Alwen and B. Tackmann, "Moderately hard functions: Definition, instantiations, and applications," in *Theory of Cryptography*, Y. Kalai and L. Reyzin, Eds., Cham: Springer International Publishing, 2017, pp. 493–526, ISBN: 978-3-319-70500-2.

[11] A. Aly, T. Ashur, E. Ben-Sasson, S. Dhooghe, and A. Szepieniec, "Design of symmetric-key primitives for advanced cryptographic protocols," *IACR Transactions on Symmetric Cryptology*, vol. 2020, no. 3, pp. 1–45, Sep. 2020. DOI: 10.13154/tosc.v2020.i3.1-45. [Online]. Available: https://tosc.iacr.org/index.php/ToSC/article/view/8695.

[12] A. Ambainis, "Upper bound on the communication complexity of private information retrieval," in *Automata, Languages and Programming*, P. Degano, R. Gorrieri, and A. Marchetti-Spaccamela, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, pp. 401–407, ISBN: 978-3-540-69194-5.

[13] M. H. Ameri, A. R. Block, and J. Blocki, *Memory-hard puzzles in the standard model with applications to memory-hard functions and resource-bounded locally decodable codes*, Cryptology ePrint Archive, Report 2021/801, https://ia.cr/2021/801, 2021.

[14] S. Ames, C. Hazay, Y. Ishai, and M. Venkitasubramaniam, "Ligero: Lightweight sublinear arguments without a trusted setup," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17, Dallas, Texas, USA: Association for Computing Machinery, 2017, pp. 2087–2104, ISBN: 9781450349468. DOI: 10.1145/3133956.3134104. [Online]. Available: https://doi.org/10.1145/3133956.3134104.

[15] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy, "Proof verification and hardness of approximation problems," in *Proceedings., 33rd Annual Symposium on Foundations of Computer Science*, 1992, pp. 14–23. DOI: 10.1109/SFCS.1992.267823.

[16] S. Arora and S. Safra, "Probabilistic checking of proofs; a new characterization of np," in *Proceedings., 33rd Annual Symposium on Foundations of Computer Science*, 1992, pp. 2–13. DOI: 10.1109/SFCS.1992.267824.

[17] S. Arora and B. Barak, *Computational Complxity: A Modern Approach*. Cambridge University Press, 2009.

[18] M. Asaar, M. Ameri, M. Salmasizadeh, and M. Aref, "A provably secure code-based concurrent signature scheme," *IET Information Security*, vol. 12, Aug. 2017. DOI: 10.1049/iet-ifs.2017.0023.

[19] M. R. Asaar, *Code-based strong designated verifier signatures: Security analysis and a new construction*, Cryptology ePrint Archive, Report 2016/779, https://ia.cr/2016/779, 2016.

[20] L. Babai and L. Fortnow, "Non-deterministic exponential time has two-prover interactive protocols," in *Computational Complexity*, vol. 1, 1991, pp. 3–40. DOI: 10.1007/BF01200056.

[21] L. Babai and L. Fortnow, "Arithmetization: A new method in structural complexity theory," *computational complexity*, vol. 1, no. 1, pp. 41–66, Mar. 1991, ISSN: 1420-8954. DOI: 10.1007/BF01200057. [Online]. Available: https://doi.org/10.1007/BF01200057.

[22] L. Babai, L. Fortnow, L. A. Levin, and M. Szegedy, "Checking computations in polylogarithmic time," in *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, May 5-8, 1991, New Orleans, Louisiana, USA*, C. Koutsougeras and J. S. Vitter, Eds., ACM, 1991, pp. 21–31. DOI: 10.1145/103418.103428. [Online]. Available: https://doi.org/10.1145/103418.103428.

[23] L. Babai and S. Moran, "Arthur-merlin games: A randomized proof system, and a hierarchy of complexity classes," *Journal of Computer and System Sciences*, vol. 36, no. 2, pp. 254–276, 1988, ISSN: 0022-0000. DOI: https://doi.org/10.1016/0022-0000(88)90028-1. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0022000088900281.

[24] B. Barak, "How to go beyond the black-box simulation barrier," in *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, 2001, pp. 106–115. DOI: 10.1109/SFCS.2001.959885.

[25] A. Beimel, Y. Ishai, E. Kushilevitz, and J.-F. Raymond, "Breaking the o(n/sup 1/(2k-1)/) barrier for information-theoretic private information retrieval," in *The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002. Proceedings.*, 2002, pp. 261–270. DOI: 10.1109/SFCS.2002.1181949.

[26] S. M. Bellovin, "Frank miller: Invetnor of the one-time pad," *Cryptologia*, vol. 35, no. 3, pp. 203–222, 2011.

[27] M. Ben-Or, S. Goldwasser, J. Kilian, and A. Wigderson, "Multi-prover interactive proofs: How to remove intractability assumptions," in *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, ser. STOC '88, Chicago, Illinois, USA: Association for Computing Machinery, 1988, pp. 113–131, ISBN: 0897912640. DOI: 10.1145/62212.62223. [Online]. Available: https://doi.org/10.1145/62212.62223.

[28] M. Ben-Or, S. Goldwasser, and A. Wigderson, "Completeness theorems for non-cryptographic fault-tolerant distributed computation," in *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, ser. STOC '88, Chicago, Illinois, USA: Association for Computing Machinery, 1988, pp. 1–10, ISBN: 0897912640. DOI: 10.1145/62212.62213. [Online]. Available: https://doi.org/10.1145/62212.62213.

[29] M. Ben-Or *et al.*, "Everything provable is provable in zero-knowledge," in *Advances in Cryptology — CRYPTO' 88*, S. Goldwasser, Ed., New York, NY: Springer New York, 1990, pp. 37–56, ISBN: 978-0-387-34799-8.

[30] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev, "Fast Reed-Solomon Interactive Oracle Proofs of Proximity," in *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*, I. Chatzigiannakis, C. Kaklamanis, D. Marx, and D. Sannella, Eds., ser. Leibniz International Proceedings in Informatics (LIPIcs), vol. 107, Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018, 14:1–14:17, ISBN: 978-3-95977-076-7. DOI: 10.4230/LIPIcs.ICALP.2018.14. [Online]. Available: http://drops.dagstuhl.de/opus/volltexte/2018/9018.

[31] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev, "Scalable zero knowledge with no trusted setup," in *Advances in Cryptology – CRYPTO 2019*, A. Boldyreva and D. Micciancio, Eds., Cham: Springer International Publishing, 2019, pp. 701–732, ISBN: 978-3-030-26954-8.

[32] E. Ben-Sasson, D. Carmon, Y. Ishai, S. Kopparty, and S. Saraf, "Proximity gaps for reed–solomon codes," in *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, 2020, pp. 900–909. DOI: 10.1109/FOCS46700.2020.00088.

[33] E. Ben-Sasson, A. Chiesa, D. Genkin, and E. Tromer, "Fast reductions from rams to delegatable succinct constraint satisfaction problems: Extended abstract," in *Proceedings of the 4th Conference on Innovations in Theoretical Computer Science*, ser. ITCS '13, Berkeley, California, USA: Association for Computing Machinery, 2013, pp. 401–414, ISBN: 9781450318594. DOI: 10.1145/2422436.2422481. [Online]. Available: https://doi.org/10.1145/2422436.2422481.

[34] E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer, and M. Virza, "Snarks for c: Verifying program executions succinctly and in zero knowledge," in *Advances in Cryptology – CRYPTO 2013*, R. Canetti and J. A. Garay, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 90–108, ISBN: 978-3-642-40084-1.

[35] E. Ben-Sasson, A. Chiesa, and N. Spooner, "Interactive oracle proofs," in *Theory of Cryptography*, M. Hirt and A. Smith, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 31–60, ISBN: 978-3-662-53644-5.

[36] E. Ben-Sasson, L. Goldberg, S. Kopparty, and S. Saraf, "DEEP-FRI: Sampling Outside the Box Improves Soundness," in *11th Innovations in Theoretical Computer Science Conference (ITCS 2020)*, T. Vidick, Ed., ser. Leibniz International Proceedings in Informatics (LIPIcs), vol. 151, Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2020, 5:1–5:32, ISBN: 978-3-95977-134-4. DOI: 10.4230/LIPIcs.ITCS.2020.5. [Online]. Available: https://drops.dagstuhl.de/opus/volltexte/2020/11690.

[37] E. Ben-Sasson, O. Goldreich, P. Harsha, M. Sudan, and S. Vadhan, "Robust pcps of proximity, shorter pcps and applications to coding," in *Proceedings of the Thirty-Sixth Annual ACM Symposium on Theory of Computing*, ser. STOC '04, Chicago, IL, USA: Association for Computing Machinery, 2004, pp. 1–10, ISBN: 1581138520. DOI: 10.1145/1007352.1007361. [Online]. Available: https://doi.org/10.1145/1007352.1007361.

[38] C. H. Bennett, G. Brassard, and J.-M. Robert, "Privacy amplification by public discussion," *SIAM J. Comput.*, vol. 17, no. 2, pp. 210–229, Apr. 1988, ISSN: 0097-5397. DOI: 10.1137/0217014. [Online]. Available: https://doi.org/10.1137/0217014.

[39] A. Biryukov, D. Dinu, and D. Khovratovich, "Argon2: New generation of memory-hard functions for password hashing and other applications," in *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, IEEE, 2016, pp. 292–302.

[40] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer, "Recursive composition and bootstrapping for snarks and proof-carrying data," in *Proceedings of the Forty-Fifth Annual ACM Symposium on Theory of Computing*, ser. STOC '13, Palo Alto, California, USA: Association for Computing Machinery, 2013, pp. 111–120, ISBN: 9781450320290. DOI: 10.1145/2488608.2488623. [Online]. Available: https://doi.org/10.1145/2488608.2488623.

[41] N. Bitansky and A. Chiesa, "Succinct arguments from multi-prover interactive proofs and their efficiency benefits," in *Advances in Cryptology – CRYPTO 2012*, R. Safavi-Naini and R. Canetti, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 255–272, ISBN: 978-3-642-32009-5.

[42] N. Bitansky, A. Chiesa, Y. Ishai, O. Paneth, and R. Ostrovsky, "Succinct non-interactive arguments via linear interactive proofs," in *Theory of Cryptography*, A. Sahai, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 315–333, ISBN: 978-3-642-36594-2.

[43] N. Bitansky, S. Goldwasser, A. Jain, O. Paneth, V. Vaikuntanathan, and B. Waters, "Time-lock puzzles from randomized encodings," in *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science*, ser. ITCS '16, Cambridge, Massachusetts, USA: Association for Computing Machinery, 2016, pp. 345–356, ISBN: 9781450340571. DOI: 10.1145/2840728.2840745. [Online]. Available: https://doi.org/10.1145/2840728.2840745.

[44] G. R. BLAKLEY, "Safeguarding cryptographic keys," in *1979 International Workshop on Managing Requirements Knowledge (MARK)*, 1979, pp. 313–318. DOI: 10.1109/MARK.1979.8817296.

[45] A. R. Block and J. Blocki, *Computationally relaxed locally decodable codes for insertion-deletion errors*, in submission, 2022.

[46] A. R. Block and J. Blocki, "Private and resource-bounded locally decodable codes for insertions and deletions," in *2021 IEEE International Symposium on Information Theory (ISIT)*, 2021, pp. 1841–1846. DOI: 10.1109/ISIT45174.2021.9518249.

[47] A. R. Block and J. Blocki, "Private and resource-bounded locally decodable codes for insertions and deletions," *CoRR*, vol. abs/2103.14122, 2021. arXiv: 2103.14122. [Online]. Available: https://arxiv.org/abs/2103.14122.

[48] A. R. Block, J. Blocki, E. Grigorescu, S. Kulkarni, and M. Zhu, "Locally Decodable/-Correctable Codes for Insertions and Deletions," in *FSTTCS 2020*, N. Saxena and S. Simon, Eds., ser. Leibniz International Proceedings in Informatics (LIPIcs), vol. 182, Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020, 16:1–16:17, ISBN: 978-3-95977-174-0. DOI: 10.4230/LIPIcs.FSTTCS.2020.16.

[49] A. R. Block, D. Gupta, H. K. Maji, and H. H. Nguyen, "Secure computation using leaky correlations (asymptotically optimal constructions)," in *Theory of Cryptography*, A. Beimel and S. Dziembowski, Eds., Cham: Springer International Publishing, 2018, pp. 36–65, ISBN: 978-3-030-03810-6.

[50] A. R. Block, D. Gupta, H. K. Maji, and H. H. Nguyen, *Secure computation using leaky correlations (asymptotically optimal constructions)*, Cryptology ePrint Archive, Report 2018/372, https://ia.cr/2018/372, 2018.

[51] A. R. Block, J. Holmgren, A. Rosen, R. D. Rothblum, and P. Soni, "Public-coin zero-knowledge arguments with (almost) minimal time and space overheads," in *Theory of Cryptography*, R. Pass and K. Pietrzak, Eds., Cham: Springer International Publishing, 2020, pp. 168–197, ISBN: 978-3-030-64378-2.

[52] A. R. Block, J. Holmgren, A. Rosen, R. D. Rothblum, and P. Soni, "Time- and space-efficient arguments from groups of unknown order," in *Advances in Cryptology – CRYPTO 2021*, T. Malkin and C. Peikert, Eds., Cham: Springer International Publishing, 2021, pp. 123–152, ISBN: 978-3-030-84259-8.

[53] A. R. Block, J. Holmgren, A. Rosen, R. D. Rothblum, and P. Soni, *Time- and space-efficient arguments from groups of unknown order*, Cryptology ePrint Archive, Paper 2021/358, https://eprint.iacr.org/2021/358, 2021. [Online]. Available: https://eprint.iacr.org/2021/358.

[54] A. R. Block, H. K. Maji, and H. H. Nguyen, "Secure computation based on leaky correlations: High resilience setting," in *Advances in Cryptology – CRYPTO 2017*, J. Katz and H. Shacham, Eds., Cham: Springer International Publishing, 2017, pp. 3–32, ISBN: 978-3-319-63715-0.

[55] A. R. Block, H. K. Maji, and H. H. Nguyen, "Secure computation with constant communication overhead using multiplication embeddings," in *Progress in Cryptology – INDOCRYPT 2018*, D. Chakraborty and T. Iwata, Eds., Cham: Springer International Publishing, 2018, pp. 375–398, ISBN: 978-3-030-05378-9.

[56] J. Blocki, K. Cheng, E. Grigorescu, X. Li, Y. Zheng, and M. Zhu, "Exponential lower bounds for locally decodable and correctable codes for insertions and deletions," in *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, IEEE, 2021, pp. 739–750. DOI: 10.1109/FOCS52979.2021.00077. [Online]. Available: https://doi.org/10.1109/FOCS52979.2021.00077.

[57] J. Blocki, V. Gandikota, E. Grigorescu, and S. Zhou, "Relaxed locally correctable codes in computationally bounded channels," *IEEE Transactions on Information Theory*, vol. 67, no. 7, pp. 4338–4360, 2021. DOI: 10.1109/TIT.2021.3076396.

[58] J. Blocki, V. Gandikota, E. Grigorescu, and S. Zhou, "Relaxed locally correctable codes in computationally bounded channels," *IEEE Transactions on Information Theory*, vol. 67, no. 7, pp. 4338–4360, 2021. DOI: 10.1109/TIT.2021.3076396.

[59] J. Blocki, B. Harsha, S. Kang, S. Lee, L. Xing, and S. Zhou, "Data-independent memory hard functions: New attacks and stronger constructions," in *Advances in Cryptology – CRYPTO 2019*, A. Boldyreva and D. Micciancio, Eds., Cham: Springer International Publishing, 2019, pp. 573–607, ISBN: 978-3-030-26951-7.

[60] J. Blocki, S. Kulkarni, and S. Zhou, "On Locally Decodable Codes in Resource Bounded Channels," in *1st Conference on Information-Theoretic Cryptography (ITC 2020)*, Y. T. Kalai, A. D. Smith, and D. Wichs, Eds., ser. Leibniz International Proceedings in Informatics (LIPIcs), vol. 163, Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020, 16:1–16:23, ISBN: 978-3-95977-151-1. DOI: 10.4230/LIPIcs.ITC.2020.16. [Online]. Available: https://drops.dagstuhl.de/opus/volltexte/2020/12121.

[61] J. Blocki and S. Lee, *On the multi-user security of short schnorr signatures with preprocessing*, Cryptology ePrint Archive, Report 2019/1105, https://ia.cr/2019/1105. To appear in EUROCRYPT 2022., 2019.

[62] J. Blocki and S. Zhou, "On the depth-robustness and cumulative pebbling cost of argon2i," in *Theory of Cryptography*, Y. Kalai and L. Reyzin, Eds., Cham: Springer International Publishing, 2017, pp. 445–465, ISBN: 978-3-319-70500-2.

[63] M. Blum and S. Kanna, "Designing programs that check their work," in *Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing*, ser. STOC '89, Seattle, Washington, USA: Association for Computing Machinery, 1989, pp. 86–97, ISBN: 0897913078. DOI: 10.1145/73007.73015. [Online]. Available: https://doi.org/10.1145/73007.73015.

[64] A. J. Blumberg, J. Thaler, V. Vu, and M. Walfish, *Verifiable computation using multiple provers*, Cryptology ePrint Archive, Report 2014/846, https://ia.cr/2014/846, 2014.

[65] D. Boneh, B. Bünz, and B. Fisch, *A survey of two verifiable delay functions*, Cryptology ePrint Archive, Report 2018/712, https://ia.cr/2018/712, 2018.

[66] D. Boneh and M. Naor, "Timed commitments," in *Advances in Cryptology — CRYPTO 2000*, M. Bellare, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 236–254, ISBN: 978-3-540-44598-2.

[67] J. Bootle, A. Cerulli, P. Chaidos, J. Groth, and C. Petit, "Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting," in *Advances in Cryptology – EUROCRYPT 2016*, M. Fischlin and J.-S. Coron, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 327–357, ISBN: 978-3-662-49896-5.

[68] J. Bootle, A. Chiesa, Y. Hu, and M. Orrù, *Gemini: Elastic snarks for diverse environments*, Cryptology ePrint Archive, Report 2022/420, https://ia.cr/2022/420, 2022.

[69] S. Bowe, J. Grigg, and D. Hopwood, *Recursive proof composition without a trusted setup*, Cryptology ePrint Archive, Report 2019/1021, https://ia.cr/2019/1021, 2019.

[70] J. Brakensiek, V. Guruswami, and S. Zbarsky, "Efficient low-redundancy codes for correcting multiple deletions," *IEEE Transactions on Information Theory*, vol. 64, no. 5, pp. 3403–3410, 2018. DOI: 10.1109/TIT.2017.2746566.

[71] M. Braverman and E. Mossel, "Noisy sorting without resampling," in *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2008, San Francisco, California, USA, January 20-22, 2008*, S. Teng, Ed., SIAM, 2008, pp. 268–276. [Online]. Available: http://dl.acm.org/citation.cfm?id=1347082.1347112.

[72] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell, "Bulletproofs: Short proofs for confidential transactions and more," in *2018 IEEE Symposium on Security and Privacy (SP)*, 2018, pp. 315–334. DOI: 10.1109/SP.2018.00020.

[73] B. Bünz, A. Chiesa, P. Mishra, and N. Spooner, "Recursive proof composition from accumulation schemes," in *Theory of Cryptography*, R. Pass and K. Pietrzak, Eds., Cham: Springer International Publishing, 2020, pp. 1–18, ISBN: 978-3-030-64378-2.

[74] B. Bünz, B. Fisch, and A. Szepieniec, Personal Communication, 2021.

[75] B. Bünz, B. Fisch, and A. Szepieniec, *Transparent snarks from dark compilers*, Cryptology ePrint Archive, Report 2019/1229, https://ia.cr/2019/1229, 2019.

[76] B. Bünz, B. Fisch, and A. Szepieniec, "Transparent snarks from dark compilers," in *Advances in Cryptology – EUROCRYPT 2020*, A. Canteaut and Y. Ishai, Eds., Cham: Springer International Publishing, 2020, pp. 677–706, ISBN: 978-3-030-45721-1.

[77] R. Canetti, Y. Lindell, R. Ostrovsky, and A. Sahai, "Universally composable two-party and multi-party secure computation," in *Proceedings of the Thiry-Fourth Annual ACM Symposium on Theory of Computing*, ser. STOC '02, Montreal, Quebec, Canada: Association for Computing Machinery, 2002, pp. 494–503, ISBN: 1581134959. DOI: 10.1145/509907.509980. [Online]. Available: https://doi.org/10.1145/509907.509980.

[78] I. Cascudo, R. Cramer, C. Xing, and C. Yuan, "Amortized complexity of information-theoretically secure mpc revisited," in *Advances in Cryptology – CRYPTO 2018*, H. Shacham and A. Boldyreva, Eds., Cham: Springer International Publishing, 2018, pp. 395–426, ISBN: 978-3-319-96878-0.

[79] I. Cascudo, I. Damgård, O. Farràs, and S. Ranellucci, "Resource-efficient ot combiners with active security," in *Theory of Cryptography*, Y. Kalai and L. Reyzin, Eds., Cham: Springer International Publishing, 2017, pp. 461–486, ISBN: 978-3-319-70503-3.

[80] M. Cenk and F. Özbudak, "On multiplication in finite fields," *J. Complex.*, vol. 26, no. 2, pp. 172–186, Apr. 2010, ISSN: 0885-064X. DOI: 10.1016/j.jco.2009.11.002. [Online]. Available: https://doi.org/10.1016/j.jco.2009.11.002.

[81] N. Chandran, V. Goyal, and A. Sahai, "New constructions for uc secure computation using tamper-proof hardware," in *Advances in Cryptology – EUROCRYPT 2008*, N. Smart, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 545–562, ISBN: 978-3-540-78967-3.

[82] D. Chaum, C. Crépeau, and I. Damgard, "Multiparty unconditionally secure protocols," in *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, ser. STOC '88, Chicago, Illinois, USA: Association for Computing Machinery, 1988, pp. 11–19, ISBN: 0897912640. DOI: 10.1145/62212.62214. [Online]. Available: https://doi.org/10.1145/62212.62214.

[83] B. Chen and S. Tessaro, "Memory-hard functions from cryptographic primitives," in *Advances in Cryptology – CRYPTO 2019*, A. Boldyreva and D. Micciancio, Eds., Cham: Springer International Publishing, 2019, pp. 543–572, ISBN: 978-3-030-26951-7.

[84] H. Chen and R. Cramer, "Algebraic geometric secret sharing schemes and secure multi-party computations over small fields," in *Advances in Cryptology - CRYPTO 2006*, C. Dwork, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 521–536, ISBN: 978-3-540-37433-6.

[85] K. Cheng, V. Guruswami, B. Haeupler, and X. Li, *Efficient linear and affine codes for correcting insertions/deletions*, 2020. arXiv: 2007.09075 [cs.IT].

[86] K. Cheng, B. Haeupler, X. Li, A. Shahrasbi, and K. Wu, "Synchronization strings: Highly efficient deterministic constructions over small alphabets," in *Proceedings of the 2019 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 2185–2204. DOI: 10.1137/1.9781611975482.132. eprint: https://epubs.siam.org/doi/pdf/10.1137/1.9781611975482.132. [Online]. Available: https://epubs.siam.org/doi/abs/10.1137/1.9781611975482.132.

[87] K. Cheng, Z. Jin, X. Li, and K. Wu, "Block Edit Errors with Transpositions: Deterministic Document Exchange Protocols and Almost Optimal Binary Codes," in *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*, C. Baier, I. Chatzigiannakis, P. Flocchini, and S. Leonardi, Eds., ser. Leibniz International Proceedings in Informatics (LIPIcs), vol. 132, Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019, 37:1–37:15, ISBN: 978-3-95977-109-2. DOI: 10.4230/LIPIcs.ICALP.2019.37. [Online]. Available: http://drops.dagstuhl.de/opus/volltexte/2019/10613.

[88] K. Cheng, Z. Jin, X. Li, and K. Wu, "Deterministic document exchange protocols, and almost optimal binary codes for edit errors," in *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, 2018, pp. 200–211. DOI: 10.1109/FOCS.2018.00028.

[89] K. Cheng and X. Li, *Efficient document exchange and error correcting codes with asymmetric information*, 2020. arXiv: 2007.00870 [cs.CC].

[90] K. Cheng, X. Li, and Y. Zheng, *Locally decodable codes with randomized encoding*, Cryptology ePrint Archive, Report 2020/031, https://ia.cr/2020/031, 2020.

[91] A. Chiesa, D. Ojha, and N. Spooner, "Fractal: Post-quantum and transparent recursive proofs from holography," in *Advances in Cryptology – EUROCRYPT 2020*, A. Canteaut and Y. Ishai, Eds., Cham: Springer International Publishing, 2020, pp. 769–793, ISBN: 978-3-030-45721-1.

[92] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan, "Private information retrieval," in *Proceedings of IEEE 36th Annual Foundations of Computer Science*, 1995, pp. 41–50. DOI: 10.1109/SFCS.1995.492461.

[93] B. Chor, O. Goldreich, J. Hasted, J. Freidmann, S. Rudich, and R. Smolensky, "The bit extraction problem or t-resilient functions," in *26th Annual Symposium on Foundations of Computer Science (sfcs 1985)*, 1985, pp. 396–407. DOI: 10.1109/SFCS.1985.55.

[94] D. V. Chudnovsky and G. V. Chudnovsky, "Algebraic complexities and algebraic curves over finite fields," *Proceedings of the National Academy of Sciences*, vol. 84, no. 7, pp. 1739–1743, 1987.

[95] G. Cormode, M. Mitzenmacher, and J. Thaler, "Practical verified computation with streaming interactive proofs," in *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ser. ITCS '12, Cambridge, Massachusetts: Association for Computing Machinery, 2012, pp. 90–112, ISBN: 9781450311151. DOI: 10.1145/2090236.2090245. [Online]. Available: https://doi.org/10.1145/2090236.2090245.

[96] N. T. Courtois, M. Finiasz, and N. Sendrier, "How to achieve a mceliece-based digital signature scheme," in *Advances in Cryptology — ASIACRYPT 2001*, C. Boyd, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 157–174, ISBN: 978-3-540-45682-7.

[97] C. Crépeau, K. Morozov, and S. Wolf, "Efficient unconditional oblivious transfer from almost any noisy channel," in *Security in Communication Networks*, C. Blundo and S. Cimato, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 47–59, ISBN: 978-3-540-30598-9.

[98] L. Dallot, "Towards a concrete security proof of courtois, finiasz and sendrier signature scheme," in *Research in Cryptology*, S. Lucks, A.-R. Sadeghi, and C. Wolf, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 65–77, ISBN: 978-3-540-88353-1.

[99] I. Damgård and Y. Ishai, "Scalable secure multiparty computation," in *Advances in Cryptology - CRYPTO 2006*, C. Dwork, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 501–520, ISBN: 978-3-540-37433-6.

[100] I. Damgård and J. B. Nielsen, "Scalable and unconditionally secure multiparty computation," in *Advances in Cryptology - CRYPTO 2007*, A. Menezes, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 572–590, ISBN: 978-3-540-74143-5.

[101] I. Damgård, J. B. Nielsen, and D. Wichs, "Isolated proofs of knowledge and isolated zero knowledge," in *Advances in Cryptology – EUROCRYPT 2008*, N. Smart, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 509–526, ISBN: 978-3-540-78967-3.

[102] I. Damgård, V. Pastro, N. Smart, and S. Zakarias, "Multiparty computation from somewhat homomorphic encryption," in *Advances in Cryptology – CRYPTO 2012*, R. Safavi-Naini and R. Canetti, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 643–662, ISBN: 978-3-642-32009-5.

[103] A. Dhagat, P. Gács, and P. Winkler, "On playing "twenty questions" with a liar," in *Proceedings of the Third Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms, 27-29 January 1992, Orlando, Florida, USA*, G. N. Frederickson, Ed., ACM/SIAM, 1992, pp. 16–22. [Online]. Available: http://dl.acm.org/citation.cfm?id=139404.139409.

[104] I. Dinur, *Probabilistically checkable proofs and codes*, 2000.

[105] Y. Dodis, R. Ostrovsky, L. Reyzin, and A. Smith, "Fuzzy extractors: How to generate strong keys from biometrics and other noisy data," *SIAM Journal on Computing*, vol. 38, no. 1, pp. 97–139, 2008. DOI: 10.1137/060651380. eprint: https://doi.org/10.1137/060651380. [Online]. Available: https://doi.org/10.1137/060651380.

[106] Y. Dodis and A. Smith, "Correcting errors without leaking partial information," in *Proceedings of the Thirty-Seventh Annual ACM Symposium on Theory of Computing*, ser. STOC '05, Baltimore, MD, USA: Association for Computing Machinery, 2005, pp. 654–663, ISBN: 1581139608. DOI: 10.1145/1060590.1060688. [Online]. Available: https://doi.org/10.1145/1060590.1060688.

[107] Z. Durumeric *et al.*, "The matter of heartbleed," in *Proceedings of the 2014 Conference on Internet Measurement Conference*, ser. IMC '14, Vancouver, BC, Canada: Association for Computing Machinery, 2014, pp. 475–488, ISBN: 9781450332132. DOI: 10.1145/2663716.2663755. [Online]. Available: https://doi.org/10.1145/2663716.2663755.

[108] Z. Dvir, P. Gopalan, and S. Yekhanin, "Matching vector codes," *SIAM J. Comput.*, vol. 40, no. 4, pp. 1154–1178, 2011.

[109] K. Efremenko, "3-query locally decodable codes of subexponential length," *SIAM J. Comput.*, vol. 41, no. 6, pp. 1694–1703, 2012.

[110] N. Ephraim, C. Freitag, I. Komargodski, and R. Pass, "Sparks: Succinct parallelizable arguments of knowledge," in *Advances in Cryptology – EUROCRYPT 2020*, A. Canteaut and Y. Ishai, Eds., Cham: Springer International Publishing, 2020, pp. 707–737, ISBN: 978-3-030-45721-1.

[111] P. Erdös, R. Graham, and E. Szemerédi, "On sparse graphs with dense long paths," *Computers & Mathematics with Applications*, vol. 1, no. 3, pp. 365–369, 1975, ISSN: 0898-1221. DOI: 10.1016/0898-1221(75)90037-1.

[112] P. Erdös, R. L. Graham, and E. Szemeredi, "On sparse graphs with dense long paths.," Stanford University, Stanford, CA, USA, Tech. Rep., 1975.

[113] M. F. Ezerman, H. T. Lee, S. Ling, K. Nguyen, and H. Wang, "A provably secure group signature scheme from code-based assumptions," in *Advances in Cryptology – ASIACRYPT 2015*, T. Iwata and J. H. Cheon, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 260–285, ISBN: 978-3-662-48797-6.

[114] U. Feige, S. Goldwasser, L. Lovasz, S. Safra, and M. Szegedy, "Approximating clique is almost np-complete," in *[1991] Proceedings 32nd Annual Symposium of Foundations of Computer Science*, 1991, pp. 2–12. DOI: 10.1109/SFCS.1991.185341.

[115] U. Feige, P. Raghavan, D. Peleg, and E. Upfal, "Computing with noisy information," *SIAM J. Comput.*, vol. 23, no. 5, pp. 1001–1018, Oct. 1994, ISSN: 0097-5397. DOI: 10.1137/S0097539791195877. [Online]. Available: https://doi.org/10.1137/S0097539791195877.

[116] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *Advances in Cryptology — CRYPTO' 86*, A. M. Odlyzko, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 1987, pp. 186–194, ISBN: 978-3-540-47721-1.

[117] C. Forler, S. Lucks, and J. Wenzel, "Memory-demanding password scrambling," in *Advances in Cryptology – ASIACRYPT 2014*, P. Sarkar and T. Iwata, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 289–305, ISBN: 978-3-662-45608-8.

[118] G. Forney, "Generalized minimum distance decoding," *IEEE Transactions on Information Theory*, vol. 12, no. 2, pp. 125–131, 1966. DOI: 10.1109/TIT.1966.1053873.

[119] G. D. Forney, *Concatenated codes*, Cambridge, MA: MIT Press, 1966.

[120] J. A. Garay, P. MacKenzie, M. Prabhakaran, and K. Yang, "Resource fairness and composability of cryptographic protocols," *Journal of Cryptology*, vol. 24, no. 4, pp. 615–658, Oct. 2011, ISSN: 1432-1378. DOI: 10.1007/s00145-010-9080-z. [Online]. Available: https://doi.org/10.1007/s00145-010-9080-z.

[121] A. Garcia and H. Stichtenoth, "On the asymptotic behaviour of some towers of function fields over finite fields," *Journal of Number Theory*, vol. 61, no. 2, pp. 248–273, 1996. DOI: doi:10.1006/jnth.1996.0147.

[122] S. Garg and A. Srinivasan, "A simple construction of io for turing machines," in *Theory of Cryptography*, A. Beimel and S. Dziembowski, Eds., Cham: Springer International Publishing, 2018, pp. 425–454, ISBN: 978-3-030-03810-6.

[123] B. Geissmann, S. Leucci, C.-H. Liu, and P. Penna, "Sorting with Recurrent Comparison Errors," in *28th International Symposium on Algorithms and Computation (ISAAC 2017)*, Y. Okamoto and T. Tokuyama, Eds., ser. Leibniz International Proceedings in Informatics (LIPIcs), vol. 92, Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017, 38:1–38:12, ISBN: 978-3-95977-054-5. DOI: 10.4230/LIPIcs.ISAAC.2017.38. [Online]. Available: http://drops.dagstuhl.de/opus/volltexte/2017/8265.

[124] R. Gennaro, C. Gentry, B. Parno, and M. Raykova, "Quadratic span programs and succinct nizks without pcps," in *Advances in Cryptology – EUROCRYPT 2013*, T. Johansson and P. Q. Nguyen, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 626–645, ISBN: 978-3-642-38348-9.

[125] C. Gentry and D. Wichs, "Separating succinct non-interactive arguments from all falsifiable assumptions," in *Proceedings of the Forty-Third Annual ACM Symposium on Theory of Computing*, ser. STOC '11, San Jose, California, USA: Association for Computing Machinery, 2011, pp. 99–108, ISBN: 9781450306911. DOI: 10.1145/1993636.1993651. [Online]. Available: https://doi.org/10.1145/1993636.1993651.

[126] E. N. Gilbert, "A comparison of signalling alphabets," *The Bell System Technical Journal*, vol. 31, no. 3, pp. 504–522, 1952. DOI: 10.1002/j.1538-7305.1952.tb01393.x.

[127] O. Goldreich, H. Karloff, L. Schulman, and L. Trevisan, "Lower bounds for linear locally decodable codes and private information retrieval," in *Proceedings 17th IEEE Annual Conference on Computational Complexity*, 2002, pp. 175–183. DOI: 10.1109/CCC.2002.1004353.

[128] 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*, ser. STOC '89, Seattle, Washington, USA: Association for Computing Machinery, 1989, pp. 25–32, ISBN: 0897913078. DOI: 10.1145/73007.73010. [Online]. Available: https://doi.org/10.1145/73007.73010.

[129] O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game," in *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, ser. STOC '87, New York, New York, USA: Association for Computing Machinery, 1987, pp. 218–229, ISBN: 0897912217. DOI: 10.1145/28395.28420. [Online]. Available: https://doi.org/10.1145/28395.28420.

[130] O. Goldreich and A. Wigderson, "Tiny families of functions with random properties: A quality-size trade-off for hashing," *Random Struct. Algorithms*, vol. 11, no. 4, pp. 315–343, 1997. DOI: 10.1002/(SICI)1098-2418(199712)11:4⟨315::AID-RSA3⟩3.0.CO;2-1. [Online]. Available: https://doi.org/10.1002/(SICI)1098-2418(199712)11:4%3C315::AID-RSA3%3E3.0.CO;2-1.

[131] S. Goldwasser, Y. T. Kalai, and G. N. Rothblum, "Delegating computation: Interactive proofs for muggles," in *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*, ser. STOC '08, Victoria, British Columbia, Canada: Association for Computing Machinery, 2008, pp. 113–122, ISBN: 9781605580470. DOI: 10.1145/1374376.1374396. [Online]. Available: https://doi.org/10.1145/1374376.1374396.

[132] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof systems," *SIAM J. Comput.*, vol. 18, no. 1, pp. 186–208, 1989. DOI: 10.1137/0218012. [Online]. Available: https://doi.org/10.1137/0218012.

[133] V. D. Goppa, "Codes on algebraic curves," in *Soviet Math. Dokl*, 1981, pp. 170–172.

[134] V. Goyal, Y. Ishai, A. Sahai, R. Venkatesan, and A. Wadia, "Founding cryptography on tamper-proof hardware tokens," in *Theory of Cryptography*, D. Micciancio, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 308–326, ISBN: 978-3-642-11799-2.

[135] V. Goyal, Y. Song, and C. Zhu, "Guaranteed output delivery comes free in honest majority mpc," in *Advances in Cryptology – CRYPTO 2020*, D. Micciancio and T. Ristenpart, Eds., Cham: Springer International Publishing, 2020, pp. 618–646, ISBN: 978-3-030-56880-1.

[136] P. A. Grillet, *Abstract Algebra* (Graduate Texts in Mathematics, 242), eng, 2nd ed. 2007. New York, NY: Springer New York, 2007, ISBN: 0-387-71568-1.

[137] J. Groth, "On the size of pairing-based non-interactive arguments," in *Advances in Cryptology – EUROCRYPT 2016*, M. Fischlin and J.-S. Coron, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 305–326, ISBN: 978-3-662-49896-5.

[138] J. Groth and Y. Ishai, "Sub-linear zero-knowledge argument for correctness of a shuffle," in *Advances in Cryptology – EUROCRYPT 2008*, N. Smart, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 379–396, ISBN: 978-3-540-78967-3.

[139] J. Guan and M. Zhandary, "Simple schemes in the bounded storage model," in *Advances in Cryptology – EUROCRYPT 2019*, Y. Ishai and V. Rijmen, Eds., Cham: Springer International Publishing, 2019, pp. 500–524, ISBN: 978-3-030-17659-4.

[140] D. Gupta, Y. Ishai, H. K. Maji, and A. Sahai, "Secure computation from leaky correlated randomness," in *Advances in Cryptology – CRYPTO 2015*, R. Gennaro and M. Robshaw, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 701–720, ISBN: 978-3-662-48000-7.

[141] T. Gur, G. Ramnarayan, and R. D. Rothblum, "Relaxed Locally Correctable Codes," in *9th Innovations in Theoretical Computer Science Conference (ITCS 2018)*, A. R. Karlin, Ed., ser. Leibniz International Proceedings in Informatics (LIPIcs), vol. 94, Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018, 27:1–27:11, ISBN: 978-3-95977-060-6. DOI: 10.4230/LIPIcs.ITCS.2018.27. [Online]. Available: http://drops.dagstuhl.de/opus/volltexte/2018/8315.

[142] V. Guruswami and R. Li, "Polynomial time decodable codes for the binary deletion channel," *IEEE Transactions on Information Theory*, vol. 65, no. 4, pp. 2171–2178, 2019. DOI: 10.1109/TIT.2018.2876861.

[143] V. Guruswami and C. Wang, "Deletion codes in the high-noise and high-rate regimes," *IEEE Transactions on Information Theory*, vol. 63, no. 4, pp. 1961–1970, 2017. DOI: 10.1109/TIT.2017.2659765.

[144] V. Guruswami, B. Haeupler, and A. Shahrasbi, "Optimally resilient codes for list-decoding from insertions and deletions," in *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, ser. STOC 2020, Chicago, IL, USA: Association for Computing Machinery, 2020, pp. 524–537, ISBN: 9781450369794. DOI: 10.1145/3357713.3384262. [Online]. Available: https://doi.org/10.1145/3357713.3384262.

[145] V. Guruswami and R. Li, "Coding against deletions in oblivious and online models," in *Proceedings of the 2018 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 625–643. DOI: 10.1137/1.9781611975031.41. eprint: https://epubs.siam.org/doi/pdf/10.1137/1.9781611975031.41. [Online]. Available: https://epubs.siam.org/doi/abs/10.1137/1.9781611975031.41.

[146] V. Guruswami and A. Smith, "Optimal rate code constructions for computationally simple channels," *J. ACM*, vol. 63, no. 4, Sep. 2016, ISSN: 0004-5411. DOI: 10.1145/2936015. [Online]. Available: https://doi.org/10.1145/2936015.

[147] V. Guruswami, C. Umans, and S. Vadhan, "Unbalanced expanders and randomness extractors from parvaresh–vardy codes," *J. ACM*, vol. 56, no. 4, Jul. 2009, ISSN: 0004-5411. DOI: 10.1145/1538902.1538904.

[148] B. Haeupler, "Optimal document exchange and new codes for insertions and deletions," in *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, 2019, pp. 334–347. DOI: 10.1109/FOCS.2019.00029.

[149] B. Haeupler, A. Rubinstein, and A. Shahrasbi, "Near-linear time insertion-deletion codes and (1+ϵ)-approximating edit distance via indexing," in *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, M. Charikar and E. Cohen, Eds., ACM, 2019, pp. 697–708. DOI: 10.1145/3313276.3316371. [Online]. Available: https://doi.org/10.1145/3313276.3316371.

[150] B. Haeupler and A. Shahrasbi, "Synchronization strings: Codes for insertions and deletions approaching the singleton bound," in *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, ser. STOC 2017, Montreal, Canada: Association for Computing Machinery, 2017, pp. 33–46, ISBN: 9781450345286. DOI: 10.1145/3055399.3055498. [Online]. Available: https://doi.org/10.1145/3055399.3055498.

[151] B. Haeupler and A. Shahrasbi, "Synchronization strings: Explicit constructions, local decoding, and applications," in *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, ser. STOC 2018, Los Angeles, CA, USA: Association for Computing Machinery, 2018, pp. 841–854, ISBN: 9781450355599. DOI: 10.1145/3188745.3188940. [Online]. Available: https://doi.org/10.1145/3188745.3188940.

[152] B. Haeupler, A. Shahrasbi, and M. Sudan, "Synchronization Strings: List Decoding for Insertions and Deletions," in *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*, I. Chatzigiannakis, C. Kaklamanis, D. Marx, and D. Sannella, Eds., ser. Leibniz International Proceedings in Informatics (LIPIcs), vol. 107, Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018, 76:1–76:14, ISBN: 978-3-95977-076-7. DOI: 10.4230/LIPIcs.ICALP.2018.76. [Online]. Available: http://drops.dagstuhl.de/opus/volltexte/2018/9080.

[153] D. Harnik, Y. Ishai, E. Kushilevitz, and J. B. Nielsen, "Ot-combiners via secure computation," in *Theory of Cryptography*, R. Canetti, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 393–411, ISBN: 978-3-540-78524-8.

[154] D. Harnik, J. Kilian, M. Naor, O. Reingold, and A. Rosen, "On robust combiners for oblivious transfer and other primitives," in *Advances in Cryptology – EUROCRYPT 2005*, R. Cramer, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 96–113, ISBN: 978-3-540-32055-5.

[155] B. Hemenway and R. Ostrovsky, "Public-key locally-decodable codes," in *Advances in Cryptology – CRYPTO 2008*, D. Wagner, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 126–143, ISBN: 978-3-540-85174-5.

[156] B. Hemenway, R. Ostrovsky, M. J. Strauss, and M. Wootters, "Public key locally decodable codes with short keys," in *Proceedings of the 14th International Workshop and 15th International Conference on Approximation, Randomization, and Combinatorial Optimization: Algorithms and Techniques*, ser. APPROX'11/RANDOM'11, Princeton, NJ: Springer-Verlag, 2011, pp. 605–615, ISBN: 9783642229343.

[157] J. Holmgren and R. Rothblum, "Delegating computations with (almost) minimal time and space overhead," in *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, 2018, pp. 124–135. DOI: 10.1109/FOCS.2018.00021.

[158] Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai, "Extracting correlations," in *2009 50th Annual IEEE Symposium on Foundations of Computer Science*, 2009, pp. 261–270. DOI: 10.1109/FOCS.2009.56.

[159] Y. Ishai, H. K. Maji, A. Sahai, and J. Wullschleger, "Single-use ot combiners with near-optimal resilience," in *2014 IEEE International Symposium on Information Theory*, 2014, pp. 1544–1548. DOI: 10.1109/ISIT.2014.6875092.

[160] Y. Ishai, M. Prabhakaran, and A. Sahai, "Founding cryptography on oblivious transfer – efficiently," in *Advances in Cryptology – CRYPTO 2008*, D. Wagner, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 572–591, ISBN: 978-3-540-85174-5.

[161] A. Jain, H. Lin, and A. Sahai, "Indistinguishability obfuscation from well-founded assumptions," in *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*. New York, NY, USA: Association for Computing Machinery, 2021, pp. 60–73, ISBN: 9781450380539. [Online]. Available: https://doi.org/10.1145/3406325.3451093.

[162] J. Justesen, "Class of constructive asymptotically good algebraic codes," *IEEE Transactions on Information Theory*, vol. 18, no. 5, pp. 652–656, 1972. DOI: 10.1109/TIT.1972.1054893.

[163] S. Kalyanaraman and C. Umans, "On obtaining pseudorandomness from error-correcting codes," in *FSTTCS 2006: Foundations of Software Technology and Theoretical Computer Science*, S. Arun-Kumar and N. Garg, Eds., Springer Berlin Heidelberg, 2006, ISBN: 978-3-540-49995-4. DOI: .

[164] R. M. Karp and R. Kleinberg, "Noisy binary search and its applications," in *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA '07, New Orleans, Louisiana: Society for Industrial and Applied Mathematics, 2007, pp. 881–890, ISBN: 9780898716245.

[165] A. Kate, G. M. Zaverucha, and I. Goldberg, "Constant-size commitments to polynomials and their applications," in *Advances in Cryptology - ASIACRYPT 2010*, M. Abe, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 177–194, ISBN: 978-3-642-17373-8.

[166] A. Kattis, K. Panarin, and A. Vlasov, *Redshift: Transparent snarks from list polynomial commitment iops*, Cryptology ePrint Archive, Report 2019/1400, https://ia.cr/2019/1400, 2019.

[167] J. Katz, *Secure computation: When theory meets practice*, Invited talk, 2019.

[168] J. Katz, "Universally composable multi-party computation using tamper-proof hardware," in *Advances in Cryptology - EUROCRYPT 2007*, M. Naor, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 115–128, ISBN: 978-3-540-72540-4.

[169] J. Katz and Y. Lindell, *Introduction to Modern Cryptography*. USA: CRC Press, 2021, ISBN: 780815354369.

[170] J. Katz and L. Trevisan, "On the efficiency of local decoding procedures for error-correcting codes," in *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing*, ser. STOC '00, Portland, Oregon, USA: Association for Computing Machinery, 2000, pp. 80–86, ISBN: 1581131844. DOI: 10.1145/335305.335315. [Online]. Available: https://doi.org/10.1145/335305.335315.

[171] A. Kerckhoffs, "La cryptographie militaire," *Journal des Sciences Militaires*, vol. IX, pp. 5–38, Jan. 1883, A copy of this paper is available at http://www.petitcolas.net/fabien/kerckhoffs.

[172] A. Kerckhoffs, "La cryptographie militaire," *Journal des Sciences Militaires*, vol. IX, pp. 161–191, Feb. 1883, A copy of this paper is available at http://www.petitcolas.net/fabien/kerckhoffs.

[173] I. Kerenidis and R. de Wolf, "Exponential lower bound for 2-query locally decodable codes via a quantum argument," in *Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing*, ser. STOC '03, San Diego, CA, USA: Association for Computing Machinery, 2003, pp. 106–115, ISBN: 1581136749. DOI: 10.1145/780542.780560. [Online]. Available: https://doi.org/10.1145/780542.780560.

[174] J. Kilian, "A general completeness theorem for two party games," in *Proceedings of the Twenty-Third Annual ACM Symposium on Theory of Computing*, ser. STOC '91, New Orleans, Louisiana, USA: Association for Computing Machinery, 1991, pp. 553–560, ISBN: 0897913973. DOI: 10.1145/103418.103475. [Online]. Available: https://doi.org/10.1145/103418.103475.

[175] J. Kilian, "A note on efficient zero-knowledge proofs and arguments (extended abstract)," in *Proceedings of the Twenty-Fourth Annual ACM Symposium on Theory of Computing*, ser. STOC '92, Victoria, British Columbia, Canada: Association for Computing Machinery, 1992, pp. 723–732, ISBN: 0897915119. DOI: 10.1145/129712.129782. [Online]. Available: https://doi.org/10.1145/129712.129782.

[176] J. Kilian, "Founding crytpography on oblivious transfer," in *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, ser. STOC '88, Chicago, Illinois, USA: Association for Computing Machinery, 1988, pp. 20–31, ISBN: 0897912640. DOI: 10.1145/62212.62215. [Online]. Available: https://doi.org/10.1145/62212.62215.

[177] J. Kilian, "More general completeness theorems for secure two-party computation," in *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing*, ser. STOC '00, Portland, Oregon, USA: Association for Computing Machinery, 2000, pp. 316–324, ISBN: 1581131844. DOI: 10.1145/335305.335342. [Online]. Available: https://doi.org/10.1145/335305.335342.

[178] Y. Kim *et al.*, "Flipping bits in memory without accessing them: An experimental study of dram disturbance errors," in *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, 2014, pp. 361–372. DOI: 10.1109/ISCA.2014.6853210.

[179] M. Kiwi, M. Loebl, and J. Matoušek, "Expected length of the longest common subsequence for large alphabets," in *LATIN 2004: Theoretical Informatics*, M. Farach-Colton, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 302–311, ISBN: 978-3-540-24698-5.

[180] R. Klein, R. Penninger, C. Sohler, and D. P. Woodruff, "Tolerant algorithms," in *Algorithms – ESA 2011*, C. Demetrescu and M. M. Halldórsson, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 736–747, ISBN: 978-3-642-23719-5.

[181] P. Kocher *et al.*, "Spectre attacks: Exploiting speculative execution," in *2019 IEEE Symposium on Security and Privacy (SP)*, 2019, pp. 1–19. DOI: 10.1109/SP.2019.00002.

[182] S. Kopparty, O. Meir, N. Ron-Zewi, and S. Saraf, "High-rate locally correctable and locally testable codes with sub-polynomial query complexity," *J. ACM*, vol. 64, no. 2, 11:1–11:42, 2017.

[183] S. Kopparty, S. Saraf, and S. Yekhanin, "High-rate codes with sublinear-time decoding," in *STOC*, 2011, pp. 167–176.

[184] J. Lee, "Dory: Efficient, transparent arguments for generalised inner products and polynomial commitments," in *Theory of Cryptography*, K. Nissim and B. Waters, Eds., Cham: Springer International Publishing, 2021, pp. 1–34, ISBN: 978-3-030-90453-1.

[185] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions and reversals.," *Soviet Physics Doklady*, vol. 10, no. 8, pp. 707–710, 1966, Doklady Akademii Nauk SSSR, V163 No4 845-848 1965.

[186] Lindell, "Parallel coin-tossing and constant-round secure two-party computation," *Journal of Cryptology*, vol. 16, no. 3, pp. 143–184, Jun. 2003, ISSN: 1432-1378. DOI: 10.1007/s00145-002-0143-7. [Online]. Available: https://doi.org/10.1007/s00145-002-0143-7.

[187] M. Lipp *et al.*, "Meltdown: Reading kernel memory from user space," in *27th USENIX Security Symposium (USENIX Security 18)*, Baltimore, MD: USENIX Association, Aug. 2018, pp. 973–990, ISBN: 978-1-939133-04-5. [Online]. Available: https://www.usenix.org/conference/usenixsecurity18/presentation/lipp.

[188] R. J. Lipton, "A new approach to information theory," in *STACS 94*, P. Enjalbert, E. W. Mayr, and K. W. Wagner, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 1994, pp. 699–708, ISBN: 978-3-540-48332-8.

[189] R. J. Lipton, "New directions in testing," in *Distributed Computing And Cryptography, Proceedings of a DIMACS Workshop, Princeton, New Jersey, USA, October 4-6, 1989*, J. Feigenbaum and M. Merritt, Eds., ser. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 2, DIMACS/AMS, 1989, pp. 191–202. DOI: 10.1090/dimacs/002/13. [Online]. Available: https://doi.org/10.1090/dimacs/002/13.

[190] S. Liu, I. Tjuawinata, and C. Xing, *On list decoding of insertion and deletion errors*, 2020. arXiv: 1906.09705 [cs.IT].

[191] M. Luby, "Lt codes," in *The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002. Proceedings.*, 2002, pp. 271–280. DOI: 10.1109/SFCS.2002.1181950.

[192] C. Lund, L. Fortnow, H. Karloff, and N. Nisan, "Algebraic methods for interactive proof systems," in *Proceedings [1990] 31st Annual Symposium on Foundations of Computer Science*, 1990, 2–10 vol.1. DOI: 10.1109/FSCS.1990.89518.

[193] F. MacWilliams and N. Sloane, *The Theory of Error-Correcting Codes*. North Holland Publishing Co., 1983.

[194] M. Mahmoody, T. Moran, and S. Vadhan, "Time-lock puzzles in the random oracle model," in *Advances in Cryptology – CRYPTO 2011*, P. Rogaway, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 39–50, ISBN: 978-3-642-22792-9.

[195] H. K. Maji, M. Prabhakaran, and M. Rosulek, "A unified characterization of completeness and triviality for secure function evaluation," in *Progress in Cryptology - INDOCRYPT 2012*, S. Galbraith and M. Nandi, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 40–59, ISBN: 978-3-642-34931-7.

[196] G. Malavolta and S. A. K. Thyagarajan, "Homomorphic time-lock puzzles and applications," in *Advances in Cryptology – CRYPTO 2019*, A. Boldyreva and D. Micciancio, Eds., Cham: Springer International Publishing, 2019, pp. 620–649, ISBN: 978-3-030-26948-7.

[197] J. L. Massey, "Some applications of coding theory in cryptography," in *Codes and Ciphers: Cryptography and Coding IV*, 1995, pp. 33–47.

[198] J. L. Massey, "Minimal codewords and secret sharing," in *Proceedings of the 6th Joint Swedish-Russian International Workshop on Information Theory*, 1993, pp. 276–279.

[199] U. M. Maurer, "Conditionally-perfect secrecy and a provably-secure randomized cipher," *Journal of Cryptology*, vol. 5, no. 1, pp. 53–66, Jan. 1992, ISSN: 1432-1378. DOI: 10.1007/BF00191321. [Online]. Available: https://doi.org/10.1007/BF00191321.

[200] T. C. May., *Timed-release crypto*, 1993. [Online]. Available: http://www.hks.net/cpunks/cpunks-0/1460.html.

[201] R. J. McEliece, "A Public-Key Cryptosystem Based On Algebraic Coding Theory," *Deep Space Network Progress Report*, vol. 44, pp. 114–116, Jan. 1978. [Online]. Available: https://ui.adsabs.harvard.edu/abs/1978DSNPR..44..114M.

[202] R. J. McEliece and D. V. Sarwate, "On sharing secrets and reed-solomon codes," *Commun. ACM*, vol. 24, no. 9, pp. 583–584, Sep. 1981, ISSN: 0001-0782. DOI: 10.1145/358746.358762. [Online]. Available: https://doi.org/10.1145/358746.358762.

[203] R. Meier and B. Przydatek, "On robust combiners for private information retrieval and other primitives," in *Advances in Cryptology - CRYPTO 2006*, C. Dwork, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 555–569, ISBN: 978-3-540-37433-6.

[204] R. Meier, B. Przydatek, and J. Wullschleger, "Robuster combiners for oblivious transfer," in *Theory of Cryptography*, S. P. Vadhan, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 404–418, ISBN: 978-3-540-70936-7.

[205] H. Mercier, V. K. Bhargava, and V. Tarokh, "A survey of error-correcting codes for channels with symbol synchronization errors," *IEEE Communications Surveys Tutorials*, vol. 12, no. 1, pp. 87–96, 2010. DOI: 10.1109/SURV.2010.020110.00079.

[206] S. Micali, "Cs proofs," in *Proceedings 35th Annual Symposium on Foundations of Computer Science*, 1994, pp. 436–453. DOI: 10.1109/SFCS.1994.365746.

[207] S. Micali, C. Peikert, M. Sudan, and D. A. Wilson, "Optimal error correction against computationally bounded noise," in *Theory of Cryptography*, J. Kilian, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 1–16, ISBN: 978-3-540-30576-7.

[208] M. Mitzenmacher, "A survey of results for deletion channels and related synchronization channels," in *Algorithm Theory – SWAT 2008*, J. Gudmundsson, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 1–3, ISBN: 978-3-540-69903-3.

[209] T. Moran and G. Segev, "David and goliath commitments: Uc computation for asymmetric parties using tamper-proof hardware," in *Advances in Cryptology – EUROCRYPT 2008*, N. Smart, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 527–544, ISBN: 978-3-540-78.

[210] J. Naor and M. Naor, "Small-bias probability spaces: Efficient constructions and applications," in *Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing*, ser. STOC '90, Baltimore, Maryland, USA: Association for Computing Machinery, 1990, pp. 213–223, ISBN: 0897913612. DOI: 10.1145/100216.100244. [Online]. Available: https://doi.org/10.1145/100216.100244.

[211] M. Naor and B. Pinkas, "Computationally secure oblivious transfer," *Journal of Cryptology*, vol. 18, no. 1, pp. 1–35, Jan. 2005, ISSN: 1432-1378. DOI: 10.1007/s00145-004-0102-6. [Online]. Available: https://doi.org/10.1007/s00145-004-0102-6.

[212] H. Niederreiter, "Knapsack-type cryptosystems and algebraic coding theory," *Prob. Contr. Inform. Theory*, vol. 15, no. 2, pp. 157–166, 1986.

[213] R. O'Donnell, *Analysis of boolean functions*, 2021. arXiv: 2105.10386 [cs.DM].

[214] R. Ostrovsky, O. Pandey, and A. Sahai, "Private locally decodable codes," in *Automata, Languages and Programming*, L. Arge, C. Cachin, T. Jurdziński, and A. Tarlecki, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 387–398, ISBN: 978-3-540-73420-8.

[215] R. Ostrovsky and A. Paskin-Cherniavsky, "Locally decodable codes for edit distance," in *Information Theoretic Security*, A. Lehmann and S. Wolf, Eds., Cham: Springer International Publishing, 2015, pp. 236–249, ISBN: 978-3-319-17470-9.

[216] C. Papadimitriou and M. Yannakakis, "Optimization, approximation, and complexity classes," in *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, ser. STOC '88, Chicago, Illinois, USA: Association for Computing Machinery, 1988, pp. 229–234, ISBN: 0897912640. DOI: 10.1145/62212.62233. [Online]. Available: https://doi.org/10.1145/62212.62233.

[217] C. Papamanthou, E. Shi, and R. Tamassia, "Signatures of correct computation," in *Theory of Cryptography*, A. Sahai, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 222–242, ISBN: 978-3-642-36594-2.

[218] T. P. Pedersen, "Non-interactive and information-theoretic secure verifiable secret sharing," in *Advances in Cryptology — CRYPTO '91*, J. Feigenbaum, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 1992, pp. 129–140, ISBN: 978-3-540-46766-3.

[219] K. Pietrzak, "Simple Verifiable Delay Functions," in *10th Innovations in Theoretical Computer Science Conference (ITCS 2019)*, A. Blum, Ed., ser. Leibniz International Proceedings in Informatics (LIPIcs), vol. 124, Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018, 60:1–60:15, ISBN: 978-3-95977-095-8. DOI: 10.4230/LIPIcs.ITCS.2019.60. [Online]. Available: http://drops.dagstuhl.de/opus/volltexte/2018/10153.

[220] V. Pless, *Introduction to the theory of error-correcting codes*. John Wiley & Sons, 2011, vol. 48.

[221] B. Przydatek and J. Wullschleger, "Error-tolerant combiners for oblivious primitives," in *Automata, Languages and Programming*, L. Aceto, I. Damgård, L. A. Goldberg, M. M. Halldórsson, A. Ingólfsdóttir, and I. Walukiewicz, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 461–472, ISBN: 978-3-540-70583-3.

[222] M. O. Rabin, "How to exchange secrets with oblivious transfer," Harvard University, Aiken Computation Lab, Tech. Rep. TR-81, 1981.

[223] T. Rabin and M. Ben-Or, "Verifiable secret sharing and multiparty protocols with honest majority," in *Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing*, ser. STOC '89, Seattle, Washington, USA: Association for Computing Machinery, 1989, pp. 73–85, ISBN: 0897913078. DOI: 10.1145/73007.73014. [Online]. Available: https://doi.org/10.1145/73007.73014.

[224] M. Rajabzadeh Asaar, M. Salmasizadeh, and M. R. Aref, "A provably secure code-based short signature scheme and its nontransferable variant," *International Journal of Communication Systems*, vol. 31, no. 6, e3519, 2018, e3519 dac.3519. DOI: https://doi.org/10.1002/dac.3519. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/dac.3519. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/dac.3519.

[225] A. Rao, "An exposition of bourgain's 2-source extractor," *ECCCTR: Electronic Colloquium on Computational Complexity*, vol. Technical Reports, 2007.

[226] O. Reingold, G. N. Rothblum, and R. D. Rothblum, "Constant-round interactive proofs for delegating computation," in *Proceedings of the Forty-Eighth Annual ACM Symposium on Theory of Computing*, ser. STOC '16, Cambridge, MA, USA: Association for Computing Machinery, 2016, pp. 49–62, ISBN: 9781450341325. DOI: 10.1145/2897518.2897652. [Online]. Available: https://doi.org/10.1145/2897518.2897652.

[227] R. L. Rivest, A. Shamir, and D. A. Wagner, "Time-lock puzzles and timed-release crypto," USA, Tech. Rep., 1996.

[228] R. Rubinfeld and M. Sudan, "Robust characterizations of polynomials withapplications to program testing," *SIAM J. Comput.*, vol. 25, no. 2, pp. 252–271, Feb. 1996, ISSN: 0097-5397. DOI: 10.1137/S0097539793255151. [Online]. Available: https://doi.org/10.1137/S0097539793255151.

[229] C. P. Schnorr, "Efficient identification and signatures for smart cards," in *Advances in Cryptology — CRYPTO' 89 Proceedings*, G. Brassard, Ed., New York, NY: Springer New York, 1990, pp. 239–252, ISBN: 978-0-387-34805-6.

[230] L. Schulman and D. Zuckerman, "Asymptotically good codes correcting insertions, deletions, and transpositions," *IEEE Transactions on Information Theory*, vol. 45, no. 7, pp. 2552–2557, 1999. DOI: 10.1109/18.796406.

[231] S. Setty, "Spartan: Efficient and general-purpose zksnarks without trusted setup," in *Advances in Cryptology – CRYPTO 2020*, D. Micciancio and T. Ristenpart, Eds., Cham: Springer International Publishing, 2020, pp. 704–737, ISBN: 978-3-030-56877-1.

[232] S. Setty and J. Lee, *Quarks: Quadruple-efficient transparent zksnarks*, Cryptology ePrint Archive, Report 2020/1275, https://ia.cr/2020/1275, 2020.

[233] R. Shaltiel and J. Silbak, "Explicit List-Decodable Codes with Optimal Rate for Computationally Bounded Channels," in *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2016)*, 2016, 45:1–45:38. DOI: 10.4230/LIPIcs.APPROX-RANDOM.2016.45. [Online]. Available: http://drops.dagstuhl.de/opus/volltexte/2016/6668.

[234] R. Shaltiel and J. Silbak, "Explicit List-Decodable Codes with Optimal Rate for Computationally Bounded Channels," in *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2016)*, K. Jansen, C. Mathieu, J. D. P. Rolim, and C. Umans, Eds., ser. Leibniz International Proceedings in Informatics (LIPIcs), vol. 60, Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2016, 45:1–45:38, ISBN: 978-3-95977-018-7. DOI: 10.4230/LIPIcs.APPROX-RANDOM.2016.45.

[235] R. Shaltiel and C. Umans, "Simple extractors for all min-entropies and a new pseudorandom generator," *J. ACM*, vol. 52, no. 2, pp. 172–216, Mar. 2005, ISSN: 0004-5411. DOI: 10.1145/1059513.1059516. [Online]. Available: https://doi.org/10.1145/1059513.1059516.

[236] A. Shamir, "Ip=pspace (interactive proof=polynomial space)," in *Proceedings [1990] 31st Annual Symposium on Foundations of Computer Science*, 1990, 11–15 vol.1. DOI: 10.1109/FSCS.1990.89519.

[237] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, Nov. 1979, ISSN: 0001-0782. DOI: 10.1145/359168.359176. [Online]. Available: https://doi.org/10.1145/359168.359176.

[238] C. E. Shannon, "A mathematical theory of communication," *The Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, 1948. DOI: 10.1002/j.1538-7305.1948.tb01338.x.

[239] C. E. Shannon, "Communication theory of secrecy systems," *The Bell System Technical Journal*, vol. 28, no. 4, pp. 656–715, 1949. DOI: 10.1002/j.1538-7305.1949.tb00928.x.

[240] A. Ta-Shma, D. Zuckerman, and S. Safra, "Extractors from reed-muller codes," in *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, 2001, pp. 638–647. DOI: 10.1109/SFCS.2001.959940.

[241] J. Sima and J. Bruck, "Optimal k-deletion correcting codes," in *2019 IEEE International Symposium on Information Theory (ISIT)*, 2019, pp. 847–851. DOI: 10.1109/ISIT.2019.8849750.

[242] J. Sima, R. Gabrys, and J. Bruck, "Optimal codes for the q-ary deletion channel," in *2020 IEEE International Symposium on Information Theory (ISIT)*, 2020, pp. 740–745. DOI: 10.1109/ISIT44484.2020.9174241.

[243] J. Sima, R. Gabrys, and J. Bruck, "Optimal systematic t-deletion correcting codes," in *2020 IEEE International Symposium on Information Theory (ISIT)*, 2020, pp. 769–774. DOI: 10.1109/ISIT44484.2020.9173986.

[244] R. Singleton, "Maximum distanceq-nary codes," *IEEE Transactions on Information Theory*, vol. 10, no. 2, pp. 116–118, 1964. DOI: 10.1109/TIT.1964.1053661.

[245] N. Sloane, "On single-deletion-correcting codes," *arXiv: Combinatorics*, 2002.

[246] M. Sudan, L. Trevisan, and S. Vadhan, "Pseudorandom generators without the xor lemma," *Journal of Computer and System Sciences*, vol. 62, no. 2, pp. 236–266, 2001, ISSN: 0022-0000. DOI: https://doi.org/10.1006/jcss.2000.1730.

[247] I. Synopsys, *The heartbleed bug*, https://heartbleed.com/, Accessed: 2021-07-12, 2014.

[248] J. Thaler, "Time-optimal interactive proofs for circuit evaluation," in *Advances in Cryptology – CRYPTO 2013*, R. Canetti and J. A. Garay, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 71–89, ISBN: 978-3-642-40084-1.

[249] L. Trevisan, "Extractors and pseudorandom generators," *J. ACM*, vol. 48, no. 4, pp. 860–879, Jul. 2001, ISSN: 0004-5411. DOI: 10.1145/502090.502099. [Online]. Available: https://doi.org/10.1145/502090.502099.

[250] C. Umans, "Pseudo-random generators for all hardnesses," in *Proceedings of the Thiry-Fourth Annual ACM Symposium on Theory of Computing*, ser. STOC '02, Montreal, Quebec, Canada: Association for Computing Machinery, 2002, pp. 627–634, ISBN: 1581134959. DOI: 10.1145/509907.509997. [Online]. Available: https://doi.org/10.1145/509907.509997.

[251] P. Valiant, "Incrementally verifiable computation or proofs of knowledge imply time/space efficiency," in *Theory of Cryptography*, R. Canetti, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 1–18, ISBN: 978-3-540-78524-8.

[252] R. Varshamov, "Estimate of the number of signals in error correcting codes," *Dolk. Akad. Nauk SSSR*, vol. 117, pp. 739–741, 1957.

[253] G. S. Vernam, "Cipher printing telegraph systems for secret wire and radio telegraphic communications," *Journal of the American Institute for Electrical Engineers*, vol. 55, pp. 109–115, 1926.

[254] S. Vladut, D. Nogin, and M. Tsfasman, *Algebraic Geometric Codes: Basic Notions*. USA: American Mathematical Society, 2007, ISBN: 0821843060.

[255] R. S. Wahby, I. Tzialla, A. Shelat, J. Thaler, and M. Walfish, "Doubly-efficient zksnarks without trusted setup," in *2018 IEEE Symposium on Security and Privacy (SP)*, 2018, pp. 926–943. DOI: 10.1109/SP.2018.00060.

[256] B. Wesolowski, "Efficient verifiable delay functions," in *Advances in Cryptology – EUROCRYPT 2019*, Y. Ishai and V. Rijmen, Eds., Cham: Springer International Publishing, 2019, pp. 379–407, ISBN: 978-3-030-17659-4.

[257] A. Wigderson, *The impact of cryptographic thinking on tcs and beyond*, Workshop: Maches made in heaven: Crypgraphy and Theoretical Computer Science, Invited talk, https://www.youtube.com/watch?v=geGsrAkzLvQ, 2020.

[258] P. Wijesekera *et al.*, "The feasibility of dynamically granted permissions: Aligning mobile privacy with user preferences," in *2017 IEEE Symposium on Security and Privacy (SP)*, 2017, pp. 1077–1093. DOI: 10.1109/SP.2017.51.

[259] S. Wolf and J. Wullschleger, "Oblivious transfer is symmetric," in *Advances in Cryptology - EUROCRYPT 2006*, S. Vaudenay, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 222–232, ISBN: 978-3-540-34547-3.

[260] D. P. Woodruff, "A quadratic lower bound for three-query linear locally decodable codes over any field," in *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, M. Serna, R. Shaltiel, K. Jansen, and J. Rolim, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 766–779, ISBN: 978-3-642-15369-3.

[261] D. P. Woodruff, "New lower bounds for general locally decodable codes," *Electron. Colloquium Comput. Complex.*, vol. 14, 2007.

[262] A. C. Yao, "How to generate and exchange secrets," in *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*, 1986, pp. 162–167. DOI: 10.1109/SFCS.1986.25.

[263] A. C. Yao, "Protocols for secure computations," in *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*, 1982, pp. 160–164. DOI: 10.1109/SFCS.1982.38.

[264] S. Yekhanin, "Locally decodable codes," *Foundations and Trends® in Theoretical Computer Science*, vol. 6, no. 3, pp. 139–255, 2012, ISSN: 1551-305X. DOI: 10.1561/0400000030. [Online]. Available: http://dx.doi.org/10.1561/0400000030.

[265] S. Yekhanin, "Towards 3-query locally decodable codes of subexponential length," *J. ACM*, vol. 55, no. 1, Feb. 2008, ISSN: 0004-5411. DOI: 10.1145/1326554.1326555. [Online]. Available: https://doi.org/10.1145/1326554.1326555.

[266] J. Zhang, T. Xie, Y. Zhang, and D. Song, "Transparent polynomial delegation and its applications to zero knowledge proof," in *2020 IEEE Symposium on Security and Privacy (SP)*, 2020, pp. 859–876. DOI: 10.1109/SP40000.2020.00052.

VITA

Alexander R. Block was born in Panorama City, California, and raised in West Hills, California, both suburbs of Los Angeles, California. He went to the University of California, Irvine, earning dual Bachelors of Science degrees in Mathematics (with Honors) and Information & Computer Science (with Honors), additionally graduating with Campuswide Honors from the Campuswide Honors Collegium (formerly known as the Campuswide Honors Program). He began his time at Purdue University in August 2015 under the guidance of Professor David Gleich (initial advisor), continuing under Professor Hemanta K. Maji (former advisor) until October 2019, then finally completing his time at Purdue under Professor Jeremiah Blocki (current advisor). During his time at Purdue University, he completed the (non-thesis) Masters of Science in Computer Science, was the Vice President and President of the Computer Science Graduate Student Association (2019-2020, 2020-2022, respectively) and was extremely fortunate to visit the FACT Center at Reichman University (IDC Herzliya) in the Summer of 2019 under the guidance of Professor Alon Rosen. At the time of this writing, he will begin a joint postdoctoral research position at the University of Maryland, College Park with Professor Jonathan Katz and Georgetown University with Professor Justin Thaler in September 2022.