SECURELY COMPUTING PIECEWISE CONSTANT CODES

a new representation of boolean functions, and applications

Benjamin E. Diamond*

Coinbase

benjamin.diamond@coinbase.com

Abstract

Piecewise constant codes form an expressive and well-understood class of codes. In this work, we show that many piecewise constant codes admit exact coverings by polynomial-cardinality collections of hyperplanes. We prove that any boolean function whose "on-set" has been covered in just this manner can be evaluated by two parties with malicious security. This represents an interesting connection between covering codes, affine-linear algebra over prime fields, and secure computation. We observe that many natural boolean functions' on-sets admit expressions as piecewise constant codes (and hence can be computed securely). Our protocol supports secure computation on *committed* inputs; we describe applications in blockchains and credentials. We finally present an efficient implementation of our protocol.

1 Introduction

Piecewise constant codes, introduced in Cohen, Lobstein, and Sloane [CLS86], are a large and tractable class of codes, with well-understood covering radii. Many of the smallest known $(n, K)R$-covering codes—that is, the smallest subsets $S \subset \{0,1\}^n$ the radius-R Hamming balls around whose elements nonetheless cover the cube—arise from piecewise constant constructions. Indeed, the $(5, 7)1$-code of [CLS86, Fig. 2], the $(11, 192)1$-code of [CLS86, Fig. 7], and the infinite family of $(2R + 4, 12)R$ codes of [CLS86, Fig. 6] (for example) all represent our current best-known upper-bounds; some of these—though far from all—are known to be tight.

Piecewise constant codes have a rich combinatorial structure. Each such code $S \subset \{0,1\}^n$ is expressed with the aid of a positive integer partition $n = n_0 + \cdots + n_{t-1}$, and a certain multidimensional, $(n_0+1) \times \cdots \times (n_{t-1} + 1)$-sized integer array; the piecewise constant codes S (with respect to this particular partition) are identified exactly by "filling in" certain cells within this array. Crucially, the covering radius of the resulting code can be straightforwardly read off from the placement of the filled cells (this is much more feasible in practice than determining it "directly").

In this paper, we show that many piecewise constant codes can be computed securely, and discuss applications in cryptography. To do this, we first describe a connection between piecewise constant codes and affine-linear algebra over prime fields. In particular, we show that "compact" piecewise constant codes admit expressions by means of hyperplane intersections. We proceed in the following way. We isolate a certain key structure within the multi-dimensional array associated to a piecewise constant code, which we call a "quasicube". In a central lemma (Lemma 3.8 below), we show that the subset $C \subset \{0,1\}^n$ represented by any particular quasicube can be *also* be expressed as a hyperplane "intersection pattern" $C = H \cap \{0,1\}^n$, for an appropriate hyperplane $H \subset \mathbb{F}_q^n$ over any sufficiently large prime field (in fact, it's enough that q have n bits). It follows immediately (see Theorem 3.10) that every "compact" piecewise constant code $S \subset \{0,1\}^n$—specifically, every code whose cell representation can be covered by polynomially many quasicubes (and, as a special case, every code with only polynomially many filled cells)—can be represented as a polynomial-cardinality union $S = \bigcup_{i=0}^{m-1} H_i \cap \{0,1\}^n$ of intersection patterns. We also give evidence that this characterization is tight, in the sense that subsets $S \subset \{0,1\}^n$ which *don't* admit compact representations of this form also lack efficient hyperplane representations (see Example 3.22).

*I would like to thank Jason Long and Yehuda Lindell for instructive discussions.

Piecewise constant codes are abundant. In Subsection 3.2, we show that many familiar boolean functions' on-sets are actually expressible as (compact) piecewise constant codes; that is, they satisfy the hypothesis of Theorem 3.10. Applying that theorem, we derive efficient hyperplane representations of a number of function families. These include those decided by depth-2 OR-of-ANDs circuits (Theorem 3.12), symmetric functions (Theorem 3.14), integer comparators (Example 3.19), set disjointness assessors (Example 3.13), and, of course, the indicator functions of piecewise constant codes (Example 3.20). We discuss a variety of further concrete examples (see e.g. Examples 3.15, 3.17, and 3.18). Our list, of course, is only partial.

1.1 Core cryptographic protocool

In Section 4, we present our core cryptographic protocol. We show that any boolean function $f : \{0,1\}^n \rightarrow \{0,1\}$ whose on-set has been expressed as a union $f^{-1}(1) = \bigcup_{i=0}^{m-1} H_i \cap \{0,1\}^n$ of intersection patterns can be computed by two parties with malicious security. Our protocol is efficient when the covering cardinality m is polynomial in n; specifically, it requires $O(n \cdot m)$ communication, $O(n+m)$ communication, and $O(\log m)$ rounds. Our techniques involve the use of public-key primitives of prime order q; our protocol can be securely instantiated under the DDH assumption alone.

We now sketch the rough idea of our main protocol, which we specify explicitly in Protocol 4.18. The parties P_0 and P_1, with input $\mathbf{x}_0$ and $\mathbf{x}_1$ in $\{0,1\}^{n/2}$, say, begin by additively $\mathbb{F}_q$-secret-sharing each of their input *bits* individually. Each party, for each of its input bits, sends one additive share of that bit to the opposite party in the clear, and moreover gives the other party a random homomorphic encryption of the remaining share (under an additively shared, jointly owned public key). Each party moreover proves—using a protocol of Groth and Kohlweiss [GK15, Fig. 1]—that its inputs are indeed bits.

Upon completing this initial exchange, the parties hold complementary additive shares of each of the input bits $(x_0, \ldots, x_{n-1})$ of the joint argument $\mathbf{x} \in \{0,1\}^n$; they also hold random *encryptions* of the other party's vector of shares. At this point, they may evaluate the hyperplanes $H_0, \ldots, H_{m-1}$ covering $f^{-1}(1)$, both on their plaintext shares and, simultaneously, on their ciphertexts representing the other party's shares. In this way, they obtain additive sharings of the resulting output scalars, as well as encryptions of the opposite party's shares of these outputs (which serve to "check the other party's work").

The hypothesis that $f^{-1}(1) = \bigcup_{i=0}^{m-1} H_i \cap \{0,1\}^n$, now, implies exactly that $f(\mathbf{x}) = 1$ if and only if *at least one* of the parties' now-held output sharings would yield zero *if* it were reconstructed. We observe that, to determine whether any such reconstruction to zero would take place, the parties may securely multiply their m shared outputs. Assuming that the resulting output is moreover multiplied by a further shared random scalar, its equality with zero reflects exactly whether $f(\mathbf{x}) = 1$ (and nothing more). We thus turn to the problem of secure iterated modular multiplication (featuring consistency with pre-held ciphertexts).

Interestingly, *two-argument* secure two-party multiplication has received extensive recent attention, in connection with threshold ECDSA signing; indeed, it plays a central role in the two-party signing protocol of Doerner, Kondi, Lee and shelat [DKLs18], and is also used in Lindell, Nof and Ranellucci [LNR18, § 6.1]. Specifically, the underlying multiplication protocol [DKLs18, Prot. 8]—which is private, but cannot guarantee consistency with already-held ciphertexts—yields, together with a higher-level protocol [LNR18, Prot. 4.7], an overall protocol which guarantees privacy *and* consistency with prior ciphertexts.

The fact that our parties need to conduct $\Theta(m)$ secure multiplications—as opposed to just one (or a constant number)—raises significant new challenges. We observe that the parties may multiply their m shared outputs in a tree-like manner, in $O(\log m)$ rounds, using a subtle recursive application of the multiplication procedures [DKLs18, Prot. 8] and [LNR18, Prot. 4.7]. Indeed, in each round, they may vectorize a number of multiplications, handling a single layer of the tree. On the one hand, we use an arbitrary-order variant of [DKLs18, § VI. D.] to atomically handle each full layer's underlying scalar multiplications. On the other hand, we adapt and extend [LNR18]'s two-multiplicand consistency protocols to our hierarchical setting. We observe that the product protocol [LNR18, Prot. A.3] acts asymmetrically on its arguments (and, in particular, requires that the parties hold openings only of *one* of its multiplicands). Our protocol accordingly asymmetrically treats the parties' even-indexed and odd-indexed tree nodes; in particular, after assuming by induction that openings exist for each even-indexed multiplicand, we perform the reconstruction steps [LNR18, Prot. 4.7 (2) (c) – (4) (a)] only at those adjacent leaf-pairs whose *parent* node occupies an even index in *its* layer. Our approach significantly bests the efficiency of the naïve strategy (in which reconstruction takes place at every leaf-pair). We believe that the resulting protocol, Protocol 4.8, is of independent interest.

Our techniques appear to generalize readily to the multiparty setting. Because of the additional logistical complexity which that setting imposes, we have declined to treat it explicitly.

1.2 Commitment-consistent secure computation

Interestingly, our protocol—because of its close link with public-key operations—offers commitment-consistency essentially for free. Commitment-consistent—or "committed"—two-party computation offers both malicious security *and* mutual assurance that the parties' inputs are consistent with explicit prior commitments. This capability is powerful; for example, Jarecki and Shmatikov [JS07, § 1] observe that a "secure *committed 2PC* protocol is a much more useful tool than a standard 2PC protocol", because such a protocol "makes it easy to ensure that multiple instances of these protocols are executed on consistent inputs, for example as prescribed by some larger protocol." Our protocol works with any off-the-shelf homomorphic commitment (or encryption) scheme defined over a group of prime order. We believe that our protocol is the first to offer this capability. (One may of course obtain homomorphic commitment-consistency generically, by encoding the commitment function within an explicit circuit; this task is unfeasible in practice.)

Commitment-consistency is discussed sporadically throughout the secure multiparty computation literature. Lindell and Rabin [LR17] note that an initial "input commitment" phase is "the norm in all known protocols", though mention in a footnote that "In some cases, it is more subtle and the inputs are more implicitly committed; e.g., via oblivious transfer. However, this is still input commitment." We argue, here, that the *explicitness* of the commitment makes a difference. That is, protocols which guarantee consistency against *external*, explicitly supplied commitments are significantly more powerful than those which do not. The reason is exactly that articulated by Jarecki and Shmatikov [JS07, § 1]: When a protocol solicits commitments only implicitly, and "freshly", during each execution—as opposed to drawing them from an external source—there is no way to guarantee consistency across executions (or with *any* larger protocol).

We survey possible applications of commitment-consistency below.

1.2.1 Secure computation over private account balances

A *private payment* scheme specifies a privacy-preserving representation of value, as well as a protocol by which this value may be transferred, which itself moreover guarantees both privacy (regarding amounts transferred, the identities of transactors, or both) and soundness (e.g., conservation of value). *Zerocash* [BSCG+14] and *Monero* [NMt16] are classic examples in the "UTXO model"; an "account-based" approach was developed in *Zether* [BAZB20] and *Anonymous Zether* [Dia21]. Private payment protocols work naturally with blockchains (which serve to preserve their state and effect transaction verification).

In a key illustration of the utility of its commitment-consistency, our protocol allows two parties to run secure computations over hidden monetary values enshrined within some larger, ongoing private payment protocol. This feature is perhaps especially appealing in account-based systems like Anonymous Zether, where users' holdings are "consolidated" into single accounts (as opposed to being spread throughout multiple UTXOs).

We note that many existing zero-knowledge proof constructions explicitly target committed values; examples include those of Groth–Kohlweiss [GK15], Bünz et al. [BBB+18], and Diamond [Dia21]. These latter techniques, naturally, appear routinely in blockchains, where their commitment-consistency plays a central role. Philosophically, our work extends this latter tradition to the setting of two-party computation, and promises analogous applications.

1.2.2 Secure computation over credentials

Direct anonymous attestation is a powerful and complex cryptographic paradigm, in which *platforms* are issued *credentials* by certain authorities, and may unforgeably and anonymously attest to these credentials. Each credential, more specifically, contains a secret identifier, together with a number of *attributes*; a platform can selectively disclose its credential's attributes in any given presentation. We refer to Camenisch, Drijvers, and Lehmann for a comprehensive treatment [CDL16].

It remains currently unfeasible for two holders of such credentials to securely compute over the attributes concealed within their credentials (while mutually assuring each other of consistency with these credentials). Our protocol's commitment-consistency makes this capability almost immediate.

In fact, our protocol is moreover compatible with the unlinkability property central to these schemes; more precisely, each party may couple its execution of our protocol with a standard verifiable presentation of its credential (successive such presentations can be linked only when the presenter wants them to be). For example, in the direct anonymous attestation scheme of [CDL16], a "credential" is essentially a vector of scalars, together with a "BBS+" signature over that vector (this signature can be procured even when some or all of the vector's underlying quantities are hidden). A presentation of such a credential reveals some of its underlying vector's components, and moreover proves knowledge of its associated signature. It is straightforward to attach to such a presentation further commitments, which provably contain precisely those messages which were hidden during the presentation. These latter commitments can be linked to the inputs of a secure computation, using our protocol.

1.3 Concrete efficiency

In Subsection 4.4, we describe a concrete implementation of our full protocol, in which f is specialized, for the sake of example, to a certain integer comparator function (see Example 3.19). Our protocol is practical, and runs over a WAN in about as much wall time as a private cryptocurrency transaction takes to generate (see e.g. [Dia21, § I] for an overview). Specifically, on the function $f : \{0,1\}^{64} \rightarrow \{0,1\}$ which compares two 32-bit unsigned integers, our protocol runs in about 2.5 seconds of wall time over a WAN, and requires exchanging about 1,500 kilobytes. The majority of our protocol's bandwidth burden is inherited from the multiplication subprotocol [DKLs18, Prot. 8]. We give further details in Subsection 4.4.

1.4 Prior work

Exact hyperplane coverings have appeared occasionally throughout the combinatorics literature. A classic work of Alon and Füredi studies the particular set $S = \{0,1\}^n - \{(0,\ldots,0)\}$, and shows that no fewer than n hyperplanes (over any field) can cover this set; as n hyperplanes clearly suffice, this result is tight. A recent work of Aaronson, Groenland, Grzesik, Johnston, and Kielak [AGG$^+$21] studies various questions around exact hyperplane coverings. For example, they estimate the *worst-case* number of hyperplanes required to cover any set $S \subset \{0,1\}^n$, as S ranges throughout all such sets (and n is fixed). Though they focus on hyperplanes over $\mathbb{R}$, their results largely carry over to the case of finite fields.

An important progenitor of our work appears in the form of Wagh, Gupta, and Chandran [WGC19, Alg. 3]. That protocol allows two *semi-honest* parties and a non-colluding, semi-honest third server to compare a secret-shared integer with a fixed public integer. Though they do not express it in these terms, their method actually entails covering the on-set of the fixed-threshold comparator function with affine hyperplanes, and evaluating these hyperplanes "over secret-share", before handing the resulting outputs to the third party, who reconstructs them and reports whether a zero is present. That protocol of course lacks malicious security, and requires a third party; moreover, it treats only one function.

Jarecki and Shmatikov [JS07] describe a maliciously secure, commitment-consistent two-party protocol. Their protocol works only with a Camenisch–Shoup-style commitment scheme, itself based on Paillier-like groups of unknown order. The protocol of Frederiksen, Pinkas and Yanai [FPY18] internally invokes an additively homomorphic commitment scheme (of known prime order); on the other hand, it does so only to construct *random* commitments, which are formed into multiplication triples during an initial preprocessing phase. That protocol does not obviously support the use of externally committed inputs.

2 Definitions and Notation

By the "natural numbers", represented by the symbol $\mathbb{N}$, we shall mean the *positive integers*. That a number q has n *bits* means that it resides in $\{2^{n-1}, \ldots, 2^n - 1\}$. Bertrand's postulate, a corollary of a weak form of the prime number theorem, implies that primes $q \in \{2^{n-1}, \ldots, 2^n - 1\}$ necessarily exist for each n (see e.g. Montgomery and Vaughan [MV06, § 2.2]).

2.1 Linear and affine algebra

We refer to Cohn [Coh82] for preliminaries on algebra.

We write q for an *odd* prime, and $\mathbb{F}_q$ for the finite field of order q (see e.g. [Coh82, § 6.3]). We have the standard notions of *vector spaces* over $\mathbb{F}_q$ (see e.g. [Coh82, § 4.1]) and of $\mathbb{F}_q$-homomorphisms, or *maps*, between vector spaces (see [Coh82, § 4.2]). A *hyperplane* is an affine-linear functional $H : \mathbb{F}_q^n \to \mathbb{F}_q$ (see Cohn [Coh82, § 4.8]). Each hyperplane admits an expression $H : (x_0, \ldots, x_{n-1}) \mapsto \alpha.x_0 + \cdots + \alpha_{n-1} \cdot x_{n-1} + \alpha$, for appropriate field elements $\alpha_0, \ldots, \alpha_{n-1}, \alpha$ in $\mathbb{F}_q$. We often identify hyperplanes H with their nullsets $\left\{ \mathbf{x} \in \mathbb{F}_q^n \;\middle|\; H(\mathbf{x}) = 0 \right\} \subset \mathbb{F}_q^n$.

By an *intersection pattern over* $\mathbb{F}_q$, or an $\mathbb{F}_q$*-intersection pattern*, we will mean a (possibly empty) set $S \subset \{0,1\}^n$ of the form $S = H \cap \{0,1\}^n$ for some hyperplane $H \subset \mathbb{F}_q^n$ (see e.g. [AGG$^+$21, p. 5]).

The following basic result occasionally allows us to replace affine-linear algebra with linear algebra:

Lemma 2.1. *For each* $\mathbf{x} \in \{0,1\}^n$*, there exists an invertible affine* $\mathbb{F}_q$*-linear map* $o_\mathbf{x} : \mathbb{F}_q^n \to \mathbb{F}_q^n$ *which maps* $\{0,1\}^n$ *to itself, and which sends* $\mathbf{x}$ *to the origin.*

Proof. We write the coordinates of $\mathbf{x}$ as $(x_0, \ldots, x_{n-1})$. The map $o_\mathbf{x}$ defined on $\mathbf{y} = (y_0, \ldots, y_{n-1}) \in \mathbb{F}_q^n$ by:

$$o_\mathbf{x}(\mathbf{y}) := \left(\begin{cases} 1 - y_i & \text{if } x_i = 1 \\ y_i & \text{if } x_i = 0 \end{cases} \right)_{i=0}^{n-1}$$

clearly satisfies the desired properties. □

We work in the *arithmetic* complexity model, in which each field operation takes constant time (see for example von zur Gathen and Gerhard [vzGG13, § 2]).

2.2 Boolean function complexity

We refer to Wegener [Weg87] and Vollmer [Vol99] for facts about the complexity of boolean functions. For each natural number k, $\mathbf{\Sigma_k}$ and $\mathbf{\Pi_k}$ denote the function families decided by polynomially-sized, unbounded fan-in, layered circuits with an OR or an AND gate at the top (respectively) and k alternating layers of gates subsequently, and with negations only applied to the inputs (see [Weg87, § 11 Def. 1.1]). The class $\mathbf{AC}^0$ is defined as $\bigcup_{i=0}^{\infty} \mathbf{\Sigma_k} \cup \mathbf{\Pi_k}$ (see [Vol99, Def. 4.5]). The class $\mathbf{NC}^1$ denotes the class of function families decided by polynomially-sized, bounded fan-in, $O(\log n)$-depth circuits (see [Vol99, Def. 4.1]). The fact that unbounded fan-in gates can be converted into log-depth trees of bounded fan-in gates implies that $\mathbf{AC}^0 \subset \mathbf{NC}^1$ (see [Vol99, Prop. 1.17]).

2.3 Coding theory

We refer to Cohen, Honkala, Litsyn, and Lobstein [CHLL97] for preliminaries on covering codes. A *code* is a subset $S \subset \{0,1\}^n$. A *subcube* is a set of the form $C = \left\{ (x_0, \ldots, x_{n-1}) \in \{0,1\}^n \;\middle|\; \bigwedge_{i=0}^{k-1} x_{c_i} = y_i \right\}$, where $\{c_0, \ldots, c_{k-1}\} \subset \{0, \ldots, n-1\}$ is a subsequence and $y_0, \ldots, y_{k-1}$ are binary constants. The *Hamming distance* between elements $\mathbf{x} = (x_0, \ldots, x_{n-1})$ and $\mathbf{y} = (y_0, \ldots, y_{n-1})$ of $\{0,1\}^n$ is $d(\mathbf{x}, \mathbf{y}) := |\{i \in \{0, \ldots, n-1\} \mid x_i \neq y_i\}|$. The *weight* of an element $\mathbf{x} \in \{0,1\}^n$ is $w(\mathbf{x}) := d(\mathbf{x}, \mathbf{0})$. The radius-$r$ *Hamming ball* around a point $\mathbf{x} \in \{0,1\}^n$ is the set $B_r(\mathbf{x}) = \{\mathbf{y} \in \{0,1\}^n \mid d(\mathbf{x}, \mathbf{y}) \leq r\}$. A code $S \subset \{0,1\}^n$ is an r*-covering code* if $\bigcup_{\mathbf{x} \in C} B_r(\mathbf{x}) = \{0,1\}^n$. A code $S \subset \{0,1\}^n$'s *covering radius* R is the smallest r for which it's an r-covering code. An $(n, K)R$ *code* is a K-element code $S \subset \{0,1\}^n$ with covering radius R.

Piecewise constant codes were introduced in Cohen, Lobstein and Sloane [CLS86], and are further discussed in [CHLL97, § 3.3]; we recall their definition here. By a *partition* of a natural number n, we shall mean a partition of the set $\{0, \ldots, n-1\}$ into nonempty subsets. We define the *refinement* relation on partitions in the obvious way. We slightly abuse notation by referring to partitions only by the sizes of their constituent subsets; that is, we describe any given partition of n using the notation $n = n_0 + \cdots + n_{t-1}$, identifying all partitions which differ by a permutation of $\{0, \ldots, n-1\}$ (this latter notation matches the classical number-theoretic notion of *partition*). Given a natural number n and a partition $n = n_0 + \ldots + n_{t-1}$, we correspondingly split each element $\mathbf{x} \in \{0,1\}^n$ into segments $\mathbf{x}_0 \parallel \cdots \parallel \mathbf{x}_{t-1}$ of appropriate lengths.

Definition 2.2. $S \subset \{0,1\}^n$ is *piecewise constant with respect to the partition* $n = \sum_{i=0}^{t-1} n_i$ if, provided S contains any word $\mathbf{x}_0 \parallel \cdots \parallel \mathbf{x}_{t-1}$ with $w(\mathbf{x}_0) = w_0, \ldots, w(\mathbf{x}_{t-1}) = w_{t-1}$, then S contains all such words.

Each piecewise constant code $S \subset \{0,1\}^n$—with respect to the partition $n = n_0 + \ldots + n_{t-1}$—say, can be represented with the aid of a certain $(n_0+1) \times \cdots \times (n_{t-1}+1)$ multidimensional array, some of whose cells are "filled in" (see e.g. [CLS86, Fig.s 3.1 and 3.2]). Indeed, each multi-index $(w_0, \ldots, w_{t-1}) \in \prod_{i=0}^{t-1}\{0, \ldots, n_i\}$ in the array represents exactly those words $\mathbf{x}_0 \parallel \cdots \parallel \mathbf{x}_{t-1} \in \{0,1\}^n$ satisfying $w(\mathbf{x}_0) = w_0, \ldots, w(\mathbf{x}_{t-1}) = w_{t-1}$.

Definition 2.3. S's *cell representation* is the subset $\overline{S} \subset \prod_{i=0}^{t-1}\{0, \ldots, n_i\}$ consisting of those multi-indices $(w_0, \ldots, w_{t-1})$ for which S contains any, and hence every, word $\mathbf{x}_0 \parallel \cdots \parallel \mathbf{x}_{t-1} \in \{0,1\}^n$ with $\bigwedge_{i=0}^{t-1} w(\mathbf{x}_i) = w_i$.

Each cell $(w_0, \ldots, w_{t-1}) \in \prod_{i=0}^{t-1}\{0, \ldots, n_i\}$ represents exactly $\prod_{i=0}^{t-1}\binom{n_i}{w_i}$ words in $\{0,1\}^n$. The cardinality of S is thus $\sum_{(w_i)_{i=0}^{t-1} \in \overline{S}} \prod_{i=0}^{t-1}\binom{n_i}{w_i}$. The covering radius of a piecewise constant code $S \subset \{0,1\}^n$ is exactly the "covering radius" of $\overline{S} \subset \prod_{i=0}^{t-1}\{0, \ldots, n_i\}$, where the latter space is given the "Manhattan distance" (we refer to [CHLL97, § 3.3] for details). It is often computationally feasible to determine this latter radius.

It is sometimes convenient to go in the "opposite direction". We record the following definition here:

Definition 2.4. Fix a partition $n = n_0 + \cdots + n_{t-1}$ and an arbitrary subset $\overline{C} \subset \prod_{i=0}^{t-1}\{0, \ldots, n_i\}$. The *pullback* $C \subset \{0,1\}^n$ of $\overline{C}$ is defined by $C := \left\{\mathbf{x}_0 \parallel \cdots \parallel \mathbf{x}_{t-1} \in \{0,1\} \mid (w(\mathbf{x}_0), \ldots, w(\mathbf{x}_{t-1})) \in \overline{C}\right\}$.

Informally, $C \subset \{0,1\}^n$ is the union, over those cells $(w_0, \ldots, w_{t-1}) \in \overline{C}$, of the codewords $\mathbf{x} \in \{0,1\}^n$ represented by $(w_0, \ldots, w_{t-1})$.

2.4 Basic security definitions

We give basic security definitions, following Katz and Lindell [KL21]. In experiment-based games involving an adversary $\mathcal{A}$, we occasionally use the notation $\mathcal{A}(\mathsf{E}_{\mathcal{A}}(\lambda))$ to denote the *output* of $\mathcal{A}$ within the game $\mathsf{E}_{\mathcal{A}}(\lambda)$ (as distinguished from whether $\mathcal{A}$ wins the experiment).

Two distribution ensembles $\{X_0(a,\lambda)\}_{a\in\{0,1\}^*;\lambda\in\mathbb{N}}$ and $\{X_1(a,\lambda)\}_{a\in\{0,1\}^*;\lambda\in\mathbb{N}}$ are *computationally indistinguishable* (see [KL21, § 8.8] and [Lin17, § 6.2]) if, for each nonuniform PPT distinguisher D, there is a negligible function μ for which, for each $a \in \{0,1\}^*$ and $\lambda \in \mathbb{N}$,

$$|\Pr[D(X_0(a,\lambda)) = 1] - \Pr[D(X_1(a,\lambda)) = 1]| \le \mu(\lambda).$$

The distributions $\{X_0(a,\lambda)\}_{a\in\{0,1\}^*;\lambda\in\mathbb{N}}$ and $\{X_1(a,\lambda)\}_{a\in\{0,1\}^*;\lambda\in\mathbb{N}}$ are *statistically indistinguishable* if there is a negligible function μ for which, for each $a \in \{0,1\}^*$ and $\lambda \in \mathbb{N}$,

$$\sum_{v\in\{0,1\}^*} |\Pr[X_0(a,\lambda) = v] - \Pr[X_1(a,\lambda) = v]| \le \mu(\lambda).$$

Statistical indistinguishability implies computational indistinguishability.

We recall the definition of a *group-generation algorithm* $\mathcal{G}$, which, on input 1^λ, outputs a cyclic group $\mathbb{G}$, its prime order q (with bit-length λ), and a generator $g \in \mathbb{G}$ (see [KL21, § 9.3.2]). We recall the notions whereby the *discrete logarithm problem is hard relative to* $\mathcal{G}$ (see [KL21, Def. 9.63]) and the *decisional Diffie–Hellman problem is hard relative to* $\mathcal{G}$ (see [KL21, Def. 9.64]).

An *encryption scheme* is a triple of algorithms $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$; given a keypair $(pk, sk) \leftarrow \mathsf{Gen}(1^\lambda)$ and a message m, we have an encryption procedure $A \leftarrow \mathsf{Enc}_{pk}(m; r)$ and a decryption $m := \mathsf{Dec}_{sk}(A)$ (see [KL21, Def. 12.1] for more details). We define the security of encryption schemes, following [KL21, Def. 12.5]:

Definition 2.5. The *multiple encryptions experiment* $\mathsf{PubK}_{\Pi,\mathcal{A}}^{\mathsf{LR-cpa}}(\lambda)$ is defined as:

1. A keypair $(pk, sk) \leftarrow \mathsf{Gen}(1^\lambda)$ is generated and a uniform bit $b \in \{0,1\}$ is chosen.
2. The adversary $\mathcal{A}$ is given pk and oracle access to $\mathsf{LR}_{pk,b}(\cdot,\cdot)$.
3. $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$.
4. The output of the experiment is defined to be 1 if and only if $b = b'$.

We say that $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ has *indistinguishable multiple encryptions* if, for each nonuniform PPT adversary $\mathcal{A}$, there exists a negligible function μ for which $\Pr[\mathsf{PubK}_{\Pi,\mathcal{A}}^{\mathsf{LR-cpa}}(\lambda) = 1] \le \frac{1}{2} + \mu(\lambda)$.

An encryption scheme is $\mathbb{F}_q$-*homomorphic*, where q is prime, if its key-space is an order-q group, and, for each key pk, the encryption function $(m; r) \mapsto \mathsf{Enc}_{pk}(m; r)$ is an $\mathbb{F}_q$-vector space homomorphism.

Example 2.6. Given a group $(\mathbb{G}, q, g) \leftarrow \mathcal{G}$, we have the resulting El Gamal encryption scheme Π (see [KL21, Cons. 12.16]), which is $\mathbb{F}_q$-homomorphic. If the decisional Diffie–Hellman problem is hard relative to $\mathcal{G}$, then Π has indistinguishable multiple encryptions (see [KL21, Thm. 12.6 and Thm. 12.18]).

A *commitment scheme* is a pair of probabilistic algorithms $(\mathsf{Gen}, \mathsf{Com})$; given public parameters $\mathsf{params} \leftarrow \mathrm{Gen}(1^\lambda)$ and a message m, we have the *commitment* $A := \mathsf{Com}_{\mathsf{params}}(m; r)$, as well as a decommitment procedure effected by sending m and r (see [KL21, § 6.6.5] for more details). We recall the notions whereby a commitment scheme is *hiding* and *binding* (see [KL21, Def. 6.13]). A commitment scheme is $\mathbb{F}_q$-*homomorphic* if, for each params, its commitment function $(m; r) \mapsto \mathsf{Com}_{\mathsf{params}}(m; r)$ is an $\mathbb{F}_q$-vector space homomorphism.

2.5 Secure two-party computation

We record security definitions for secure two-party computation. Our setting is essentially that of Lindell [Lin17, § 6.6.2]; we recall the details here mainly for self-containedness. We have the notions of *functionalities* $\mathcal{F}$ and *protocols* Π. In our two-party setting, a *round* consists of a single message sent from one party to the other. We adopt a space-saving device whereby we stipulate in advance that throughout the paper, if, during any protocol, any hybrid sub-functionality returns a failure value to any honest party at any time, that party immediately aborts. We also omit mention of such things as session identifiers when possible.

We recall the general definition of maliciously secure two-party computation (see [Lin17, § 6.6.1]):

Definition 2.7. We fix a functionality $\mathcal{F}$, a protocol Π, a real-world adversary $\mathcal{A}$, a simulator $\mathcal{S}$, and a corrupt party $C \in \{0, 1\}$. We consider the distributions:

- $\mathsf{Real}_{\Pi,\mathcal{A},C}(\mathbf{x}_0, \mathbf{x}_1, \lambda)$: Generate a run of Π with security parameter λ, in which the honest party P_{1-C} uses the input $\mathbf{x}_{1-C}$ and $\mathcal{A}$ controls P_C's messages. Return the outputs of $\mathcal{A}$ and P_{1-C}.
- $\mathsf{Ideal}_{\mathcal{F},\mathcal{S},C}(\mathbf{x}_0, \mathbf{x}_1, \lambda)$: Run $S(1^\lambda, C, \mathbf{x}_C)$ until it outputs $\mathbf{x}'_C$, or else outputs $(\texttt{abort})$ to $\mathcal{F}$, who halts. Give $\mathbf{x}'_C$ and $\mathbf{x}_{1-C}$ to $\mathcal{F}$, and obtain outputs v_0 and v_1. Give v_C to $\mathcal{S}$; if $\mathcal{S}$ outputs $(\texttt{abort})$, then $\mathcal{F}$ outputs $(\texttt{abort})$ to P_{1-C}; otherwise, $\mathcal{F}$ gives v_{1-C} to P_{1-C}. Return the outputs of $\mathcal{S}$ and P_{1-C}.

We say that Π *securely computes* $\mathcal{F}$ *in the presence of one static malicious corruption with abort*, or that Π *securely computes* $\mathcal{F}$, if, for each corrupt party $C \in \{0, 1\}$ and real-world nonuniform PPT adversary $\mathcal{A}$ corrupting C, there is an expected polynomial-time simulator $\mathcal{S}$ corrupting C in the ideal world such that

$$\left\{\mathsf{Real}_{\Pi,\mathcal{A},C}(\mathbf{x}_0, \mathbf{x}_1, \lambda)\right\}_{\mathbf{x}_0,\mathbf{x}_1,\lambda} \stackrel{c}{\equiv} \left\{\mathsf{Ideal}_{\mathcal{F},\mathcal{S},C}(\mathbf{x}_0, \mathbf{x}_1, \lambda)\right\}_{\mathbf{x}_0,\mathbf{x}_1,\lambda},$$

where the elements $\mathbf{x}_0$ and $\mathbf{x}_1$ of $\{0, 1\}^*$ are required to have equal lengths.

Our most important functionality captures the *commitment-consistent* computation of some fixed boolean function $f : \{0, 1\}^n \to \{0, 1\}$. We adopt the convention whereby P_0 "owns" the even-indexed inputs and P_1 "owns" the odd-indexed inputs.

FUNCTIONALITY 2.8 ($\mathcal{F}_f$—main functionality)**.**
The functionality works with players P_0 and P_1, and a function $f : \{0, 1\}^n \to \{0, 1\}$, where n is even.

- Upon receiving $(\texttt{commit}; \mathbf{x}_\nu)$, from P_ν, where $\mathbf{x}_\nu \in \{0, 1\}^{n/2}$, $\mathcal{F}_f$ sends $(\texttt{received})$ to $P_{1-\nu}$.
- Upon receiving $(\texttt{evaluate})$ from both parties, $\mathcal{F}_f$ interleaves $\mathbf{x}_0$ and $\mathbf{x}_1$ to obtain the input $\mathbf{x} \in \{0, 1\}^n$, evaluates $v := f(\mathbf{x})$, and outputs $(\texttt{evaluate}, v)$ to both P_0 and P_1.

2.6 Zero-knowledge proofs

We present definitions for zero-knowledge proofs, following the monograph of Hazay and Lindell [HL10, § 6].

We fix a binary relation $R \subset \{0, 1\}^* \times \{0, 1\}^*$, whose elements (x, w) satisfy $|w| = \mathsf{poly}(|x|)$ for some polynomial poly. If $(x, w) \in R$, we call x a *statement* and w its *witness*. The *zero-knowledge proof of knowledge ideal functionality*, or *ZKPOK functionality*, works as follows:

FUNCTIONALITY 2.9 ($\mathcal{F}_{\mathsf{zk}}^R$—ZKPOK ideal functionality for the relation R).
A relation R is fixed.

- Upon receiving a message of the form $(\texttt{prove}, x; w)$, $\mathcal{F}_{\mathsf{zk}}^R$ stores $(\texttt{prove}, x, R(x, w))$ in memory.
- Upon receiving a message of the form $(\texttt{verify}, x)$, $\mathcal{F}_{\mathsf{zk}}^R$ checks whether $(\texttt{prove}, x, R(x, w))$ is in memory. If it is, $\mathcal{F}_{\mathsf{zk}}^R$ returns $(\texttt{verify}, R(x, w))$; otherwise, $\mathcal{F}_{\mathsf{zk}}^R$ returns $(\texttt{verify}, 0)$.

This functionality appears in e.g. [HL10, § 6.5.3], though we use a slightly nonstandard syntax.

The ZKPOK ideal functionality can be instantiated with the aid of so-called Σ-protocols. We begin with the following abstract three-move protocol template (see [HL10, Prot. 6.2.1]):

PROTOCOL 2.10 (General three-move protocol template for the relation R).
$\mathcal{P}$ and $\mathcal{V}$ both have a statement x. $\mathcal{P}$ has a witness w such that $(x, w) \in R$.

1: $\mathcal{P}$ sends an initial message a to the verifier V.
2: $\mathcal{V}$ sends a random λ-bit string e to q.
3: $\mathcal{P}$ sends a reply z.
4: $\mathcal{V}$ chooses to *accept* or *reject* based only on the data (x, a, e, z).

We have the formal notion of *Σ-protocols* [HL10, Def. 6.2.2], which we reproduce here:

Definition 2.11. A protocol Π of the form Protocol 2.10 is said to be a *Σ-protocol* for the relation R if the following conditions hold:

- **Completeness.** If $\mathcal{P}$ and $\mathcal{V}$ follow the protocol on inputs (x, w) and x, respectively, where $(x, w) \in R$, then $\mathcal{V}$ always accepts.
- **Special soundness.** There exists a polynomial-time extractor X which, given any x and accepting transcripts (a, e, z) and (a, e', z') on x for which $e \neq e'$, outputs a witness w for which $(x, w) \in R$.
- **Honest verifier zero knowledge.** There exists a polynomial-time simulator M which, on inputs λ and x, outputs a random transcript (a, e, z) distributed exactly as in an interaction between $\mathcal{P}$ and $\mathcal{V}$.

We recall the random oracle model and the Fiat–Shamir transform (see e.g. [KL21, Cons. 13.9]). In order to make a protocol Π of the form of Protocol 2.10 non-interactive, $\mathcal{P}$ and $\mathcal{V}$ proceed in the following way. $\mathcal{P}$ submits the initial message a to the random oracle, and obtains a challenge e; the proof consists of (a, e, z). When verifying the proof, $\mathcal{V}$ recomputes e from a using a second oracle query.

Σ-protocols made non-interactive in this way securely instantiate the ZKPOK ideal functionality:

Theorem 2.12. *Fix a relation R and a Σ-protocol Π for R. The non-interactive protocol obtained upon applying the Fiat–Shamir transform to Π securely instantiates the ideal ZKPOK functionality $\mathcal{F}_{\mathsf{zk}}^R$.*

Proof. Though this theorem is essentially accepted folklore, it is not easy to prove, and it is difficult to find a complete proof. The theorem essentially follows from a combination of the ideas of Pointcheval and Stern [PS00, Thm. 1] and Hazay and Lindell [HL10, Thm. 6.5.6]. □

Using a generalized version of the Schnorr protocol, we obtain Σ-protocols for a number of important relations. We fix an $\mathbb{F}_q$-vector space homomorphism $\phi : \mathbb{G}_0 \to \mathbb{G}_1$, and the corresponding preimage relation:

$$R_\phi = \{(h; g) \mid \phi(g) = h\}.$$

We have the protocol:

PROTOCOL 2.13 (Generalized Σ-protocol Π_ϕ for R_ϕ).
$\mathcal{P}$ and $\mathcal{V}$ both have $\phi : \mathbb{G}_0 \to \mathbb{G}_1$ and an element $h \in \mathbb{G}_1$. $\mathcal{P}$ has an element $g \in \mathbb{G}_0$ such that $\phi(g) = h$.

1: $\mathcal{P}$ randomly samples $r \leftarrow \mathbb{G}_0$, and sends $\mathcal{V}$ the image $a := \phi(r)$.
2: $\mathcal{V}$ samples $e \leftarrow \mathbb{F}_q$ and sends e to $\mathcal{P}$.
3: $\mathcal{P}$ sets $z := r + c \cdot G$ and sends z to $\mathcal{V}$.

4: $\mathcal{V}$ accepts iff $f(z) \stackrel{?}{=} a + c \cdot H$.

Theorem 2.14. *The protocol Π_ϕ is a Σ-protocol for the relation R_ϕ.*

Proof. This is essentially proven in [HL10, §§ 6.1–6.2]; though that proof targets the Schnorr protocol, the proof is identical in the more general setting. □

We record applications of Theorem 2.14 here; we will use these below.

Example 2.15. The classic Schnorr protocol (see e.g. [KL21, Fig. 13.2]) specializes Protocol 2.13 to the map $\phi : \mathbb{F}_q \to \mathbb{G}$ sending $\phi : x \mapsto x \cdot g$, for some group $(\mathbb{G}, q, g) \leftarrow \mathcal{G}(1^\lambda)$.

Example 2.16. A well-known technique proves that a homomorphic ciphertext A encrypts 0 under the public key pk; this protocol specializes to a "proof of Diffie–Hellman tuple" under the El Gamal scheme. This latter protocol appears in e.g. [HL10, Prot. 6.2.4] and [LNR18, § 3.3 (2)]. We record the relation here:

$$R_{\mathsf{DH}} = \{(pk, A; r) \mid A = \mathsf{Enc}_{pk}(0; r)\}.$$

This protocol arises upon specializing Protocol 2.13 to the map $\phi : r \mapsto \mathsf{Enc}_{pk}(0; r)$; using Theorems 2.12 and 2.14, we obtain a secure instantiation of the corresponding ideal functionality, which we call $\mathcal{F}_{\mathsf{zk}}^{\mathsf{DH}}$.

Example 2.17. A similar protocol can be used to prove knowledge of the message and randomness of an El Gamal ciphertext; this relation appears in [LNR18, § 3.3 (4)]. We reproduce it here:

$$R_{\mathsf{EG}} = \{(pk, A; m, r) \mid A = \mathsf{Enc}_{pk}(m; r)\}.$$

To securely instantiate $\mathcal{F}_{\mathsf{zk}}^{\mathsf{EG}}$, we define $\phi : (m, r) \mapsto \mathsf{Enc}_{pk}(m; r)$, and apply Theorems 2.12 and 2.14.

Example 2.18. A further related protocol shows that two ciphertexts are related by a re-randomization operation, and that, in particular, one encrypts 0 if and only if the other does (and moreover is random subject to this condition). This protocol appears in [LNR18, § 3.3 (2)]. We have the relation:

$$R_{\mathsf{RE}} = \{(pk, A_0, A_1; s, r) \mid A_1 = s \cdot A_0 + \mathsf{Enc}_{pk}(0; r)\}.$$

One may securely instantiate $\mathcal{F}_{\mathsf{zk}}^{\mathsf{RE}}$ by setting $\phi : (s, r) \mapsto s \cdot A_0 + \mathsf{Enc}_{pk}(0; r)$.

Example 2.19. Protocol 2.13 can be used to prove that two ciphertexts encrypt the same message. We have the relation:

$$R_{\mathsf{EqMsg}} = \{(\mathsf{params}, pk_0, pk_1, A_0, A_1; m, r_0, r_1) \mid A_0 = \mathsf{Enc}_{pk_0}(m; r_0) \wedge A_1 = \mathsf{Enc}_{pk_1}(m; r_1)\}.$$

We obtain a Σ-protocol for R_{EqMsg} upon specializing Protocol 2.13 to the map $\phi : (m, r_0, r_1) \mapsto (\mathsf{Enc}_{pk_0}(m; r_0), \mathsf{Enc}_{pk_1}(m; r_1))$. This technique appears in [FMMO19, § 6.1]. Applying Theorem 2.12, we obtain a secure instantiation of the corresponding ideal functionality $\mathcal{F}_{\mathsf{zk}}^{\mathsf{EqMsg}}$.

Example 2.20. Using an almost identical technique, we obtain a proof that a commitment and a ciphertext "contain" the same message. We have the relation:

$$R_{\mathsf{ComMsg}} = \{(\mathsf{params}, pk, A_0, A_1; m, r_0, r_1) \mid A_0 = \mathsf{Com}_{\mathsf{params}}(m; r_0) \wedge A_1 = \mathsf{Enc}_{pk}(m; r_1)\}.$$

We obtain a Σ-protocol for R_{ComMsg} from Protocol 2.13 and $\phi : (m, r_0, r_1) \mapsto (\mathsf{Com}_{\mathsf{params}}(m; r_0), \mathsf{Enc}_{pk}(m; r_1))$.

Example 2.21. The relation R_{Prod} of [LNR18, § 3.3 (5)] allows a prover to demonstrate that a particular El Gamal ciphertext equals the (re-randomized) *scalar multiple* of one ciphertext by the *message* of a further ciphertext. We recall the relation here:

$$R_{\mathsf{Prod}} = \{(pk, A, A_0, A_1; m, r_0, r_1) \mid A = m \cdot A_0 + \mathsf{Enc}_{pk}(0; r_0) \wedge A_1 = \mathsf{Enc}_{pk}(m; r_1)\}.$$

The relation R_{Prod}—and a corresponding Σ-protocol for it—also arises as a specialization of R_ϕ above; indeed, it's enough to specialize Π_ϕ to the map $\phi : (m, r_0, r_1) \mapsto (m \cdot A_0 + \mathsf{Enc}_{pk}(0; r_0), \mathsf{Enc}_{pk}(m; r_1))$. Theorem 2.12 yields a secure instantiation of the resulting ideal functionality $\mathcal{F}_{\mathsf{zk}}^{\mathsf{Prod}}$.

The following Σ-protocol does *not* arise as a specialization of Protocol 2.13.

Example 2.22. A "bit-commitment" proof shows that a public ciphertext A contains a bit. More precisely:

$$R_{\mathsf{BitProof}} = \{(pk, A; m, r) \mid A = \mathsf{Enc}_{pk}(m; r) \wedge m \in \{0, 1\}\}.$$

We write Π_{BitProof} for the protocol [GK15, Fig. 1] of Groth and Kohlweiss. We recall the following result:

Theorem 2.23 (Groth–Kohlweiss [GK15, Thm. 2])**.** *Π_{BitProof} is a Σ-protocol for the relation R_{BitProof}.*

Applying Theorem 2.12, we obtain a secure instantiation of the ideal functionality $\mathcal{F}_{\mathsf{zk}}^{\mathsf{BitProof}}$.

We finally recall the "committed non-interactive zero knowledge" ideal functionality $\mathcal{F}_{\mathsf{com\text{-}zk}}^{R}$ associated to a relation R (see e.g. [LNR18, Func. 3.4]):

FUNCTIONALITY 2.24 ($\mathcal{F}_{\mathsf{com\text{-}zk}}^{R}$—committed ZKPOK functionality R)**.**
A relation R is fixed. There are two players, P_0 and P_1.

- Upon receiving a message of the form $(\texttt{commit-prove}, x; w)$, from player P_ν say, $\mathcal{F}_{\mathsf{zk}}^{R}$ stores $(\texttt{commit-prove}, x, R(x, w))$ in memory and sends $(\texttt{proof-receipt})$ to $P_{1-\nu}$.
- Upon receiving a message of the form $(\texttt{decommit-prove}, x)$, from player P_ν say, $\mathcal{F}_{\mathsf{com\text{-}zk}}^{R}$ checks whether $(\texttt{commit-prove}, x, R(x, w))$ is in memory. If it is, $\mathcal{F}_{\mathsf{com\text{-}zk}}^{R}$ sends $(\texttt{decommit-prove}, x, R(x, w))$ to $P_{1-\nu}$; otherwise, $\mathcal{F}_{\mathsf{com\text{-}zk}}^{R}$ sends $(\texttt{decommit-prove}, x, 0)$ to $P_{1-\nu}$.

As [LNR18, § 3.3] argues, $\mathcal{F}_{\mathsf{com\text{-}zk}}^{R}$ can be securely instantiated given a ZKPOK for R and a commitment scheme. We thus likewise obtain analogous instantiations of $\mathcal{F}_{\mathsf{com\text{-}zk}}^{R}$ for each relation R discussed above.

3 Piecewise Constant Codes and Hyperplane Coverings

In this section, we study which boolean functions $f : \{0,1\}^n \to \{0,1\}$—or more precisely, which sets $S := f^{-1}(1) \subset \{0,1\}^n$—can be covered using polynomially many hyperplanes over an n-bit prime q.

The following definition is implicit in [AF93] and [AGG⁺21].

Definition 3.1. We say that a family $\{H_i\}_{i=0}^{m-1}$ of affine hyperplanes in $\mathbb{F}_q^n$ *covers* a subset $S \subset \{0,1\}^n$ if $S = \bigcup_{i=0}^{m-1} H_i \cap \{0,1\}^n$.

That is, the hyperplanes' respective restrictions to the cube cover exactly S, and no further cube elements. Equivalently, the family $\{H_i\}_{i=0}^{m-1}$ expresses S as a union of intersection patterns.

Definition 3.2. The class $\mathbf{H}$ consists of those languages $L \subset \{0,1\}^*$ for which, for each $n \in \mathbb{N}$, the on-set $S_n := L \cap \{0,1\}^n$ can be covered by polynomially many hyperplanes over some fixed n-bit prime q.

That is, $L \in \mathbf{H}$ if and only if, for some polynomial function $m = \mathsf{poly}(n)$ and each $n \in \mathbb{N}$, there is some n-bit prime q such that $L \cap \{0,1\}^n = \bigcup_{i=0}^{m-1} H_i \cap \{0,1\}^n$, where each $H_i \subset \mathbb{F}_q^n$ is an affine hyperplane. In this setting, we also say that the function family $\{f_n : \{0,1\}^n \to \{0,1\}\}_{n \in \mathbb{N}}$ defined by $f_n(\mathbf{x}) := \mathbf{x} \in L$ is *efficiently computable by affine hyperplanes.*

3.1 Main theorem on piecewise constant codes and intersection patterns

Our main mathematical result shows that those $S \subset \{0,1\}^n$ expressible as "compact" piecewise constant codes are also coverable by polynomial-cardinality collections of hyperplanes. We derive various consequences of this result below.

We begin with a handful of definitions and lemmas. The following lemma is purely combinatorial:

Lemma 3.3. *Fix a natural number n. Across all partitions $n = n_0 + \cdots + n_{t-1}$ of n, the product expression $\prod_{i=0}^{t-1}(n_i + 1)$ is maximized by the partition $n = 1 + \cdots + 1$ (where it attains the value 2^n).*

Proof. We fix an arbitrary partition $n = \sum_{i=0}^{t-1} n_i$, and suppose that some summand $n_i > 1$. Though the term n_i alone contributes $(n_i + 1)$ to the product, splitting it into the further terms 1 and $(n_i - 1)$ would preserve the sum, and yet contribute to the product a factor of $2 \cdot n_i$, which is strictly larger than $n_i + 1$. $\square$

We also state the following related lemma, whose proof is similar:

Lemma 3.4. *For each partition $n = \sum_{i=0}^{t-1} n_i$ for which some summand $n_i \geq 3$, we have $\prod_{i=0}^{t-1}(n_i+1) \leq 2^{n-1}$.*

Though apparently new, Lemmas 3.3 and 3.4 evoke various classical problems (see e.g. Došlić [Doš05]).

There is a particular sort of pattern in a piecewise constant code's multidimensional array which will be of special importance to us.

Definition 3.5. Fix $n \in \mathbb{N}$ and a partition $n = n_0 + \cdots + n_{t-1}$. We call a subset $\overline{C} \subset \prod_{i=0}^{t-1}\{0, \ldots, n_i\}$ a *quasicube* if $\overline{C}$ takes the form

$$\overline{C} = \left\{(w_0, \ldots, w_{t-1}) \in \textstyle\prod_{i=0}^{t-1}\{0, \ldots, n_i\} \;\middle|\; \bigwedge_{i=0}^{k-1} w_{c_i} = v_i\right\},$$

where $\{c_0, \ldots, c_{k-1}\} \subset \{0, \ldots, t-1\}$ is a subsequence, and $v_i \in \{0, \ldots, n_{c_i}\}$ for each $i \in \{0, \ldots, k-1\}$.

In other words, a quasicube consists of those multi-indices *some* of whose components w_{c_i} are bound to fixed constants $v_i \in \{0, \ldots, n_{c_i}\}$, and the rest of which are free.

Example 3.6. Each single cell $\{(w_0, \ldots, w_{t-1})\} \subset \prod_{i=0}^{t-1}\{0, \ldots, n_i\}$ is obviously a quasicube (with all values bound, so that $\{c_0, \ldots, c_{t-1}\} = \{0, \ldots, t-1\}$ and $v_i = w_{c_i}$ for each $i \in \{0, \ldots, t-1\}$).

Example 3.7. Each code $S \subset \{0,1\}^n$ becomes piecewise constant with respect to the "trivial partition" $n = 1 + \cdots + 1$. This particular partition's corresponding cell array degenerates to the cube $\{0,1\}^n$ itself, and $\overline{S} = S$ for each $S \subset \{0,1\}^n$. Moreover, the quasicubes correspond exactly to the subcubes $C \subset \{0,1\}^n$.

The following lemma is the technical core of this section:

Lemma 3.8. *Fix a natural number $n \in \mathbb{N}$ and any n-bit prime q. For each partition $n = n_0 + \cdots + n_{t-1}$ and each quasicube $\overline{C} \subset \prod_{i=0}^{t-1}\{0, \ldots, n_i\}$, the pullback $C \subset \{0,1\}^n$ of C is an $\mathbb{F}_q$-intersection pattern.*

Proof. We prove the lemma by constructing an appropriate hyperplane. We fix $\overline{C}$ and q as in the hypothesis of the lemma, and write $\{c_0, \ldots, c_{k-1}\} \subset \{0, \ldots, t-1\}$ and $(v_0, \ldots, v_{k-1}) \in \prod_{i=0}^{k-1}\{0, \ldots, n_{c_i}\}$ for the bound values guaranteed to exist by definition of $\overline{C}$. We now define:

$$H : (x_0, \ldots, x_{n-1}) = \mathbf{x}_0 \parallel \cdots \parallel \mathbf{x}_{t-1} \mapsto \sum_{i=0}^{k-1} \left(\prod_{j<i}(n_{c_j} + 1)\right) \cdot \left(\sum_{l=0}^{n_{c_i}-1} \mathbf{x}_{c_i,l} - v_i\right),$$

where $\mathbf{x}_{c_i,l}$ denotes the l^{th} bit of the segment $\mathbf{x}_{c_i}$ (for $l \in \{0, \ldots, n_{c_i} - 1\}$). H is clearly a hyperplane.

We now argue that H correctly satisfies $H \cap \{0,1\}^n = C$. We prove this fact by induction on k, the number of bound values in the quasicube. For each $k^* \in \{0, \ldots, k\}$, we write $\overline{C}_{k^*}$ for the quasicube defined by $\overline{C}$'s first k^* bound values $(v_0, \ldots, v_{k^*-1})$ and C_{k^*} for its pullback, and moreover consider the partial sum $H_{k^*} : (x_0, \ldots, x_{n-1}) \mapsto \sum_{i=0}^{k^*-1} \prod_{j<i}(n_{c_j}+1) \cdot \left(\sum_{l=0}^{n_{c_i}-1} \mathbf{x}_{c_i,l} - v_i\right)$; we argue that $H_{k^*} \cap \{0,1\}^n = C_{k^*}$ for each $k^* \in \{0, \ldots, k\}$. In the base case $k^* = 0$, there's nothing to prove. We thus fix $k^* \in \{1, \ldots, k\}$, and assume by induction that $H_{k^*-1}(\mathbf{x}) = 0$ if and only if $\mathbf{x} \in C_{k^*-1}$; we *moreover* assume that $H_{k^*-1}(\mathbf{x})$ (viewed as an element of $\mathbb{Z}$) resides within the symmetric integer range $\left\{-\prod_{j<k^*-1}(n_{c_j} + 1) + 1, \ldots, \prod_{j<k^*-1}(n_{c_j} + 1) - 1\right\}$ for each $\mathbf{x} \in \{0,1\}^n$ (i.e., regardless of whether $\mathbf{x} \in C_{k^*-1}$). We now consider the $k^* - 1^{\text{st}}$ (i.e., highest) summand of H_{k^*}. The inner expression $\sum_{l=0}^{n_{c_{k^*-1}}-1} \mathbf{x}_{c_{k^*-1},l} - v_{k^*-1}$ clearly equals 0 if and only if $w(\mathbf{x}_{k^*-1}) = v_{k^*-1}$; in any case, it moreover resides within the integer range $\{-n_{c_{k^*-1}}, \ldots, n_{c_{k^*-1}}\}$ (actually, it resides within $\{-v_{k^*-1}, \ldots, n_{c_{k^*-1}} - v_{k^*-1}\}$, but our weaker bound is enough). By adding $H_{k^*-1}(\mathbf{x})$ to $\prod_{j<k^*-1}(n_{c_j} + 1)$ times this latter expression, we see that the result $H_{k^*}(\mathbf{x})$ has absolute value at most:

$$\prod_{j<k^*-1}(n_{c_j} + 1) - 1 + \left(\prod_{j<k^*-1}(n_{c_j} + 1)\right) \cdot n_{c_{k^*-1}} = \prod_{j<k^*}(n_{c_j} + 1) - 1.$$

This is exactly the range we need in order to preserve the inductive hypothesis. It remains to argue that $H_{k^*}(\mathbf{x}) = 0$ if and only if $\mathbf{x} \in C_{k^*}$. But $\mathbf{x} \in C_{k^*}$ if and only if $\mathbf{x} \in C_{k^*-1}$ and $w(\mathbf{x}_{c_{k^*-1}}) = v_{k^*-1}$. If both of these are true, then both $H_{k^*-1}(\mathbf{x})$ (by induction) and the top summand (discussed above) equal 0, as needed. On the other hand, if *either* of these conditions is false, then either $H_{k^*-1}(\mathbf{x})$ is a nonzero element of $\left\{-\prod_{j<k^*-1}(n_{c_j}+1)+1, \ldots, \prod_{j<k^*-1}(n_{c_j}+1)-1\right\}$ (by induction), or the top summand is $\prod_{j<k^*-1}(n_{c_j}+1)$ times a nonzero element of $\{-n_{c_{k^*-1}}, \ldots, n_{c_{k^*-1}}\}$ (by above). The sum of two such elements cannot be zero (the sets are both additively symmetric and disjoint, and no cancellation can happen).

Completing the induction, we see that $H(\mathbf{x})$, viewed as an integer, is an element of the range $\left\{-\prod_{j<k}(n_{c_j}+1)+1, \ldots, \prod_{j<k}(n_{c_j}+1)-1\right\}$, which moreover is 0 (as an integer) if and only if $\mathbf{x} \in C$. It remains to argue that this quantity cannot overflow modulo q (and so unduly yield the sum of 0 in $\mathbb{F}_q$). By Lemma 3.3, $\prod_{j<k}(n_{c_j}+1)$ is at most 2^n. We thus see that if $q \geq 2^n$, then no overflow can occur.

We can weaken this requirement to $q \geq 2^{n-1}$, with a bit of extra work. Exploiting Lemmas 3.3 and 3.4, we note that the stronger upper-bound $\prod_{j<k}(n_{c_j}+1) \leq 2^{n-1}$ in fact holds *unless* $k = t$—so that all of $\overline{C}$'s components are bound, and $\sum_{i=0}^{k-1} n_i = \sum_{i=0}^{t-1} n_i = n$—*and* moreover the partition $n = n_0 + \cdots + n_{t-1}$ consists only of 1s and 2s (in fact, it can contain at most two 2s, but we ignore this further fact). We handle this latter case separately using a different construction. After permuting coordinates, we may assume that the 2s occur in contiguous pairs at the beginning; we write $t^* \in \{0, \ldots, \lfloor \frac{n}{2} \rfloor\}$ for the number for which $n_0 = \cdots = n_{t^*-1} = 2$ and $n_{t^*} = \cdots = n_{t-1} = 1$. After applying Lemma 2.1, we may further assume that $v_{t^*} = \cdots = v_{t-1} = 0$. It follows similarly to the above argument that the hyperplane

$$H : (x_0, \ldots, x_{n-1}) \mapsto \sum_{i=0}^{t^*-1} 3^i \cdot (x_{2i} + x_{2i+1} - v_i) + 3^{t^*} \cdot \left(\sum_{i=t^*}^{t-1} x_{2 \cdot t^* + i} \right)$$

suffices to define C; moreover, it returns integers in the range $\{-3^{t^*}+1, \ldots, 3^{t^*}-1+3^{t^*} \cdot (n - 2 \cdot t^*)\}$, which is well within $\{-2^{n-1}+1, \ldots, 2^{n-1}-1\}$ regardless of $t^* \in \{0, \ldots, \lfloor \frac{n}{2} \rfloor\}$. This completes the proof. □

Remark 3.9. The proof of Lemma 3.8 can be understood from the following perspective. The individual hyperplanes $\sum_{l=0}^{n_{c_i}-1-1} \mathbf{x}_{c_i,l} - v_i$ (for $i \in \{0, \ldots, k-1\}$) constructed during the proof of Lemma 3.8 intersect in a $n-k$-dimensional *affine flat*, which intersects the cube exactly at C; the challenge is to "extend" this flat into a hyperplane without accruing new cube points. Having computed the individual hyperplanes $\sum_{l=0}^{n_{c_i}-1-1} \mathbf{x}_{c_i,l} - v_i$ (for $i \in \{0, \ldots, k-1\}$), H interprets these k individual outputs as the "digits" of a number in a nonstandard, mixed-radix, signed-digit positional numeral system—whose respective "places" range throughout $\{-n_{c_i}, \ldots, n_{c_i}\}$—and returns the resulting number. The key property of this (unusual) system is that while numbers don't in general have unique representations, 0 *does*.

We now present the main result of this subsection, a consequence of Lemma 3.8:

Theorem 3.10. *If $\{S_n \subset \{0,1\}^n\}_{n \in \mathbb{N}}$ is such that each S_n is expressible as a piecewise constant code whose cell representation $\overline{S_n}$ moreover admits a covering by polynomially many quasicubes, then $\{S_n\}_{n \in \mathbb{N}}$ is in* **H**.

Proof. If S_n is piecewise constant—with respect to $n = n_0 + \cdots + n_{t-1}$, say—and $\overline{S_n} = \bigcup_{i=0}^{m-1} \overline{C_i}$ for quasicubes $\overline{C_i} \subset \prod_{i=0}^{t-1}\{0, \ldots, n_i\}$, then likewise $S_n = \bigcup_{i=0}^{m-1} C_i$, where, for each $i \in \{0, \ldots, m-1\}$, C_i is the pullback of $\overline{C_i}$. The result follows immediately from Lemma 3.8. □

Theorem 3.10 is extremely powerful, and subsumes all hyperplane-covering constructions we're aware of. We immediately record the following corollary:

Corollary 3.11. *If a family $\{S_n \subset \{0,1\}^n\}_{n \in \mathbb{N}}$ is such that each S_n is a piecewise constant code with a cell representation $\overline{S_n}$ consisting of only polynomially many filled cells, then $\{S_n\}_{n \in \mathbb{N}}$ is in* **H**.

Proof. Every cell is a quasicube (see Example 3.6); the result thus follows directly from Theorem 3.10. □

We discuss specific consequences in the next subsection.

3.2 Applications of the main theorem

We now apply Theorem 3.10 to a number of specific cases. Surprisingly, many natural boolean functions $f : \{0,1\}^n \to \{0,1\}$ have on-sets $f^{-1}(1)$ which satisfy the hypothesis of Theorem 3.10. We begin with the following fundamental result:

Theorem 3.12. $\mathbf{\Sigma_2} \subset \mathbf{H}$.

Proof. Each code $S_n := L \cap \{0,1\}^n$ is piecewise constant with respect to the trivial partition $n = \sum_{i=0}^{n-1} 1$ of Example 3.7. By hypothesis on L, each S_n is moreover coverable by polynomially many subcubes; these are exactly the quasicubes with respect to $n = \sum_{i=0}^{n-1} 1$, and the result follows from Theorem 3.10. □

Concretely, each subcube $C \subset S_n$—of the form $C = \left\{(x_0, \ldots, x_{n-1}) \in \{0,1\}^n \;\middle|\; \bigwedge_{i=0}^{k-1} x_{c_i} = 0\right\}$, say, for some subsequence $\{c_0, \ldots, c_{k-1}\} \subset \{0, \ldots, n-1\}$ (we assume that C contains the origin, by Lemma 2.1)—can be exactly covered by the single hyperplane $H : (x_0, \ldots, x_{n-1}) \mapsto \sum_{i=0}^{k-1} x_{c_i}$, provided that $q > n$.

The following function appears in Goldreich [Gol04, § 7.2.1]:

Example 3.13. For even n, the function $f : (x_0, \ldots, x_{n-1}) \mapsto \bigvee_{i=0}^{n/2-1} (x_{2i} \wedge x_{2i+1})$ returns true if and only the bitwise AND of its argument's even-indexed and odd-indexed substrings contains a 1. Alternatively, f checks whether the sets represented by these even-indexed and odd-indexed substrings are non-disjoint. As f is decided by a $\mathbf{\Sigma_2}$-circuit, we conclude from Theorem 3.12 that f is efficiently computable by hyperplanes. In fact, f is piecewise constant even with respect to the coarser partition $n = \sum_{i=0}^{n/2-1} 2$; its cell representation under this partition is a union of $\frac{n}{2}$ quasicubes (each with one component bound to 2 and the rest free). These $\frac{n}{2}$ quasicubes collectively cover $3^{n/2} - 2^{n/2}$—an exponentially large number in n—cells; we conclude that the relative generality of Theorem 3.10 (over and above Corollary 3.11) conveys utility. Concretely, the hyperplanes $H_i : (x_0, \ldots, x_{n-1}) \mapsto x_{2i} + x_{2i+1} - 2$ (for $i \in \left\{0, \ldots, \frac{n}{2} - 1\right\}$) suffice to compute f (if $q \geq 3$).

We write $\mathbf{S}_n$ for the set of symmetric functions on n variables (see e.g. [Weg87, § 3.4]).

Theorem 3.14. *Each symmetric function f in $\mathbf{S}_n$ is coverable by at most $n+1$ hyperplanes.*

Proof. By definition, each symmetric f's on-set is piecewise constant with respect to the partition $n = n$. Because the entire cell array of this partition has just $n+1$ cells, the result follows from Corollary 3.11. □

Concretely, it suffices to use the hyperplane $H_i : (x_0, \ldots, x_{n-1}) \mapsto \sum_{i=0}^{n-1} x_i - i$ to cover each filled cell $i \in \{0, \ldots, n\}$ (i.e., each i for which $\{\mathbf{x} \in \{0,1\}^n \mid w(\mathbf{x}) = i\} \subset f^{-1}(1)$). This is correct so long as $q > n$.

Example 3.15. The *majority* function $f : (x_0, \ldots, x_{n-1}) \mapsto \sum_{i=0}^{n-1} x_i \geq \left\lceil \frac{n}{2} \right\rceil$ is obviously symmetric; we conclude from Theorem 3.14 that $f^{-1}(1)$ is coverable by at most $n+1$ hyperplanes (in fact, $\left\lfloor \frac{n}{2} \right\rfloor + 1$ suffice).

Corollary 3.16. $\mathbf{H} \not\subset \mathbf{AC}^0$.

Proof. We refer to Example 3.15 and the fact that majority is not in $\mathbf{AC}^0$ (see [Weg87, § 11 Thm. 4.1]). □

Example 3.17. For even n, the *Hamming-weight comparator* function $f : (x_0, \ldots, x_{n-1}) \mapsto \sum_{i=0}^{n/2-1} x_{2i} - x_{2i+1} \geq 0$ differs from majority by pre-composition with the affine bijection $(x_0, \ldots, x_{n-1}) \mapsto (x_0, 1 - x_1, \ldots, x_{n-2}, 1 - x_{n-1})$. We conclude from Example 3.15 that $f^{-1}(1)$ can be covered using $\frac{n}{2} + 1$ hyperplanes.

The following function also appears in Goldreich [Gol04, § 7.2.1]:

Example 3.18. For even n, the function $f : (x_0, \ldots, x_{n-1}) \mapsto \bigwedge_{i=0}^{n/2-1} (x_{2i} \oplus \overline{x_{2i+1}})$ checks whether its argument's even-indexed and odd-indexed substrings are equal. By applying the affine-linear bijection $(x_0, \ldots, x_{n-1}) \mapsto (x_0, 1 - x_1, \ldots, x_{n-2}, 1 - x_{n-1})$, we see that it's enough to consider the function $f : (x_0, \ldots, x_{n-1}) \mapsto \bigwedge_{i=0}^{n/2-1} (x_{2i} \oplus x_{2i+1})$. This latter function f's on-set $f^{-1}(1) \subset \{0,1\}^n$ is piecewise constant with respect to the partition $n = \sum_{i=0}^{n/2-1} 2$, represented moreover by the *single* cell $(1, \ldots, 1) \in \prod_{i=0}^{n/2-1} \{0,1,2\}$. Applying Lemma 3.8, we see that $H : (x_0, \ldots, x_{n-1}) \mapsto \sum_{i=0}^{n/2-1} 3^i \cdot (x_{2i} + x_{2i+1} - 1)$ satisfies $f^{-1}(1) = H \cap \{0,1\}^n$ (at least if $q \geq 3^{n/2}$). Interestingly, this hyperplane returns exactly the integer whose *balanced ternary* representation is $(x_{2i} + x_{2i+1} - 1)_{i=0}^{n/2-1}$ (this integer is 0 if and only if $f(\mathbf{x}) = 1$).

Example 3.19. Again for even n, we consider the integer comparison function $f : (x_0, \ldots, x_{n-1}) \mapsto \sum_{i=0}^{n/2-1} 2^i \cdot (x_{2i} - x_{2i+1}) > 0$ (true if and only if the little-endian unsigned integers $\mathbf{x}_0$ and $\mathbf{x}_1$ represented respectively by $\mathbf{x}$'s even-indexed and odd-indexed substrings satisfy $\mathbf{x}_0 > \mathbf{x}_1$). By applying the affine bijection $(x_0, \ldots, x_{n-1}) \mapsto (x_0, 1 - x_1, \ldots, x_{n-2}, 1 - x_{n-1})$, we see that it's equivalent to consider the function $f : (x_0, \ldots, x_{n-1}) \mapsto \sum_{i=0}^{n/2-1} 2^i \cdot (x_{2i} + x_{2i+1}) \geq 2^{n/2}$ (true if and only if the sum $\mathbf{x}_0 + \mathbf{x}_1 \geq 2^{n/2}$ overflows).

We argue first that this latter function f is such that $f^{-1}(1)$ is piecewise constant with respect to the partition $n = \sum_{i=0}^{n/2-1} 2$, and hence that Theorem 3.10 applies to f. Indeed, f is evaluated by the following variant of a standard comparison circuit:

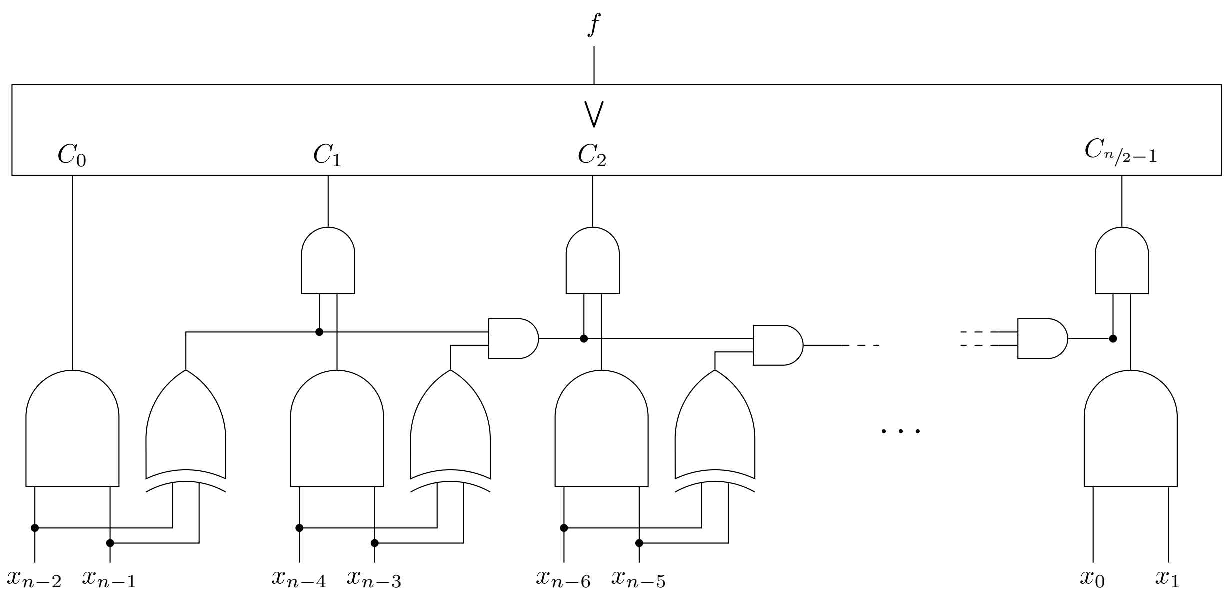

Figure 1: A well-known boolean circuit evaluating whether two integers' sum generates a carry.

We observe that each among the $\frac{n}{2}$ *output wires* of this circuit evaluates to true exactly on the pullback of a quasicube (with respect to $n = \sum_{i=0}^{n/2-1} 2$). Indeed, for each $i \in \{0, \ldots, \frac{n}{2} - 1\}$, the wire labeled C_i of the above circuit is exactly the pullback of the quasicube $\overline{C}_i \subset \prod_{i=0}^{n/2-1} \{0, 1, 2\}$ defined by the trailing indices $\left\{\frac{n}{2} - 1 - i, \frac{n}{2} - 1\right\} \subset \left\{0, \ldots, \frac{n}{2} - 1\right\}$, respectively bound to the values $(v_0, \ldots, v_i) = (2, 1, \ldots, 1)$. Each such pullback is an intersection pattern, by Theorem 3.10; we conclude that $f^{-1}(1)$ is coverable by $\frac{n}{2}$ hyperplanes.

Applying a version of the "special case" of the proof of Lemma 3.8, we obtain the concrete expressions:

$$H_i : (x_0, \ldots, x_{n-1}) \mapsto \sum_{j<i} 2^j \cdot \left(x_{n-2\cdot(j+1)} + x_{n-2\cdot(j+1)+1} - 1\right) + 2^i \cdot \left(x_{n-2\cdot(i+1)} + x_{n-2\cdot(i+1)+1} - 2\right)$$

for $i \in \left\{0, \ldots, \frac{n}{2} - 1\right\}$; these are correct so long as $q \geq \frac{3}{2} \cdot 2^{n/2}$. Analogous hyperplanes for the original comparator f follow from an appropriate affine transformation. We note that these hyperplanes can be evaluated on any input $\mathbf{x} \in \{0, 1\}^n$ in $O(n)$ total time, using an obvious expression-sharing scheme.

Example 3.20. In [CLS86, Fig. 6], Cohen, Lobstein and Sloane—using a piecewise constant construction—introduce a new family of $(2R+4, 12)R$-codes (i.e., cardinality-12 codes $S \subset \{0, 1\}^{2R+4}$ whose covering radius is R). In particular, their construction establishes the upper-bound $K(2R + 4, R) \leq 12$ for each $R \geq 1$ (i.e., there exist R-covering codes $S \subset \{0, 1\}^{2R+4}$ of cardinality 12 or less). As of the publication of [CHLL97], 12 remains the best-known upper bound of $K(2R+4, R)$ for *each* value $R \in \{1, \ldots, 10\}$ treated in the extensive [CHLL97, Table 6.1] (this upper-bound is known to be tight only in the cases $R \in \{1, 2\}$).

The construction [CLS86, Fig. 6] uses the partition $2R + 4 = (2R - 2) + 3 + 3$, and employs exactly 4 cells $(w_0, w_1, w_2) \in \{0, \ldots, 2R - 2\} \times \{0, 1, 2, 3\} \times \{0, 1, 2, 3\}$ (namely $(0, 1, 0)$, $(0, 2, 3)$, $(2R - 2, 3, 1)$ and $(2R - 2, 0, 2)$). Corollary 3.11 implies that S can be covered by exactly 4 hyperplanes; these are correct so long as the prime field order $q \geq (2R - 1) \cdot 4^2$.

3.3 Upper limits on H's size

In this section, we turn to negative results on the power of hyperplane-based computation.

Theorem 3.21. $\mathbf{H} \subset \mathbf{NC}^1$.

Proof. We prove the theorem by an explicit construction. We fix a collection of hyperplanes $\{H_i\}_{i=0}^{m-1}$ over an n-bit prime q, and construct a corresponding fan-in 2 log-depth circuit.

We first express each individual hyperplane H_i as a log-depth boolean circuit. The linear combination $\alpha_i + \alpha_{i,0} \cdot x_0 + \cdots + \alpha_{i,n-1} \cdot x_{n-1}$ evaluated by H_i, restricted to boolean inputs, is actually a *subset sum* (i.e., each $\alpha_{i,j}$ is either present or absent). We thus set each x_j as the select bit of a multiplexer with inputs the n-bit string of 0s and $\alpha_{i,j}$ (we recall that q is an n-bit prime). By [Vol99, Thm. 1.20], the "iterated addition" of the n n-bit outputs of the multiplexers and the affine constant α_i can be carried out using a log-depth bounded fan-in circuit. The output of this circuit—namely, $\alpha_i + \alpha_{i,0} \cdot x_0 + \cdots + \alpha_{i,n-1} \cdot x_{n-1} = H_i(\mathbf{x})$—is an *integer* of bit-length $n + O(\log n)$; we must reduce this number modulo q (actually, it's enough to check whether it's a multiple of q). This is essentially [Vol99, Ex. 1.19 (a)], and can be done in log-depth using Barrett's modular reduction; in particular, we apply Menezes, van Oorschot, and Vanstone [MvOV97, Alg. 14.42] to $\mathbf{x}$ using the radix $b = 4$. The resulting circuit uses only a constant number of shifts and multiplications (which themselves can be carried out in log-depth; see [Vol99, Thm. 1.23]).

It remains to check whether *any* of the equalities $H_i(\mathbf{x}) \equiv 0 \pmod{q}$ is true (for $i \in \{0, \ldots, m-1\}$). This can be done using a tree of OR gates of depth $O(\log m)$, which is $O(\log n)$ if m is polynomial in n. $\square$

The construction of Theorem 3.21 is depicted in Figure 2.

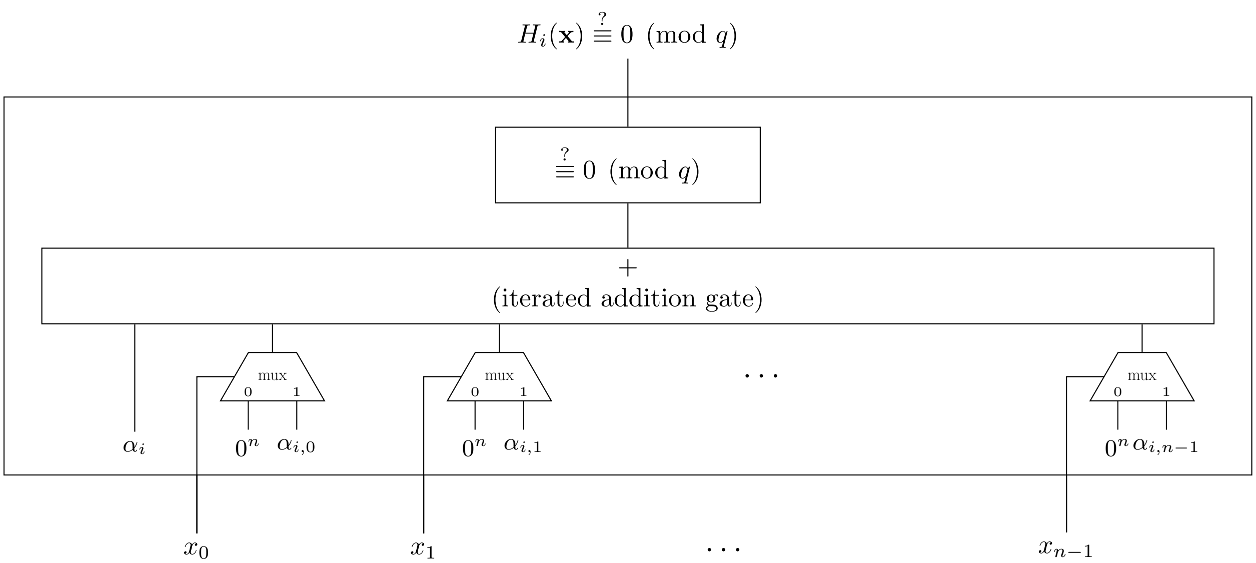

Figure 2: A log-depth, bounded fan-in boolean circuit evaluating an affine hyperplane.

Though we're currently unable to prove it, we believe that the inclusion of Theorem 3.21 is strict. To argue this, we begin with the following example, which is interesting in its own right; indeed, it describes a family which does *not* satisfy the hypothesis of Theorem 3.10. The function f below (perhaps tellingly) is called the "Achilles' heel function" in the logic synthesis literature (see e.g. [BHMSV84, § 4.9]).

Example 3.22. For even n, the function $f : (x_0, \ldots, x_{n-1}) \mapsto \bigwedge_{i=0}^{n/2-1} (x_{2i} \vee x_{2i+1})$ returns true if and only if the bitwise OR of its argument's even-indexed and odd-indexed substrings consists entirely of 1s. Alternatively, f checks whether the union of the two subsets of $\{0, \ldots, \frac{n}{2} - 1\}$ represented respectively by its argument's even-indexed and odd-indexed substrings equals the whole set $\{0, \ldots, \frac{n}{2} - 1\}$. The function f is in $\mathbf{\Pi_2}$; moreover, f is piecewise constant with respect to $n = \sum_{i=0}^{n/2-1} 2$ (f is the complement of the function of Example 3.13, up to a reflection of the cube).

We show that f does *not* satisfy the hypothesis of Theorem 3.10. In fact, for *any* partition of n with respect to which f becomes piecewise constant, its resulting cell representation requires exponentially many quasicubes to cover. To prove this, we first argue that f is piecewise constant *only* with respect to $n = \sum_{i=0}^{n/2-1} 2$ (and its refinements). Indeed, we fix an arbitrary partition, say $\mathcal{F}$, of $\{0, \ldots, n-1\}$, which is not a refinement of $\{0, \ldots, n-1\} = \bigsqcup_{i=0}^{n/2-1} \{2i, 2i+1\}$, and assume for contradiction that $f^{-1}(1)$ is piecewise constant with respect to $\mathcal{F}$. By hypothesis on $\mathcal{F}$, there exist elements j_0 and j_1 of $\{0, \ldots, n-1\}$ which belong to the same subset $F \in \mathcal{F}$ (say) but for which $\{j_0, j_1\} \neq \{2i, 2i+1\}$ holds for each $i \in \{0, \ldots, \frac{n}{2}-1\}$. Without loss of generality, we assume that both j_0 and j_1 are even. We treat separately the cases $\{j_0, j_0+1, j_1, j_1+1\} \not\subset F$ and $\{j_0, j_0+1, j_1, j_1+1\} \subset F$. In the former case, we have $j_k + 1 \notin F$ for some $k \in \{0, 1\}$. Because the three components $(x_{j_k}, x_{j_k+1}, x_{j_{1-k}})$ attain the values $(1, 0, 0)$ at *some* appropriate element $(x_0, \ldots, x_{n-1}) \in f^{-1}(1)$, we conclude that $f^{-1}(1)$'s cell representation with respect to $\mathcal{F}$ contains a cell for which the weights at both $\{j_0, j_1\}$'s and $\{j_k+1\}$'s respective subsets of $\mathcal{F}$ are simultaneously less than full. It follows that $f^{-1}(1)$ contains all such vectors, including at least one for which the components $(x_{j_k}, x_{j_k+1}, x_{j_{1-k}})$ attain the values $(0, 0, 1)$. This contradicts the definition of $f^{-1}(1)$. We now suppose that $\{j_0, j_0+1, j_1, j_1+1\} \subset F$. We let $k \in \{0, 1\}$ be arbitrary, and note that the components $(x_{j_k}, x_{j_k+1}, x_{j_{1-k}})$ attain the values $(0, 1, 0)$ at some appropriate element of $(x_0, \ldots, x_{n-1}) \in f^{-1}(1)$. We conclude that $f^{-1}(1)$'s cell representation with respect to $\mathcal{F}$ contains a cell whose weight at F is at least two less than full. This implies that the components $(x_{j_k}, x_{j_k+1}, x_{j_{1-k}})$ also attain the values $(0, 0, 1)$ at some element $(x_0, \ldots, x_{n-1}) \in f^{-1}(1)$, which again contradicts the definition of $f^{-1}(1)$. It thus remains only to treat $\bigsqcup_{i=0}^{n/2-1} \{2i, 2i+1\}$ and its refinements. We suppose that $\mathcal{F}$ is a (possibly non-strict) refinement of $\bigsqcup_{i=0}^{n/2-1} \{2i, 2i+1\}$; we write $\overline{S}$ for $f^{-1}(1)$'s cell representation with respect to this partition. Because $\left|f^{-1}(1)\right| = 3^{n/2}$, it suffices to argue that, for each quasicube $\overline{C}$ satisfying $\overline{C} \subset \overline{S}$, the pullback C of $\overline{C}$ satisfies $|C| \leq 2^{n/2}$. By hypothesis on $\mathcal{F}$, each adjacent tuple $\{2i, 2i+1\}$ equals the disjoint union of either one or two among $\mathcal{F}$'s subsets; moreover, by definition of $f^{-1}(1)$, at least one of these subsets' components must be bound for any quasicube $\overline{C} \subset \overline{S}$. Expanding cases, we see that each tuple $\{2i, 2i+1\}$ contributes a factor of at most 2 to the product expression defining C's size (for $i \in \{0, \ldots, \frac{n}{2}-1\}$). This completes the proof.

I would like thank Jason Long for suggesting the consideration of this function.

It is also extremely interesting to ask whether the family f of Example 3.22 belongs to $\mathbf{H}$. Is non-membership in $\mathbf{H}$ would imply $\mathbf{\Pi_2} \not\subset \mathbf{H}$ and hence $\mathbf{AC}^0 \not\subset \mathbf{H}$. Intriguingly, extensive empirical evidence indicates that for each n-bit prime field $\mathbb{F}_q$ (and, indeed, for any field), each affine hyperplane $H \subset \mathbb{F}_q^n$ for which $H \cap \{0,1\}^n \subset f^{-1}(1)$ moreover satisfies $H \cap \{0,1\}^n \leq 2^{n/2}$. This fact, if true, would imply, as before, that $\mathbf{\Pi_2} \not\subset \mathbf{H}$; it would also suggest a close connection in the *negative* direction between piecewise constant codes and hyperplane coverings. We record this claim as a conjecture:

Conjecture 3.23. *If an affine hyperplane $H \subset \mathbb{F}_q^n$ satisfies $H \cap \{0,1\}^n \subset f^{-1}(1)$, then $H \cap \{0,1\}^n \leq 2^{n/2}$.*

We have been unable to prove or disprove Conjecture 3.23 despite significant effort.

4 Commitment-Consistent 2PC

In this section, we describe a hyperplane-based protocol for commitment-consistent secure two-party computation, as well as a key subprotocol for commitment-consistent secure iterated modular multiplication.

4.1 Review of private multiplication

Our protocols make use of the ZKPOK ideal functionalities $\mathcal{F}_{\mathsf{zk}}^{\mathsf{DH}}$, $\mathcal{F}_{\mathsf{zk}}^{\mathsf{EqMsg}}$, $\mathcal{F}_{\mathsf{zk}}^{\mathsf{Prod}}$, and $\mathcal{F}_{\mathsf{zk}}^{\mathsf{BitProof}}$ already discussed in Subsection 2.6 above.

Following Lindell, Nof and Ranellucci, [LNR18, § 2.3], we moreover recall the notion of a secure multiplication protocol which is "private, but not necessarily correct".

FUNCTIONALITY 4.1 ($\mathcal{F}_{\mathsf{PrivMult}}$—the underlying private multiplication functionality)**.**
Players P_0 and P_1 and a prime q are fixed.

- Upon receiving $\left(\texttt{multiply}, (\langle\alpha_i\rangle_\nu, \langle\beta_i\rangle_\nu)_{i=0}^{m'-1}\right)$ from both parties P_ν, $\mathcal{F}_{\mathsf{PrivMult}}$ proceeds as follows:

  1: allocate a length-m' output vector $(\langle\gamma_i\rangle_\nu)_{i=0}^{m'-1}$ for each party $\nu \in \{0, 1\}$.
  2: **for** $i \in \{0, \ldots, m'-1\}$ **do**
  3: set $\gamma_i := (\langle\alpha_i\rangle_0 + \langle\alpha_i\rangle_1) \cdot (\langle\beta_i\rangle_0 + \langle\beta_i\rangle_1) \pmod q$.
  4: randomly additively share $\gamma_i = \langle\gamma_i\rangle_0 + \langle\gamma_i\rangle_1 \pmod q$.

  $\mathcal{F}_{\mathsf{PrivMult}}$ sends the message $\left(\texttt{multiply}, (\langle\gamma_i\rangle_\nu)_{i=0}^{m'-1}\right)$ to P_ν, for each $\nu \in \{0, 1\}$.

Definition 4.2 (Lindell–Nof–Ranellucci [LNR18, § 2.3])**.** A protocol Π_{PrivMult} for Functionality 4.1 is *private* if, for each $C \in \{0, 1\}$, each real-world nonuniform PPT adversary $\mathcal{A}$ corrupting P_C, and each pair $\left(\langle\alpha_i\rangle_{1-C}, \langle\beta_i\rangle_{1-C}\right)_{i=0}^{m'-1}$ and $\left(\langle\alpha'_i\rangle_{1-C}, \langle\beta'_i\rangle_{1-C}\right)_{i=0}^{m'-1}$ of inputs on the part of P_{1-C}, the distributions describing $\mathcal{A}$'s output in Π_{PrivMult} in case P_{1-C} uses either of these inputs are computationally indistinguishable.

We now recall that $\mathcal{F}_{\mathsf{PrivMult}}$ can be instantiated privately, using a protocol of Doerner, Kondi, Lee, and shelat [DKLs18, § VI. D.]. An issue arises from the fact that, in that particular protocol, P_0 and P_1 directly submit the (vectors of) scalars they'd like to multiply componentwise, whereas, in Functionality 4.1, P_0 and P_1 only possess joint additive sharings of the desired multiplicands (that is, the functionality must first *reconstruct*, and only then multiply). We accommodate this issue in the following way. Given any pair $\alpha_i = \langle\alpha_i\rangle_0 + \langle\alpha_i\rangle_1$ and $\beta_i = \langle\beta_i\rangle_0 + \langle\beta_i\rangle_1$ (say) of *jointly* held multiplicands, P_0 and P_1 can obtain additive sharings of $\alpha_i \cdot \beta_i$ using 2 (simultaneous and vectorized) invocations of [DKLs18, § VI. D.], as we now argue. Indeed, by the distributive law, $\alpha_i \cdot \beta_i = (\langle\alpha_i\rangle_0 + \langle\alpha_i\rangle_1) \cdot (\langle\beta_i\rangle_0 + \langle\beta_i\rangle_1)$ equals

$$\langle\alpha_i\rangle_0 \cdot \langle\beta_i\rangle_0 + \langle\alpha_i\rangle_0 \cdot \langle\beta_i\rangle_1 + \langle\alpha_i\rangle_1 \cdot \langle\beta_i\rangle_0 + \langle\alpha_i\rangle_1 \cdot \langle\beta_i\rangle_1 .$$

P_0 and P_1 can locally compute the first and last terms $\langle\alpha_i\rangle_0 \cdot \langle\beta_i\rangle_0$ and $\langle\alpha_i\rangle_1 \cdot \langle\beta_i\rangle_1$, respectively. To obtain additive sharings of the middle two terms, P_0 and P_1 can submit $(\langle\alpha_i\rangle_0, \langle\beta_i\rangle_0)$ and $(\langle\beta_i\rangle_1, \langle\alpha_i\rangle_1)$, respectively, to [DKLs18, § VI. D.] (note the reversal of order). Upon obtaining the respective outputs $(\langle\eta_i\rangle_0, \langle\xi_i\rangle_0)$ and $(\langle\eta_i\rangle_1, \langle\xi_i\rangle_1)$, say, P_0 and P_1 can return $\langle\alpha_i\rangle_0 \cdot \langle\beta_i\rangle_0 + \langle\eta_i\rangle_0 + \langle\xi_i\rangle_0$ and $\langle\alpha_i\rangle_1 \cdot \langle\beta_i\rangle_1 + \langle\eta_i\rangle_1 + \langle\xi_i\rangle_1$ (respectively). By the above discussion, these outputs yield random shares of $\alpha_i \cdot \beta_i$, as desired. I would like to thank Yehuda Lindell for helping to clarify this point.

Using this argument, we thus obtain:

Theorem 4.3 (Doerner et al.)**.** *The protocol [DKLs18, § VI. D.]—used in the above way, with arity* $2 \cdot m'$*—yields an implementation of Functionality 4.1 which is private in the sense of Definition 4.2.*

We finally make use the following functionality from Lindell, Nof and Ranellucci [LNR18, Func. 4.2].

FUNCTIONALITY 4.4 ($\mathcal{F}_{\mathsf{CheckDH}}$—joint assessment of a Diffie–Hellman tuple)**.**
Two players P_0 and P_1 are fixed, as well as an $\mathbb{F}_q$-homomorphic encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$.

- Upon receiving $(\texttt{init})$ from both parties, $\mathcal{F}_{\mathsf{CheckDH}}$ runs $(pk, sk) \leftarrow \mathsf{Gen}(1^\lambda)$ and outputs $(\texttt{key}, pk)$.
- Upon receiving $(\texttt{check}, A)$ from both parties, $\mathcal{F}_{\mathsf{CheckDH}}$ returns $\left(\texttt{check}, \mathsf{Dec}_{sk}(A) \stackrel{?}{=} 0\right)$ to both.

The result [LNR18, Prop. 7.2] yields a secure instantiation of $\mathcal{F}_{\mathsf{CheckDH}}$ (or at least of its specialization to the El Gamal scheme, which is secure under the DDH assumption):

Lemma 4.5 (Lindell–Nof–Ranellucci)**.** *The protocol [LNR18, Prot. 7.1] securely computes* $\mathcal{F}_{\mathsf{CheckDH}}$ *in the* $\left(\mathcal{F}_{\mathsf{zk}}^{\mathsf{RE}}, \mathcal{F}_{\mathsf{com\text{-}zk}}^{\mathsf{DH}}\right)$*-hybrid model.*

Remark 4.6. Because our protocol has only two parties, we may slightly simplify the structure of the initialization subprotocol [LNR18, Prot. 4.3] of [LNR18, Prot. 7.1]. Indeed, instead of requiring that *all* parties invoke $\mathcal{F}_{\mathsf{com\text{-}zk}}^{R_{\mathsf{DL}}}$, we may dictate that P_0 go first, and that P_0 alone commit to its proof; P_1 must then prove, but not commit. Precisely this approach is taken by [DKLs18, Prot. 2] (the only difference there is that that sharing is multiplicative, as opposed to additive). An identical simplification can moreover be carried out in steps 1. and 2. of [LNR18, Prot. 7.1].

4.2 Secure iterated multiplication

We introduce the following key ideal functionality, for iterated secure modular multiplication which *moreover* is consistent with pre-held ciphertexts. We actually present a variant which also multiplicatively randomizes the resulting product (this randomization can be optionally removed, as we note in Remark 4.14 below).

Our iterated multiplication functionality works using ciphertexts which both parties know. To formally capture the fact that both parties must collectively know these common ciphertexts, we add a *committing* phase to our functionality, which distributes them.

FUNCTIONALITY 4.7 ($\mathcal{F}_{\mathsf{IterMult}}$—commitment-consistent iterated multiplication functionality)**.**
$\mathcal{F}_{\mathsf{IterMult}}$ involves parties P_0 and P_1 and an $\mathbb{F}_q$-homomorphic encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$.

- Upon receiving $(\texttt{init})$ from both parties, $\mathcal{F}_{\mathsf{IterMult}}$ forwards these messages to $\mathcal{F}_{\mathsf{CheckDH}}$, and returns the response $(\texttt{key}, pk)$ to both parties.
- Upon receiving $\left(\texttt{commit}, (Y_{i,\nu})_{i=0}^{m-1}\right)$ from a party P_ν, $\mathcal{F}_{\mathsf{IterMult}}$ forwards the message to $P_{1-\nu}$.
- Upon receiving $\left(\texttt{multiply}; (\langle y_i \rangle_\nu, s_{i,\nu})_{i=0}^{m-1}\right)$ from both parties P_ν, $\mathcal{F}_{\mathsf{IterMult}}$ executes:

```
randomly sample y ← F_q.    ▷ to omit re-randomization, replace this assignment with y := 1.
for i ∈ {0, ..., m − 1} do
    for ν ∈ {0, 1} do
        require Y_{i,ν} =? Enc_pk(⟨y_i⟩_ν ; s_{i,ν}); else send (multiply-abort) to both parties and abort.
    reconstruct y_i := ⟨y_i⟩_0 + ⟨y_i⟩_1  (mod q).
    overwrite y := y · y_i  (mod q).
output (multiply, y) to both parties.
```

We now show how to securely compute $\mathcal{F}_{\mathsf{IterMult}}$ in $O(\log m)$ rounds, by recursively applying ideas from [LNR18, Prot. 4.7]. We make use of a private multiplication subprotocol Π_{PrivMult}, which "privately" computes Functionality 4.1 in the sense of Definition 4.2; in practice, we use that of Doerner et al. [DKLs18, § VI. D.] (see Theorem 4.3).

Our protocol, roughly, is a recursive variant of [LNR18, Prot. 4.7], which repetitively performs *appropriate* parts of that protocol in a tree-like manner. In particular, its lines 8–10 below correspond to [LNR18, Prot. 4.7 (1) – (2) (a)], and are performed once for each adjacent pair of shared elements in each layer of the tree (the sum of [LNR18, Prot. 4.7 (2) (b)] is done "lazily", and is *implicit* in line 8 of the *next* tree layer). The lines 12–16 correspond to [LNR18, Prot. 4.7 (2) (c) – (4) (a)], and are carried out once for each adjacent pair of tree elements whose *parent node* occupies an even index in *its* layer. The idea of this is that the protocol anticipates the block 8–10 of the next recursive call, which requires full openings $(\langle y_{2i} \rangle_\nu, s_{2i,\nu})$ of each even-indexed ciphertext Y_{2i} (in the last iteration, where $m' = 1$, the protocol must also anticipate the final opening process of lines 24–26). Lines 24–26 correspond to [LNR18, Prot. 4.7 (4) (b) – (5)], and are performed exactly once per tree, at the root. For notational convenience, we assume that m is a power of 2 in the following protocol.

PROTOCOL 4.8 (Π_{IterMult}—commitment-consistent iterated multiplication protocol)**.**
Our protocol involves players P_0 and P_1, an $\mathbb{F}_q$-homomorphic encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$, and a private multiplication subprotocol Π_{PrivMult}.

Setup. Each player P_ν submits $(\texttt{init})$ to $\mathcal{F}_{\mathsf{CheckDH}}$, and stores the response $(\texttt{key}, pk)$ from $\mathcal{F}_{\mathsf{CheckDH}}$.

Commitment. Each player P_ν sends $(Y_{i,\nu})_{i=0}^{m-1}$ to $P_{1-\nu}$ and receives $(Y_{i,1-\nu})_{i=0}^{m-1}$ from $P_{1-\nu}$.

Multiplication. On input $(\langle y_i \rangle_\nu, s_{i,\nu})_{i=0}^{m-1}$, each player P_ν proceeds as follows:

```
1: for i ∈ {0, ..., m − 1} do
2:     submit (prove, Y_{i,ν}; ⟨y_i⟩_ν, s_{i,ν}) to F_zk^EG.
3:     submit (verify, Y_{i,1−ν}) to F_zk^EG.
```

4: **procedure** RecursiveMultiply $\left(\left(\langle y_i\rangle_\nu, s_{i,\nu}, Y_{i,\nu}, Y_{i,1-\nu}\right)_{i=0}^{m-1}\right)$

5: write $m' := m/2$ and allocate the empty length-m' vector $\left(\langle y'_i\rangle_\nu, s'_{i,\nu}, Y'_{i,\nu}, Y'_{i,1-\nu}\right)_{i=0}^{m'-1}$.

6: conduct Π_{PrivMult} on the input $\left(\langle y_{2i}\rangle_\nu, \langle y_{2i+1}\rangle_\nu\right)_{i=0}^{m'-1}$; assign to $\left(\langle y'_i\rangle_\nu\right)_{i=0}^{m'-1}$ the resulting output.

7: **for** $i \in \{0, \ldots, m'-1\}$ **do**

8: sample $r'_i \leftarrow \mathbb{F}_q$ and set $Y'_{i,\nu} := \langle y_{2i}\rangle_\nu \cdot (Y_{2i+1,0} + Y_{2i+1,1}) + \mathsf{Enc}_{pk}(0; r'_i)$.

9: send $Y'_{i,\nu}$ to $P_{1-\nu}$ and submit $\left(\texttt{prove}, Y'_{i,\nu}, Y_{2i+1,0} + Y_{2i+1,1}, Y_{2i,\nu}; \langle y_{2i}\rangle_\nu, r'_i, s_{2i,\nu}\right)$ to $\mathcal{F}_{\mathsf{zk}}^{\mathsf{Prod}}$.

10: receive $Y'_{i,1-\nu}$ from $P_{1-\nu}$ and submit $\left(\texttt{verify}, Y'_{i,1-\nu}, Y_{2i+1,0} + Y_{2i+1,1}, Y_{2i,1-\nu}\right)$ to $\mathcal{F}_{\mathsf{zk}}^{\mathsf{Prod}}$.

11: **if** i is even **then**

12: temporarily stash the value $Y'_i := Y'_{i,0} + Y'_{i,1}$.

13: sample $s'_{i,\nu} \leftarrow \mathbb{F}_q$ and overwrite $Y'_{i,\nu} := \mathsf{Enc}_{pk}\left(\langle y'_i\rangle_\nu; s'_{i,\nu}\right)$.

14: send the new $Y'_{i,\nu}$ to $P_{1-\nu}$ and submit $\left(\texttt{prove}, Y'_{i,\nu}; \langle y'_i\rangle_\nu, s'_{i,\nu}\right)$ to $\mathcal{F}_{\mathsf{zk}}^{\mathsf{EG}}$.

15: overwrite $Y'_{i,1-\nu}$ with the new value from $P_{1-\nu}$ and submit $(\texttt{verify}, Y'_{i,1-\nu})$ to $\mathcal{F}_{\mathsf{zk}}^{\mathsf{EG}}$.

16: submit $(\texttt{check}, Y'_i - Y'_{i,0} - Y'_{i,1})$ to $\mathcal{F}_{\mathsf{CheckDH}}$.

17: **if** $m' > 1$ **then return** RecursiveMultiply $\left(\left(\langle y'_i\rangle_\nu, s'_{i,\nu}, Y'_{i,\nu}, Y'_{i,1-\nu}\right)_{i=0}^{m'-1}\right)$.

18: **else return** $\left(\langle y'_0\rangle_\nu, s'_{0,\nu}, Y'_{0,\nu}, Y'_{0,1-\nu}\right)$.

19: assign $\left(\langle y'_0\rangle_\nu, s'_{0,\nu}, Y'_{0,\nu}, Y'_{0,1-\nu}\right) \leftarrow$ RecursiveMultiply $\left(\left(\langle y_i\rangle_\nu, s_{i,\nu}, Y_{i,\nu}, Y_{i,1-\nu}\right)_{i=0}^{m-1}\right)$.

20: pick a_ν and r_ν randomly from $\mathbb{F}_q$ and encrypt $A_\nu := \mathsf{Enc}_{pk}(a_\nu; r_\nu)$.

21: send A_ν to P_{1-C} and submit $(\texttt{prove}, A_\nu; a_\nu, r_\nu)$ to $\mathcal{F}_{\mathsf{zk}}^{\mathsf{EG}}$.

22: receive $A_{1-\nu}$ from P_{1-C} and submit $(\texttt{verify}, A_{1-\nu})$ to $\mathcal{F}_{\mathsf{zk}}^{\mathsf{EG}}$.

23: assign $\left(\langle y'_0\rangle_\nu, s'_{0,\nu}, Y'_{0,\nu}, Y'_{0,1-\nu}\right) \leftarrow$ RecursiveMultiply $\left((a_\nu, r_\nu, A_\nu, A_{1-\nu}) \parallel \left(\langle y'_0\rangle_\nu, s'_{0,\nu}, Y'_{0,\nu}, Y'_{0,1-\nu}\right)\right)$.

24: send $\langle y'_0\rangle_\nu$ to $P_{1-\nu}$ and submit $\left(\texttt{prove}, pk, Y'_{0,\nu} - \mathsf{Enc}_{pk}(\langle y'_0\rangle_\nu; 0); s'_{0,\nu}\right)$ to $\mathcal{F}_{\mathsf{zk}}^{\mathsf{DH}}$.

25: receive $\langle y'_0\rangle_{1-\nu}$ from $P_{1-\nu}$ and submit $\left(\texttt{verify}, pk, Y'_{0,1-\nu} - \mathsf{Enc}_{pk}(\langle y'_0\rangle_{1-\nu}; 0)\right)$ to $\mathcal{F}_{\mathsf{zk}}^{\mathsf{DH}}$.

26: output $\langle y'_0\rangle_0 + \langle y'_0\rangle_1 \pmod q$.

Theorem 4.9. *If Π_{PrivMult} satisfies Definition 4.2 and $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ has indistinguishable multiple encryptions, then Protocol 4.8 securely computes Functionality 4.7 in the $\left(\mathcal{F}_{\mathsf{zk}}^{\mathsf{DH}}, \mathcal{F}_{\mathsf{zk}}^{\mathsf{EG}}, \mathcal{F}_{\mathsf{CheckDH}}\right)$-hybrid model.*

Proof. We define an appropriate simulator. As a space-saving device, we stipulate throughout that, upon the failure of any of its checks, $\mathcal{S}$ immediately sends $(\texttt{abort})$ to $\mathcal{F}_f$, outputs what $\mathcal{A}$ outputs, and halts.

On the input $(1^\lambda, C)$, $\mathcal{S}$ operates as follows:

1. When $\mathcal{A}$ sends $(\texttt{init})$ to $\mathcal{F}_{\mathsf{CheckDH}}$, $\mathcal{S}$ forwards $(\texttt{init})$ to $\mathcal{F}_{\mathsf{IterMult}}$. When $\mathcal{S}$ receives $(\texttt{key}, pk)$ from $\mathcal{F}_{\mathsf{IterMult}}$, $\mathcal{S}$ internally sends $\mathcal{A}$ $(\texttt{key}, pk)$, as if from $\mathcal{F}_{\mathsf{CheckDH}}$.
2. Upon receiving $\left(\texttt{commit}, (Y_{i,1-C})_{i=0}^{m-1}\right)$ from $\mathcal{F}_{\mathsf{IterMult}}$, $\mathcal{S}$ forwards the ciphertexts to $\mathcal{A}$, as if from P_{1-C}. When $\mathcal{A}$ sends $(Y_{i,C})_{i=0}^{m-1}$ to P_{1-C}, $\mathcal{S}$ forwards $\left(\texttt{commit}, (Y_{i,C})_{i=0}^{m-1}\right)$ to $\mathcal{F}_{\mathsf{IterMult}}$.
3. $\mathcal{S}$ plays the role of P_{1-C} in the "multiplication" portion of Protocol 4.8; that is, $\mathcal{S}$ runs Algorithm 1.

We invoke a sequence of hybrid distribution families:

D_0: Corresponds to $\mathsf{Ideal}_{\mathcal{F},\mathcal{S},C}$.

D_1: Same as D_0, except that $\mathcal{S}$ is given P_{1-C}'s actual inputs $\left(\langle y_i\rangle_{1-C}, s_{i,1-C}\right)_{i=0}^{m-1}$, and supplies these, instead of $(0)_{i=0}^{m-1}$, in its main top-level invocation of Algorithm 1 in line 21.

D_2: Same as D_1, except that $\mathcal{S}$ skips line 28, and moreover P_{1-C} is not given the output $(\texttt{multiply}, y)$ directly from $\mathcal{F}_{\mathsf{IterMult}}$, but rather is given $\langle y'_0\rangle_0 + \langle y'_0\rangle_1$, as computed from $\mathcal{S}$'s local state at line 32.

D_3: Same as D_2, except that $\mathcal{S}$ instead uses the assignments $Y'_{i,1-C} \leftarrow \langle y_{2i}\rangle_{1-C} \cdot (Y_{2i+1,0} + Y_{2i+1,1}) + \mathsf{Enc}_{pk}(0)$ in line 8, $Y'_{i,1-C} \leftarrow \mathsf{Enc}_{pk}\left(\langle y'_i\rangle_{1-C}\right)$ in line 13, and $A_{1-C} \leftarrow \mathsf{Enc}_{pk}(a_{1-C})$ in line 22.

D_4: Corresponds to $\mathsf{Real}_{\Pi,\mathcal{A},C}$.

Algorithm 1 Simulator for Protocol 4.8

for $i \in \{0, \ldots, m-1\}$ **do**
    when $\mathcal{A}$ submits $(\texttt{verify}, Y_{i,1-C})$ to $\mathcal{F}_{\mathsf{zk}}^{\mathsf{EG}}$, check that the statement $Y_{i,1-C}$ is as received from $\mathcal{F}_{\mathsf{IterMult}}$.
    when $\mathcal{A}$ submits $(\texttt{prove}, Y_{i,C}; \langle y_i \rangle_C, s_{i,C})$ to $\mathcal{F}_{\mathsf{zk}}^{\mathsf{EG}}$, check $Y_{i,C}$ and the relation R_{EG}; store $(\langle y_i \rangle_C, s_{i,C})$.
procedure $\mathsf{RecursiveSimulate}\left(\left(\langle y_i \rangle_C, \langle y_i \rangle_{1-C}, Y_{i,C}, Y_{i,1-C}\right)_{i=0}^{m-1}\right)$
    write $m' := m/2$ and allocate the empty length-m' vector $\left(\langle y'_i \rangle_C, \langle y'_i \rangle_{1-C}, Y'_{i,C}, Y'_{i,1-C}\right)_{i=0}^{m'-1}$.
    engage in Π_{PrivMult} with $\mathcal{A}$ on input $\left(\langle y_{2i} \rangle_{1-C}, \langle y_{2i+1} \rangle_{1-C}\right)_{i=0}^{m'-1}$; assign to $\left(\langle y'_i \rangle_{1-C}\right)_{i=0}^{m'-1}$ the output.
    for $i \in \{0, \ldots, m'-1\}$ **do**
        generate $Y'_{i,1-C} \leftarrow \mathsf{Enc}_{pk}(0)$ as a random encryption of 0; send $Y'_{i,1-C}$ to $\mathcal{A}$.
        when $\mathcal{A}$ submits $\left(\texttt{verify}, Y'_{i,1-C}, Y_{2i+1}, Y_{2i,1-C}\right)$ to $\mathcal{F}_{\mathsf{zk}}^{\mathsf{Prod}}$:

- require that $Y_{2i+1} \stackrel{?}{=} Y_{2i+1,0} + Y_{2i+1,1}$ and $Y_{2i,1-C}$ match the function's passed-in arguments.
- require that the statement element $Y'_{i,1-C}$ matches that just simulated and sent to $\mathcal{A}$.

        when $\mathcal{A}$ sends $Y'_{i,C}$ to P_{1-C} and submits $\left(\texttt{prove}, Y'_{i,C}, Y_{2i+1}, Y_{2i,C}; \langle y_{2i} \rangle_C, r'_i, s_{2i,C}\right)$ to $\mathcal{F}_{\mathsf{zk}}^{\mathsf{Prod}}$:

- require that $Y_{2i+1} \stackrel{?}{=} Y_{2i+1,0} + Y_{2i+1,1}$ and $Y_{2i,C}$ match the function's passed-in arguments.
- require that the statement element $Y'_{i,C}$ matches that which $\mathcal{A}$ just sent separately to P_{1-C}.
- check manually that R_{Prod} holds on $\left(Y'_{i,C}, Y_{2i+1}, Y_{2i,C}; \langle y_{2i} \rangle_C, r'_i, s_{2i,C}\right)$.

        if i is even **then**
            temporarily stash the value $Y'_i := Y'_{i,0} + Y'_{i,1}$.
            randomly encrypt and overwrite $Y'_{i,1-C} \leftarrow \mathsf{Enc}_{pk}(0)$; send the new $Y'_{i,1-C}$ to $\mathcal{A}$.
            when $\mathcal{A}$ submits $\left(\texttt{verify}, Y'_{i,1-C}\right)$ to $\mathcal{F}_{\mathsf{zk}}^{\mathsf{EG}}$, ensure that $Y'_{i,1-C}$ matches that just sent to $\mathcal{A}$.
            when $\mathcal{A}$ sends $Y'_{i,C}$ to P_{1-C} and submits $\left(\texttt{prove}, Y'_{i,C}; \langle y'_i \rangle_C, s'_{i,C}\right)$ to $\mathcal{F}_{\mathsf{zk}}^{\mathsf{EG}}$, check $Y'_{i,C}$ and R_{EG}.
            when $\mathcal{A}$ submits $(\texttt{check}, Y'_i - Y'_{i,0} - Y'_{i,1})$ to $\mathcal{F}_{\mathsf{CheckDH}}$:

- require that $\mathcal{A}$'s submitted statement indeed matches $Y'_i - Y'_{i,0} - Y'_{i,1}$ (as determined locally).
- require that $\mathcal{A}$'s above-extracted $\langle y'_i \rangle_C \stackrel{?}{=} (\langle y_{2i} \rangle_0 + \langle y_{2i} \rangle_1) \cdot (\langle y_{2i+1} \rangle_0 + \langle y_{2i+1} \rangle_1) - \langle y'_i \rangle_{1-C}$.

        else
            store the intermediate value $\langle y'_i \rangle_C := (\langle y_{2i} \rangle_0 + \langle y_{2i} \rangle_1) \cdot (\langle y_{2i+1} \rangle_0 + \langle y_{2i+1} \rangle_1) - \langle y'_i \rangle_{1-C}$.
    if $m' > 1$ **then return** $\mathsf{RecursiveSimulate}\left(\left(\langle y'_i \rangle_C, \langle y'_i \rangle_{1-C}, Y'_{i,C}, Y'_{i,1-C}\right)_{i=0}^{m'-1}\right)$.
    else return $\left(\langle y'_0 \rangle_C, \langle y'_0 \rangle_{1-C}, Y'_{0,C}, Y'_{0,1-C}\right)$.
assign $\left(\langle y'_0 \rangle_C, \langle y'_0 \rangle_{1-C}, s'_{0,C}, Y'_{0,C}, Y'_{0,1-C}\right) \leftarrow \mathsf{RecursiveSimulate}\left(\left(\langle y_i \rangle_C, 0, Y_{i,C}, Y_{i,1-C}\right)_{i=0}^{m-1}\right)$.
$a_{1-C} \leftarrow \mathbb{F}_q$ randomly, but set $A_{1-C} \leftarrow \mathsf{Enc}_{pk}(0)$ as a random encryption of 0; send A_{1-C} to $\mathcal{A}$.
when $\mathcal{A}$ submits $(\texttt{verify}, A_{1-C})$ to $\mathcal{F}_{\mathsf{zk}}^{\mathsf{EG}}$ with the right statement A_{1-C}, respond $(\texttt{verify}, 1)$.
when $\mathcal{A}$ sends A_C to P_{1-C} and submits $(\texttt{prove}, A_C; a_C, r_C)$ to $\mathcal{F}_{\mathsf{zk}}^{\mathsf{EG}}$, ensure A_C matches and R_{EG} holds.
concatenate $\left(\langle y_i \rangle_C, \langle y_i \rangle_{1-C}, Y_{i,C}, Y_{i,1-C}\right)_{i=0}^{1} := (a_C, a_{1-C}, A_C, A_{1-C}) \parallel \left(\langle y'_0 \rangle_C, \langle y'_0 \rangle_{1-C}, Y'_{0,C}, Y'_{0,1-C}\right)$.
overwrite $\left(\langle y'_0 \rangle_C, \langle y'_0 \rangle_{1-C}, Y'_{0,C}, Y'_{0,1-C}\right) \leftarrow \mathsf{RecursiveSimulate}\left(\left(\langle y_i \rangle_C, \langle y_i \rangle_{1-C}, Y_{i,C}, Y_{i,1-C}\right)_{i=0}^{1}\right)$.
send $\mathcal{A}$'s extracted inputs as $\left(\texttt{multiply}, \left(\langle y_i \rangle_C, s_{i,C}\right)_{i=0}^{m-1}\right)$ to $\mathcal{F}_{\mathsf{IterMult}}$; receive the output y from $\mathcal{F}_{\mathsf{IterMult}}$.
using the output $(\texttt{multiply}, y)$ received from $\mathcal{F}_{\mathsf{IterMult}}$, overwrite $\langle y'_0 \rangle_{1-C} := y - \langle y'_0 \rangle_C$.
send $\langle y'_0 \rangle_{1-C}$ to $\mathcal{A}$; if $\mathcal{A}$ aborts, send $(\texttt{abort})$ to $\mathcal{F}_{\mathsf{IterMult}}$.
when $\mathcal{A}$ submits $\left(\texttt{verify}, pk, Y'_{0,1-C} - \mathsf{Enc}_{pk}\left(\langle y'_0 \rangle_{1-C}; 0\right)\right)$ to $\mathcal{F}_{\mathsf{zk}}^{\mathsf{DH}}$:

- require that $\mathcal{A}$'s statement indeed matches $Y'_{0,1-C} - \mathsf{Enc}_{pk}\left(\langle y'_0 \rangle_{1-C}; 0\right)$ (as determined locally).

when $\mathcal{A}$ sends $\langle y'_0 \rangle_C$ to P_{1-C} and submits $\left(\texttt{prove}, pk, Y'_{0,C} - \mathsf{Enc}_{pk}\left(\langle y'_0 \rangle_C; 0\right); s_{0,C}\right)$ to $\mathcal{F}_{\mathsf{zk}}^{\mathsf{DH}}$:

- require that $\mathcal{A}$'s statement indeed matches $Y'_{0,C} - \mathsf{Enc}_{pk}\left(\langle y'_0 \rangle_C; 0\right)$ (as determined locally).
- check manually that $\mathcal{F}_{\mathsf{zk}}^{\mathsf{DH}}$ holds on $\left(pk, Y'_{0,C} - \mathsf{Enc}_{pk}\left(\langle y'_0 \rangle_C; 0\right); s_{0,C}\right)$ (this implies $\langle y'_0 \rangle_C$ matches).

send $(\texttt{continue})$ to $\mathcal{F}_{\mathsf{IterMult}}$ and terminate.

Lemma 4.10. *If the underlying multiplication subprotocol* Π_{PrivMult} *is private in the sense of Definition 4.2, then the distributions* $\{D_0(\mathbf{x}_0, \mathbf{x}_1, \lambda)\}_{\mathbf{x}_0,\mathbf{x}_1,\lambda}$ *and* $\{D_1(\mathbf{x}_0, \mathbf{x}_1, \lambda)\}_{\mathbf{x}_0,\mathbf{x}_1,\lambda}$ *are computationally indistinguishable.*

Proof. $\mathcal{A}$'s views in D_0 and D_1 differ only in the inputs $\mathcal{S}$ supplies to Π_{PrivMult}. This difference affects not just $\mathcal{S}$'s outermost execution of RecursiveSimulate (initiated at line 21), but also the subsequent recursive subcalls, as well as the final execution (initiated at line 26); in these latter calls, $\mathcal{S}$ uses inputs which depend on the outputs of prior calls. In any case, the lemma follows from our hypothesis on Π_{PrivMult}. □

Lemma 4.11. *The distributions* $\{D_1(\mathbf{x}_0, \mathbf{x}_1, \lambda)\}_{\mathbf{x}_0,\mathbf{x}_1,\lambda}$ *and* $\{D_2(\mathbf{x}_0, \mathbf{x}_1, \lambda)\}_{\mathbf{x}_0,\mathbf{x}_1,\lambda}$ *are identical.*

Proof. $\mathcal{A}$'s views in the two distributions are identical until its receipt of $\langle y'_0\rangle_{1-C}$ in line 29; moreover, in both distributions, this latter share differs by $\langle y'_0\rangle_C$ from P_{1-C}'s output. It thus suffices to show that P_{1-C}'s respective outputs y and $\langle y'_0\rangle_0 + \langle y'_1\rangle_1$ in the two distributions are distributed identically, conditioned on $\mathcal{S}$'s reaching line 29. Because a_{1-C} is random, so is a_0+a_1; we conclude that y and $(a_0+a_1)\cdot\prod_{i=0}^{m-1}\left(\langle y_i\rangle_0 + \langle y_i\rangle_1\right)$ follow the same distribution. It thus in turn suffices to show that this latter quantity equals $\langle y'_0\rangle_0 + \langle y'_1\rangle_1$ (as determined on line 32). This latter condition itself captures the correctness of the multiplication protocol, and follows by induction from lines 16 and 18 (the base case comes from line 3 and the definition of D_1). □

Lemma 4.12. *If* $\Pi' = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *has indistinguishable multiple encryptions, then the distribution ensembles* $\{D_2(\mathbf{x}_0, \mathbf{x}_1, \lambda)\}_{\mathbf{x}_0,\mathbf{x}_1,\lambda}$ *and* $\{D_3(\mathbf{x}_0, \mathbf{x}_1, \lambda)\}_{\mathbf{x}_0,\mathbf{x}_1,\lambda}$ *are computationally indistinguishable.*

Proof. We suppose for contradiction that there exists a distinguisher D, a polynomial $p(\Lambda)$, and an infinite collection of triples $(\mathbf{x}_0, \mathbf{x}_1, \lambda)$ for each among which $|\Pr[\mathsf{D}_2(\mathbf{x}_0, \mathbf{x}_1, \lambda) = 1] - \Pr[\mathsf{D}_3(\mathbf{x}_0, \mathbf{x}_1, \lambda) = 1]| \geq \frac{1}{p(\lambda)}$ (strictly speaking, we must insist that infinitely many distinct values λ appear throughout these triples). Without loss of generality—that is, after possibly flipping D's output bit and constraining the set of triples $(\mathbf{x}_0, \mathbf{x}_1, \lambda)$—we may assume that $\Pr[\mathsf{D}_3(\mathbf{x}_0, \mathbf{x}_1, \lambda) = 1] - \Pr[\mathsf{D}_2(\mathbf{x}_0, \mathbf{x}_1, \lambda) = 1] \geq \frac{1}{p(\lambda)}$ for each triple.

We define an adversary $\mathcal{A}'$ attacking the multiple encryptions experiment $\mathsf{PubK}^{\mathsf{LR\text{-}cpa}}_{\Pi',\mathcal{A}'}$ as follows. For each λ for which a triple exists, $\mathcal{A}'$, using the advice $(\mathbf{x}_0, \mathbf{x}_1)$, plays $\mathsf{PubK}^{\mathsf{LR\text{-}cpa}}_{\Pi',\mathcal{A}'}(\lambda)$ in the following way, given pk and access to the oracle $\mathsf{LR}_{pk,b}(\cdot,\cdot)$. $\mathcal{A}'$ runs an instance of $\mathcal{S}$, with the following modifications. Instead of executing 1. above, $\mathcal{A}'$ internally simulates $\mathcal{F}_{\mathsf{CheckDH}}$ giving $\mathcal{A}$ $(\mathtt{key}, pk)$, where pk is the experimenter's public key. Using P_{1-C}'s inputs, $\mathcal{A}'$ computes $Y_{i,1-C} := \mathsf{Enc}_{pk}\left(\langle y_i\rangle_{1-C}, s_{i,1-C}\right)$ for each $i \in \{0, \ldots, m-1\}$, and simulates P_{1-C} sending $\mathcal{A}$ $(Y_{i,1-C})_{i=0}^{m-1}$.

Moreover, $\mathcal{A}'$ applies the following modifications to Algorithm 1. In line 8, $\mathcal{A}'$ generates $Y'_{i,1-C} \leftarrow \mathsf{LR}_{pk,b}\left(0, \langle y_{2i}\rangle_{1-C}\cdot\left(\langle y_{2i+1}\rangle_0 + \langle y_{2i+1}\rangle_1\right)\right)$ using an oracle call. Similarly, in line 13, $\mathcal{A}'$ obtains and overwrites $Y'_{i,1-C} \leftarrow \mathsf{LR}_{pk,b}\left(0, \langle y'_i\rangle_{1-C}\right)$ from the oracle (using the output $\left(\langle y'_i\rangle_{1-C}\right)_{i=0}^{m'-1}$ it obtained from Π_{PrivMult} in line 6). Finally, in line 22, $\mathcal{A}'$ generates $A_{1-C} \leftarrow \mathsf{LR}_{pk,b}(0, a_{1-C})$ using a further oracle call. $\mathcal{A}'$ proceeds otherwise as specified in D_2 and D_3, and runs D on the resulting output. $\mathcal{A}'$ outputs whatever D outputs.

If the experimenter's bit $b = 0$, then $\mathcal{A}$'s view in its simulation by $\mathcal{A}'$ clearly matches its view in D_2;, if $b = 1$, then $\mathcal{A}$'s view matches its view in D_3. Indeed, we note in particular that, in D_3, the distribution of $Y'_{i,1-C} \leftarrow \langle y_{2i}\rangle_{1-C}\cdot(Y_{2i+1,0} + Y_{2i+1,1}) + \mathsf{Enc}_{pk}(0)$ is exactly that of a random encryption of $\langle y_{2i}\rangle_{1-C}\cdot\left(\langle y_{2i+1}\rangle_0 + \langle y_{2i+1}\rangle_1\right)$.

We conclude that:

$$\begin{aligned}
\Pr\left[\mathsf{PubK}^{\mathsf{LR\text{-}cpa}}_{\Pi',\mathcal{A}'}(\lambda) = 1\right] &= \frac{1}{2}\cdot\left(\Pr\left[\mathcal{A}'\left(\mathsf{PubK}^{\mathsf{LR\text{-}cpa}}_{\Pi',\mathcal{A}}(\lambda)\right) = 0 \mid b = 0\right] + \Pr\left[\mathcal{A}'\left(\mathsf{PubK}^{\mathsf{LR\text{-}cpa}}_{\Pi',\mathcal{A}'}(\lambda)\right) = 1 \mid b = 1\right]\right) \\
&= \frac{1}{2}\cdot\left(\Pr[D(\mathsf{D}_2(\mathbf{x}_0, \mathbf{x}_1, \lambda)) = 0] + \Pr[D(\mathsf{D}_3(\mathbf{x}_0, \mathbf{x}_1, \lambda)) = 1]\right) \\
&= \frac{1}{2}\cdot\left(1 - \Pr[D(\mathsf{D}_2(\mathbf{x}_0, \mathbf{x}_1, \lambda)) = 1] + \Pr[D(\mathsf{D}_3(\mathbf{x}_0, \mathbf{x}_1, \lambda)) = 1]\right) \\
&= \frac{1}{2} + \frac{1}{2}\cdot\left(\Pr[D(\mathsf{D}_3(\mathbf{x}_0, \mathbf{x}_1, \lambda)) = 1] - \Pr[D(\mathsf{D}_2(\mathbf{x}_0, \mathbf{x}_1, \lambda)) = 1]\right). \\
&\geq \frac{1}{2} + \frac{1}{2\cdot p(\lambda)},
\end{aligned}$$

where the last step is exactly our hypothesis on D. This inequality, which holds for infinitely many λ, contradicts our assumption that $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ has indistinguishable multiple encryptions. □

Lemma 4.13. *The distributions* $\{D_3(\mathbf{x}_0, \mathbf{x}_1, \lambda)\}_{\mathbf{x}_0, \mathbf{x}_1, \lambda}$ *and* $\{D_4(\mathbf{x}_0, \mathbf{x}_1, \lambda)\}_{\mathbf{x}_0, \mathbf{x}_1, \lambda}$ *are identical.*

Proof. It remains to argue that $\mathcal{S}$'s abort conditions in D_3 match those employed by P_{1-C} and the various functionalities in D_4 (i.e., in the real world). This fact is essentially self-evident, except perhaps at $\mathcal{S}$'s check in line 16. In that line, $\mathcal{S}$ proceeds if and only if $\mathcal{A}$'s extracted message $\langle y'_i \rangle_C$ and $\mathcal{S}$'s output $\langle y'_i \rangle_{1-C}$ from Π_{PrivMult} add to the "correct" product, itself computed using certain quantities memoized from prior executions. We claim that $\mathcal{S}$'s abort condition here is equivalent to $\mathcal{F}_{\mathsf{CheckDH}}$'s. Because $\langle y'_i \rangle_C$ opens the overwritten $Y'_{i,C}$ by definition of R_{EG} (see line 15) and $\langle y'_i \rangle_{1-C}$ opens the overwritten $Y'_{i,1-C}$ by construction (see the definition of D_3), this claim in turn is equivalent to that whereby the message of Y'_i is the product expression $(\langle y_{2i} \rangle_0 + \langle y_{2i} \rangle_1) \cdot (\langle y_{2i+1} \rangle_0 + \langle y_{2i+1} \rangle_1)$. By the construction of the summands $Y'_{i,0}$ and $Y'_{i,1}$ of Y'_i and the definition of R_{Prod}, this fact itself holds so long as $\langle y_{2i+1} \rangle_0 + \langle y_{2i+1} \rangle_1$ is the message of $Y_{2i+1,0} + Y_{2i+1,1}$. This latter fact again holds by an inductive argument. Indeed, we refer to the check 18 of the *prior* recursive call, together with an identical product argument, and the inductive hypothesis. The base case again holds by virtue of the supplied inputs (see line 3 and the definition of D_1). ☐

We complete the proof upon combining Lemmas 4.10, 4.11, 4.12, and 4.13. ☐

Remark 4.14. To omit the re-randomization step of line 1 of $\mathcal{F}_{\mathsf{IterMult}}$, we may simply skip Protocol 4.8's lines 20–23. Similarly, the simulator $\mathcal{S}$ would correspondingly skip lines 22–26 of Algorithm 1.

Remark 4.15. Interestingly, Protocol 4.8 uses the odd-indexed components of the initial randomness vector $(s_{i,\nu})_{i=0}^{m-1}$ *only* as witnesses for $\mathcal{F}_{\mathsf{zk}}^{\mathsf{EG}}$ in the first line, and nowhere else. The body of RecursiveMultiply itself never uses the odd-indexed randomnesses submitted to it, and in fact declines to populate them altogether in the recursive inputs it prepares. This phenomenon owes to the fact that the "product" functionality $\mathcal{F}_{\mathsf{zk}}^{\mathsf{Prod}}$ treats its arguments asymmetrically, and in particular requires the message and randomness only of one of its "multiplicands". Protocol 4.8 could, of course, execute the reconstruction block 12–16 at *every*—and not just every even—index i, but the effort so exerted would be wasted.

Remark 4.16. Actually, *only* the odd-indexed initial inputs $\langle y_i \rangle_0$ and $\langle y_i \rangle_1$ need to be treated in Protocol 4.8's lines 2–3 (see also lines 2–3 of Algorithm 1). Indeed, the even-indexed values appear anyway, in lines 9–10 (see also lines 9–10 of Algorithm 1), where they're submitted as witnesses to R_{Prod}; this relation in particular implies R_{EG}. In fact, by the same reasoning, we may eliminate entirely the $\mathcal{F}_{\mathsf{zk}}^{\mathsf{EG}}$ proofs from lines 21 and 22 of Protocol 4.8; we preserve these above essentially to simplify our exposition.

Remark 4.17. We contrast Lemma 4.13 with the security argument [LNR18, Thm. B.1, Mult., 9. (b)]. There, to simulate $\mathcal{F}_{\mathsf{CheckDH}}$, $\mathcal{S}$ essentially checks (in our notation) whether $\mathcal{A}$'s extracted witness $\langle y'_i \rangle_C$ and $\mathcal{S}$'s output $\langle y'_i \rangle_{1-C}$ from Π_{PrivMult} add to the correct output (as discerned directly from the functionality). Our $\mathcal{S}$ lacks this recourse, as the element to which these quantities "should" add is, in our case, generally (i.e., except in the last execution) some intermediate value unavailable from the functionality. The content of Lemma 4.13, then, is essentially that $\mathcal{S}$ can nonetheless correctly emulate $\mathcal{F}_{\mathsf{CheckDH}}$'s abort behavior on the basis solely of both parties' initial inputs and its own outputs in Π_{PrivMult}. Indeed, having extracted $\mathcal{A}$'s initial inputs $(\langle y_i \rangle_C)_{i=0}^{m-1}$ in line 3 and given P_{1-C}'s inputs $\left(\langle y_i \rangle_{1-C}\right)_{i=0}^{m-1}$, $\mathcal{S}$ can exactly determine the message of each intermediate sum Y'_i. This latter calculation requires a recursive memoization, aided by the induction-preserving step in line 18. In fact, each initial pair $(\langle y_i \rangle_0, \langle y_i \rangle_1)$ can influence as many as $\Theta(\log m)$ memoized expressions before it is used to check some opening $\langle y'_i \rangle_C$ (e.g., consider the case $i = m - 1$).

4.3 Main protocol

We now give our main protocol for Functionality 2.8. We assume that a particular instance of that functionality—that is, a boolean function $f : \{0,1\}^n \to \{0,1\}$—has been fixed.

In order to simplify its exposition, we implement Protocol 4.18's "commitment" consistency using homomorphic *encryption*. We could just as well have used a homomorphic commitment scheme (compare Examples 2.19 and 2.20). Informally, we repurpose our encryption scheme as a perfectly binding commitment scheme. Our encryption-based approach makes Protocol 4.18's compatibility with Zether [BAZB20] and Anonymous Zether [Dia21] somewhat more immediate, though the approaches are philosophically analogous.

PROTOCOL 4.18 (Main protocol)**.**
We fix players P_0 and P_1, an $\mathbb{F}_q$-homomorphic encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$, and a covering $f^{-1}(1) = \bigcup_{i=0}^{m-1} H_i \cap \{0,1\}^n$ using $\mathbb{F}_q$-hyperplanes.

Setup. Each P_ν submits $(\texttt{init})$ to $\mathcal{F}_{\mathsf{IterMult}}$, and stores the response $(\texttt{key}, pk)$.

Commitment. Each party P_ν runs $(pk_\nu, sk_\nu) \leftarrow \mathsf{Gen}(1^\lambda)$ and encrypts $A_\nu \leftarrow \mathsf{Enc}_{pk_\nu}\left(\sum_{i=0}^{n/2-1} 2^i \cdot x_{\nu,i}\right)$. P_ν sends (pk_ν, A_ν) to $P_{1-\nu}$ and receives $(pk_{1-\nu}, A_{1-\nu})$ from $P_{1-\nu}$.

Evaluation. On input $\mathbf{x}_\nu = \left(x_{\nu,0}, \ldots, x_{\nu,n/2-1}\right) \in \{0,1\}^{n/2}$, P_ν executes the following steps:

```
for i ∈ {0, ..., n/2 − 1} do
    randomly additively secret-share x_{ν,i} = ⟨x_{ν,i}⟩_0 + ⟨x_{ν,i}⟩_1 in F_q.
    sample r_{ν,i,ν} ← F_q and encrypt A_{ν,i} := Enc_pk(x_{ν,i}; r_{ν,i,ν}).
    send A_{ν,i} and ⟨x_{ν,i}⟩_{1−ν} to P_{1−ν}, and submit (prove, pk, A_{ν,i}; x_{ν,i}, r_{ν,i,ν}) to F_zk^BitProof.
    locally write r_{ν,i,1−ν} := 0, A_{ν,i,1−ν} := Enc_pk(⟨x_{ν,i}⟩_{1−ν}; 0), and A_{ν,i,ν} := A_{ν,i} − A_{ν,i,1−ν}.
    receive A_{1−ν,i} and ⟨x_{1−ν,i}⟩_ν from P_{1−ν}, and submit (verify, pk, A_{1−ν,i}) to F_zk^BitProof.
    locally write r_{1−ν,i,ν} := 0, A_{1−ν,i,ν} := Enc_pk(⟨x_{1−ν,i}⟩_ν; 0), and A_{1−ν,i,1−ν} := A_{1−ν,i} − A_{1−ν,i,ν}.

submit (prove, pk_ν, pk, A_ν, Σ_{i=0}^{n/2−1} 2^i · A_{ν,i}; x_ν, r_ν, Σ_{i=0}^{n/2−1} 2^i · r_{ν,i,ν}) to F_zk^EqMsg.
submit (verify, pk_{1−ν}, pk, A_{1−ν}, Σ_{i=0}^{n/2−1} 2^i · A_{1−ν,i}) to F_zk^EqMsg.
evaluate the hyperplanes (H_i)_{i=0}^{m−1} on your plaintexts and the opposite party's ciphertexts; i.e., set:
```

$$\left(\langle y_i\rangle_\nu, s_{i,\nu}\right)_{i=0}^{m-1} := \left(H_i\left(\left(\langle x_{0,i}\rangle_\nu, r_{0,i,\nu}\right), \left(\langle x_{1,i}\rangle_\nu, r_{1,i,\nu}\right)_{i=0}^{n/2-1}\right)\right)_{i=0}^{m-1}, \tag{1}$$

$$\left(Y_{i,1-\nu}\right)_{i=0}^{m-1} := \left(H_i\left(\left(A_{0,i,1-\nu}\right), \left(A_{1,i,1-\nu}\right)\right)_{i=0}^{n/2-1}\right)_{i=0}^{m-1}. \tag{2}$$

```
submit (commit, (Enc_pk(⟨y_i⟩_ν; s_{i,ν}))_{i=0}^{m−1}) to F_IterMult.
upon receiving (commit, (Y_{i,1−ν})_{i=0}^{m−1}) from F_IterMult, ensure that the ciphertexts match those of (2).
submit (multiply, (⟨y_i⟩_ν, s_{i,ν})_{i=0}^{m−1}) to F_IterMult, and receive the output y.
output y =? 0.
```

Theorem 4.19. *If* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *has indistinguishable multiple encryptions, then Protocol 4.18 securely computes Functionality 2.8 in the* $\left(\mathcal{F}_{\mathsf{zk}}^{\mathsf{BitProof}}, \mathcal{F}_{\mathsf{zk}}^{\mathsf{EqMsg}}, \mathcal{F}_{\mathsf{IterMult}}\right)$*-hybrid model.*

Proof. We first define an appropriate simulator. We stipulate as before that, before aborting upon a failed check, $\mathcal{S}$ sends $(\texttt{abort})$ to $\mathcal{F}_f$ and outputs what $\mathcal{A}$ outputs.

$\mathcal{S}$ operates as follows, given $(1^\lambda, C, \mathbf{x}_C)$:

1. When $\mathcal{A}$ sends $(\texttt{init})$ to $\mathcal{F}_{\mathsf{IterMult}}$, $\mathcal{S}$ runs $(pk, sk) \leftarrow \mathsf{Gen}(1^\lambda)$, and sends $\mathcal{A}$ $(\texttt{key}, pk)$ as if from $\mathcal{F}_{\mathsf{IterMult}}$.

2. $\mathcal{S}$ generates $(pk_{1-C}, sk_{1-C}) \leftarrow \mathsf{Gen}(1^\lambda)$ and simulates $A_{1-C} \leftarrow \mathsf{Enc}_{pk_{1-C}}(0)$ as a random encryption of 0. $\mathcal{S}$ internally simulates P_{1-C} giving $\mathcal{A}$ (pk_{1-C}, A_{1-C}). $\mathcal{S}$ receives (pk_C, A_C) from $\mathcal{A}$.

3. $\mathcal{S}$ randomly simulates the ciphertexts $A_{1-C,i} \leftarrow \mathsf{Enc}_{pk}(0)$, and samples the "shares" $\langle x_{1-C,i}\rangle_C \leftarrow \mathbb{F}_q$ randomly; $\mathcal{S}$ sends these internally to $\mathcal{A}$. To each message $(\texttt{verify}, pk, A_{1-C,i})$, $\mathcal{S}$ responds $(\texttt{verify}, 1)$. Upon $\mathcal{A}$'s sending $A_{C,i}$ and $\langle x_{C,i}\rangle_{1-C}$ to P_{1-C} and $(\texttt{prove}, pk, A_{C,i}; x_{C,i}, r_{C,i,C})$ to $\mathcal{F}_{\mathsf{zk}}^{\mathsf{BitProof}}$, $\mathcal{S}$ ensures that the ciphertexts $A_{C,i}$ match, and that $R_{\mathsf{BitProof}}\left(pk, A_{C,i}; x_{C,i}, r_{C,i,C}\right)$ (for each $i \in \left\{0, \ldots, \frac{n}{2} - 1\right\}$).

4. Upon $\mathcal{A}$'s message $\left(\texttt{verify}, pk_{1-C}, pk, A_{1-C}, \sum_{i=0}^{n/2-1} 2^i \cdot A_{1-C,i}\right)$ to $\mathcal{F}_{\mathsf{zk}}^{\mathsf{EqMsg}}$, $\mathcal{S}$ ensures that the statement matches the appropriate previously received or simulated quantities, and responds $(\texttt{verify}, 1)$.

Upon $\mathcal{A}$'s message $\left(\texttt{prove}, pk_C, pk, A_C, \sum_{i=0}^{n/2-1} 2^i \cdot A_{C,i}; x_C, r_C, \sum_{i=0}^{n/2-1} 2^i \cdot r_{C,i,C}\right)$ to $\mathcal{F}_{\mathsf{zk}}^{\mathsf{EqMsg}}$, $\mathcal{S}$ ensures that its statement matches all prior quantities, and checks that the relation R_{EqMsg} holds.

5. Using the values $A_{C,i}$ and $\langle x_{C,i}\rangle_{1-C}$ $\mathcal{A}$ sent and the values $A_{1-C,i}$ and $\langle x_{1-C,i}\rangle_C$ $\mathcal{S}$ simulated, $\mathcal{S}$, rederives each ciphertext $A_{C,i,C} := A_{C,i} - \mathsf{Enc}_{pk}\left(\langle x_{C,i}\rangle_{1-C}; 0\right)$ and $A_{1-C,i,C} := \mathsf{Enc}_{pk}\left(\langle x_{1-C,i}\rangle_C; 0\right)$ (as P_{1-C} would) and $A_{1-C,i,1-C} := A_{1-C,i} - \mathsf{Enc}_{pk}\left(\langle x_{1-C,i}\rangle_C; 0\right)$ and $A_{C,i,1-C} := \mathsf{Enc}_{pk}\left(\langle x_{C,i}\rangle_{1-C}; 0\right)$ (as P_C would). Using these and (2), $\mathcal{S}$ manually recomputes the ciphertexts $(Y_{i,C})_{i=0}^{m-1}$ and $(Y_{i,1-C})_{i=0}^{m-1}$.

6. $\mathcal{S}$ sends $\mathcal{A}$ $\left(\texttt{commit}, (Y_{i,1-C})_{i=0}^{m-1}\right)$, as if from $\mathcal{F}_{\mathsf{IterMult}}$. When $\mathcal{A}$ submits $\left(\texttt{commit}, (Y_{i,C})_{i=0}^{m-1}\right)$ to $\mathcal{F}_{\mathsf{IterMult}}$, $\mathcal{S}$ ensures that the ciphertexts in $\mathcal{A}$'s message match those which $\mathcal{S}$ just computed above.

7. When $\mathcal{A}$ submits $\left(\texttt{multiply}, (\langle y_i\rangle_C, s_{i,C})_{i=0}^{m-1}\right)$ to $\mathcal{F}_{\mathsf{IterMult}}$, $\mathcal{S}$ ensures that $Y_{i,C} \stackrel{?}{=} \mathsf{Enc}_{pk}(\langle y_i\rangle_C; s_{i,C})$ for each $i \in \{0, \ldots, m-1\}$. If if this check fails, $\mathcal{S}$ sends $(\texttt{multiply-abort})$ to $\mathcal{A}$ and halts.

8. $\mathcal{S}$ submits $(\texttt{commit}, \mathbf{x}'_C)$ and $(\texttt{evaluate})$ to $\mathcal{F}_f$, where $\mathbf{x}'_C := \left(x_{C,0}, \ldots, x_{C,n/2-1}\right)$; $\mathcal{S}$ receives $(\texttt{evaluate}, v)$, where $v \in \{0,1\}$. $\mathcal{S}$ sets $y \leftarrow \mathbb{F}_q$ or $y := 0$ accordingly as $v = 0$ or $v = 1$, and simulates $\mathcal{F}_{\mathsf{IterMult}}$ giving $\mathcal{A}$ $(\texttt{multiply}, y)$. If $\mathcal{A}$ sends $(\texttt{abort})$ to $\mathcal{F}_{\mathsf{IterMult}}$, then $\mathcal{S}$ sends $(\texttt{abort})$ to $\mathcal{F}_f$. Otherwise, $\mathcal{S}$ sends $(\texttt{continue})$ to $\mathcal{F}_f$, who releases v to P_{1-C}. $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs.

We prove the theorem by means of a sequence of hybrid distribution families.

D_0: Corresponds to $\mathsf{Ideal}_{\mathcal{F},\mathcal{S},C}$.

D_1: Same as D_0, except $\mathcal{S}$ is given P_{1-C}'s input $\mathbf{x}_{1-C}$, and the ideal P_{1-C}'s output is determined not using $(\texttt{evaluate}, v)$ from the functionality, but rather by $\mathcal{S}$, who, in step 8. above, interleaves $\mathbf{x}'_C$ and $\mathbf{x}_{1-C}$ to obtain $v := f(\mathbf{x})$, assigns $y \leftarrow \mathbb{F}_q$ or $y := 0$ accordingly as $v = 0$ or $v = 1$, and gives P_{1-C} $y \stackrel{?}{=} 0$.

D_2: Same as D_1, except $\mathcal{S}$, using P_{1-C}'s actual input $\mathbf{x}_{1-C} = \left(x_{1-C,0}, \ldots, x_{1-C,n/2-1}\right)$, sets $A_{1-C,i} \leftarrow \mathsf{Enc}_{pk}(x_{1-C,i})$ for each $i \in \left\{0, \ldots, \frac{n}{2} - 1\right\}$ in step 3.

D_3: Same as D_2, except $\mathcal{S}$ moreover sets $A_{1-C} \leftarrow \mathsf{Enc}_{pk_{1-C}}\left(\sum_{i=0}^{n/2-1} 2^i \cdot x_{1-C,i}\right)$ in step 2. above.

D_4: Corresponds to $\mathsf{Real}_{\Pi,\mathcal{A},C}$.

Lemma 4.20. *The distribution ensembles* $\{D_0(\mathbf{x}_0, \mathbf{x}_1, \lambda)\}_{\mathbf{x}_0,\mathbf{x}_1,\lambda}$ *and* $\{D_1(\mathbf{x}_0, \mathbf{x}_1, \lambda)\}_{\mathbf{x}_0,\mathbf{x}_1,\lambda}$ *are statistically indistinguishable.*

Proof. These distributions differ only in how P_{1-C}'s output is determined; it's v in D_0 and $y \stackrel{?}{=} 0$ in D_1. These quantities in turn differ only if $v = 0$ but $\mathcal{S}$ draws the unlucky sample $y = 0$ randomly from $\mathbb{F}_q$. More formally, for each $v \in \{0,1\}$, the difference $\left|\Pr\left[P_{1-C}\left(\mathsf{D}_0(\mathbf{x}_0, \mathbf{x}_1, \lambda)\right) = v\right] - \left[P_{1-C}\left(\mathsf{D}_1(\mathbf{x}_0, \mathbf{x}_1, \lambda)\right) = v\right]\right|$ is at most $\frac{1}{q} \in O\left(\frac{1}{2^\lambda}\right)$, which is negligible. ☐

Lemma 4.21. *If* $\Pi' = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *has indistinguishable multiple encryptions, then the distribution ensembles* $\{D_1(\mathbf{x}_0, \mathbf{x}_1, \lambda)\}_{\mathbf{x}_0,\mathbf{x}_1,\lambda}$ *and* $\{D_2(\mathbf{x}_0, \mathbf{x}_1, \lambda)\}_{\mathbf{x}_0,\mathbf{x}_1,\lambda}$ *are computationally indistinguishable.*

Proof. We fix as before a distinguisher D, a polynomial $p(\Lambda)$, and an infinite collection of triples $(\mathbf{x}_0, \mathbf{x}_1, \lambda)$ for which $\left|\Pr\left[\mathsf{D}_1(\mathbf{x}_0, \mathbf{x}_1, \lambda) = 1\right] - \Pr\left[\mathsf{D}_2(\mathbf{x}_0, \mathbf{x}_1, \lambda) = 1\right]\right| \geq \frac{1}{p(\lambda)}$; we again assume without loss of generality that $\Pr\left[\mathsf{D}_2(\mathbf{x}_0, \mathbf{x}_1, \lambda) = 1\right] - \Pr\left[\mathsf{D}_1(\mathbf{x}_0, \mathbf{x}_1, \lambda) = 1\right] \geq \frac{1}{p(\lambda)}$ holds for each triple.

We again define an adversary $\mathcal{A}'$ attacking the multiple encryptions experiment $\mathsf{PubK}_{\Pi',\mathcal{A}'}^{\mathsf{LR\text{-}cpa}}$. For each λ for which a triple exists, $\mathcal{A}'$, on the advice $(\mathbf{x}_0, \mathbf{x}_1)$ and given pk and access to the oracle $\mathsf{LR}_{pk,b}(\cdot,\cdot)$, $\mathcal{A}'$ initiates the following variant of $\mathcal{S}$. In step 1. above, $\mathcal{A}'$ simulates the message $(\texttt{key}, pk)$ to $\mathcal{A}$ as if from $\mathcal{F}_{\mathsf{IterMult}}$, using the experimenter's public key. In step 2., $\mathcal{S}$ sets $(pk_{1-C}, sk_{1-C}) \leftarrow \mathsf{Gen}(1^\lambda)$ and $A_{1-C} \leftarrow \mathsf{Enc}_{pk_{1-C}}(0)$ as usual. In step 3., $\mathcal{A}'$ constructs $A_{1-C,i} \leftarrow \mathsf{LR}_{pk,b}\left(0, x_{1-C,i}\right)$ using an oracle call, for each $i \in \left\{0, \ldots, \frac{n}{2} - 1\right\}$. $\mathcal{A}'$ proceeds otherwise as in D_1 and D_2, and runs D on the resulting output. $\mathcal{A}'$ outputs whatever D outputs.

If the experimenter's bit $b = 0$ or $b = 1$, then the joint distribution of $\mathcal{A}$'s and P_{1-C}'s outputs—that is, the distribution of D's input—exactly matches D_1 or D_2, respectively. We conclude as before that:

$$\begin{aligned}\Pr\left[\mathsf{PubK}^{\mathsf{LR\text{-}cpa}}_{\Pi',\mathcal{A}'}(\lambda)=1\right] &= \frac{1}{2}\cdot\left(\Pr\left[\mathcal{A}'\left(\mathsf{PubK}^{\mathsf{LR\text{-}cpa}}_{\Pi',\mathcal{A}}(\lambda)\right)=0\mid b=0\right]+\Pr\left[\mathcal{A}'\left(\mathsf{PubK}^{\mathsf{LR\text{-}cpa}}_{\Pi',\mathcal{A}'}(\lambda)\right)=1\mid b=1\right]\right)\\ &= \frac{1}{2}\cdot\left(\Pr[D(\mathsf{D}_1(\mathbf{x}_0,\mathbf{x}_1,\lambda))=0]+\Pr\left[D(\mathsf{D}_2(\mathbf{x}_0,\mathbf{x}_1,\lambda))=1\right]\right)\\ &= \frac{1}{2}\cdot\left(1-\Pr[D(\mathsf{D}_1(\mathbf{x}_0,\mathbf{x}_1,\lambda))=1]+\Pr\left[D(\mathsf{D}_2(\mathbf{x}_0,\mathbf{x}_1,\lambda))=1\right]\right)\\ &= \frac{1}{2}+\frac{1}{2}\cdot\left(\Pr\left[D(\mathsf{D}_2(\mathbf{x}_0,\mathbf{x}_1,\lambda))=1\right]-\Pr[D(\mathsf{D}_1(\mathbf{x}_0,\mathbf{x}_1,\lambda))=1]\right).\\ &\geq \frac{1}{2}+\frac{1}{2\cdot p(\lambda)},\end{aligned}$$

where the last step is our hypothesis on D. This again contradicts our assumption that $(\mathsf{Gen},\mathsf{Enc},\mathsf{Dec})$ has indistinguishable multiple encryptions. ☐

Lemma 4.22. *If $\Pi' = (\mathsf{Gen},\mathsf{Enc},\mathsf{Dec})$ has indistinguishable multiple encryptions, then the distribution ensembles $\{\mathsf{D}_2(\mathbf{x}_0,\mathbf{x}_1,\lambda)\}_{\mathbf{x}_0,\mathbf{x}_1,\lambda}$ and $\{\mathsf{D}_3(\mathbf{x}_0,\mathbf{x}_1,\lambda)\}_{\mathbf{x}_0,\mathbf{x}_1,\lambda}$ are computationally indistinguishable.*

Proof. We define an adversary $\mathcal{A}'$ attacking $\mathsf{PubK}^{\mathsf{LR\text{-}cpa}}_{\Pi',\mathcal{A}'}$ as above; this lemma is essentially the same as Lemma 4.21, but applied to P_{1-C}'s initial ciphertext A_{1-C} in step 2. In this reduction, $\mathcal{A}'$ generates $(pk, sk) \leftarrow \mathsf{Gen}(1^\lambda)$ in step 1. in the usual way, and internally sends $\mathcal{A}$ $(\mathtt{key}, pk)$ as if from $\mathcal{F}_{\mathsf{IterMult}}$. $\mathcal{A}'$ uses the *experimenter's* public key as pk_{1-C} in step 2., generates $A_{1-C} \leftarrow \mathsf{LR}_{pk_{1-C},b}\left(0, \sum_{i=0}^{n/2-1} 2^i \cdot x_{1-C,i}\right)$ using an oracle call, and internally gives $\mathcal{A}$ (pk_{1-C}, A_{1-C}) as if from P_{1-C}. Elsewhere, $\mathcal{A}'$ proceeds as in D_2 and D_3. $\mathcal{A}'$ runs the distinguisher D on the resulting output, and returns whatever D does. If the experimenter's bit b equals 0 or 1, then D's input is distributed exactly as D_2 and D_3, respectively; the lemma follows exactly as Lemma 4.21. ☐

Lemma 4.23. *The distributions $\{\mathsf{D}_3(\mathbf{x}_0,\mathbf{x}_1,\lambda)\}_{\mathbf{x}_0,\mathbf{x}_1,\lambda}$ and $\{\mathsf{D}_4(\mathbf{x}_0,\mathbf{x}_1,\lambda)\}_{\mathbf{x}_0,\mathbf{x}_1,\lambda}$ are identical.*

Proof. These distributions "differ" only in that P_{1-C}'s output is determined using $v := f(\mathbf{x})$ in D_3 and by $\mathcal{F}_{\mathsf{IterMult}}$ in D_4 (i.e., in the real world). This lemma captures the correctness of the protocol, and follows from the condition $f^{-1}(1) = \bigcup_{i=0}^{m-1} H_i \cap \{0,1\}^n$. Indeed, $\mathcal{S}$'s input $\mathbf{x}$ and $\mathcal{F}_{\mathsf{IterMult}}$'s inputs $(\langle y_i\rangle_0, \langle y_i\rangle_1)_{i=0}^{m-1}$ are related by the hyperplane expressions (1) in any successful run of the protocol. By the hypothesis $f^{-1}(1) = \bigcup_{i=0}^{m-1} H_i \cap \{0,1\}^n$, $v = 1$ if and only if $\langle y_{i^*}\rangle_0 + \langle y_{i^*}\rangle_1 = 0$ for some $i^* \in \{0,\ldots,m-1\}$. It follows that $\mathcal{S}$'s simulated output distribution and $\mathcal{F}_{\mathsf{IterMult}}$'s real-world output distribution are identical. ☐

Combining Lemmas 4.20, 4.21, 4.22, and 4.23, we conclude the proof of the theorem. ☐

Remark 4.24. The steps 12 and 13 essentially facilitate formal compliance with the interface of $\mathcal{F}_{\mathsf{IterMult}}$, and can be omitted in a real-life implementation of Protocol 4.18. More concretely, the "commitment" phase of Protocol 4.8—as well as that protocol's lines 2–3—can be omitted *when* Protocol 4.8 serves as a subprotocol within Protocol 4.18. As proving this fact would require breaking our abstractions, we simply note it here informally. Indeed, each party P_ν computes the opposite party's ciphertexts $(Y_{i,1-\nu})_{i=0}^{m-1}$ in (2). If, instead of exchanging and mutually validating these ciphertexts, the parties simply proceeded with Protocol 4.8, then any discrepancy would necessarily emerge—and induce an abort—in lines 9 and 10 of that protocol. Theoretically speaking, lines 2 and 3 are unnecessary when Protocol 4.8 resides within Protocol 4.18, as the simulator of that latter protocol can independently compute the messages of $\mathcal{A}$'s ciphertexts.

Remark 4.25. As each player P_ν in Protocol 4.18 invokes $\mathcal{F}_{\mathsf{zk}}^{\mathsf{BitProof}}$ $\frac{n}{2}$ times, we could have replaced that latter ideal functionality with a "vectorized" variant; such a functionality can moreover be securely instantiated with $O(\log n)$-sized proofs, using aggregated Bulletproofs [BBB+18, § 4.3]. Because our protocol requires that the $\Theta(n)$ *statements* $A_{0,i}$ and $A_{1,i}$ be exchanged regardless, we have elected not to use Bulletproofs, which are slightly more computationally costly in practice than the standalone bit-proofs of [GK15, Fig. 1].

4.4 Efficiency

In this subsection, we describe the efficiency of Protocols 4.8 and 4.18, both theoretical and concrete. We present a full implementation of both protocols, including all required zero-knowledge proofs (summarized in Subsection 2.6) and the multiplication subprotocol of Doerner et al. [DKLs18, § VI. D.] (see Theorem 4.3). The entire implementation comprises about 2,500 lines of Go code. About 1,000 among these lines constitute the multiplication subprotocol; 500 or so more serve zero-knowledge proofs. Protocols 4.8 and 4.18 occupy the remaining 1,000 lines; among these, the former protocol represents the significant majority. Our implementation is single-threaded; certain parts could in principle be parallelized.

We take $(\mathbb{G}, q, g)$ throughout our implementation to be the `secp256k1` elliptic curve group, defined in [Bro10, § 2.4.1]. The group order q is a 256-bit prime. We use the implementation of that curve in Go's `btcec` package. Throughout, we set $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ to be the El Gamal scheme over $(\mathbb{G}, q, g)$ (see Example 2.6 above).

We benchmarked our protocol by running both players as separate processes on a single 2019 MacBook Pro (with a 2.6 GHz 6-Core Intel Core i7 processor), where *moreover* all traffic was tunneled through a WAN. "Wall time" reflects the time the protocol took over this WAN, whose upload and download speeds were respectively clocked at around 600 Mbps and 200 Mbps (this time can be slightly larger for the player who computes last; we consistently reported the larger time). The "elliptic curve multiplications" column counts the number of curve scalar multiplications each party must compute throughout its executing the protocol. "Bytes sent" refers to the number of bytes which each party must send the other throughout the course of its running the protocol. The parties in fact must send each other different amounts, because of their asymmetric roles in [DKLs18, § VI. D.]. The difference between these amounts ranges from a factor of 10% in the case $m = 8$ to about 30% when $m = 64$; we report the larger quantity in each benchmark. As we work in the two-party setting, we don't report "rounds", but rather the total number of messages sent (by either party to the other). This simplifies the exposition, and also reflects certain simplifications we apply in "commit-then-prove" scenarios (e.g., see Remark 4.6). We don't report the costs of our protocol's setup and key-generation phases, as these are identical to those of [DKLs18] and [LNR18].

m	EC Multiplications	Bytes Sent	Total Messages	Wall Time
8	397	378 KB	28	1,048 ms
16	651	749 KB	34	1,626 ms
32	1,259	1,491 KB	40	2,267 ms
64	2,475	2,972 KB	46	3,706 ms
asymp.	$\Theta(m)$	$\Theta(m)$	$\Theta(\log m)$	$\Theta(m)$

Table 1: Costs of Protocol 4.8, for different m.

Our benchmarks for Protocol 4.18 specialize that latter protocol to the comparator function of Example 3.19. We note that that function—which compares two $\frac{n}{2}$-bit integers—can be covered using $m = \frac{n}{2}$ hyperplanes; we recall finally that these can be evaluated in $O(n)$ total time. We note that the majority of the complexity of Protocol 4.18, in most measures, comes from its multiplication portion (namely, Protocol 4.8).

n	EC Multiplications	Bytes Sent	Total Messages	Wall Time
16	550	380 KB	30	1,261 ms
32	1,038	754 KB	36	1,641 ms
64	2,014	1,501 KB	42	2,442 ms
128	3,966	2,991 KB	48	3,945 ms
asymp.	$\Theta(n)$	$\Theta(n)$	$\Theta(\log n)$	$\Theta(n)$

Table 2: Costs of Protocol 4.18 (specialized to the function of Example 3.19), for different n.

References

[AF93] Noga Alon and Zoltán Füredi. Covering the cube by affine hyperplanes. *European Journal of Combinatorics*, 14(2):79–83, 1993.

[AGG+21] James Aaronson, Carla Groenland, Andrzej Grzesik, Tom Johnston, and Bartłomiej Kielak. Exact hyperplane covers for subsets of the hypercube. *Discrete Mathematics*, 344(9), 2021.

[BAZB20] Benedikt Bünz, Shashank Agrawal, Mahdi Zamani, and Dan Boneh. Zether: Towards privacy in a smart contract world. In *International Conference on Financial Cryptography and Data Security*, 2020.

[BBB+18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *IEEE Symposium on Security and Privacy*, pages 315–334, 2018.

[BHMSV84] Robert K. Brayton, Gary D. Hachtel, Curtis T. McMullen, and Alberto L. Sangiovanni-Vincentelli. *Logic Minimization Algorithms for VLSI Synthesis*. The Kluwer International Series in Engineering and Computer Science. Kluwer Academic Publishers, 1984.

[Bro10] Daniel R. L. Brown. SEC 2: Recommended elliptic curve domain parameters. Technical report, Standards for Efficient Cryptography, 2010. Version 2.0.

[BSCG+14] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from Bitcoin. In *IEEE Symposium on Security and Privacy*, pages 459–474, 2014. Full version.

[CDL16] Jan Camenisch, Manu Drijvers, and Anja Lehmann. Anonymous attestation using the strong Diffie Hellman assumption revisited. In Michael Franz and Panos Papadimitratos, editors, *Trust and Trustworthy Computing*, pages 1–20, Cham, 2016. Springer International Publishing.

[CHLL97] Gérard Cohen, Iiro Honkala, Simon Litsyn, and Antoine Lobstein. *Covering Codes*, volume 54 of *North-Holland Mathematical Library*. North-Holland, 1997.

[CLS86] G. Cohen, A. Lobstein, and N. Sloane. Further results on the covering radius of codes. *IEEE Transactions on Information Theory*, 32(5):680–694, 1986.

[Coh82] P.M. Cohn. *Algebra*, volume 1. John Wiley & Sons, second edition, 1982.

[Dia21] Benjamin E. Diamond. Many-out-of-many proofs and applications to Anonymous Zether. In *IEEE Symposium on Security and Privacy*, pages 1800–1817, 2021.

[DKLs18] Jack Doerner, Yashvanth Kondi, Eysa Lee, and abhi shelat. Secure two-party threshold ECDSA from ECDSA assumptions. In *IEEE Symposium on Security and Privacy*, pages 980–997, 2018.

[Doš05] Tomislav Došlić. Maximum product over partitions into distinct parts. *Journal of Integer Sequences*, 8, 2005.

[FMMO19] Prastudy Fauzi, Sarah Meiklejohn, Rebekah Mercer, and Claudio Orlandi. Quisquis: A new design for anonymous cryptocurrencies. In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology – ASIACRYPT 2019*, pages 649–678. Springer International Publishing, 2019.

[FPY18] Tore K. Frederiksen, Benny Pinkas, and Avishay Yanai. Committed MPC. In Michel Abdalla and Ricardo Dahab, editors, *Public-Key Cryptography – PKC 2018*, pages 587–619, Cham, 2018. Springer International Publishing.

[GK15] Jens Groth and Markulf Kohlweiss. One-out-of-many proofs: Or how to leak a secret and spend a coin. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015*, volume 9057 of *Lecture Notes in Computer Science*, pages 253–280. Springer Berlin Heidelberg, 2015.

[Gol04] Oded Goldreich. *Foundations of Cryptography*, volume 2. Basic Applications. Cambridge University Press, 2004.

[HL10] Carmit Hazay and Yehuda Lindell. *Efficient Secure Two-Party Protocols*. Information Security and Cryptography. Springer, 2010.

[JS07] Stanisław Jarecki and Vitaly Shmatikov. Efficient two-party secure computation on committed inputs. In Moni Naor, editor, *Advances in Cryptology – EUROCRYPT 2007*, pages 97–114, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[KL21] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography*. CRC Press, third edition, 2021.

[Lin17] Yehuda Lindell. *Tutorials on the Foundations of Cryptography: Dedicated to Oded Goldreich*, chapter How to Simulate It – A Tutorial on the Simulation Proof Technique, pages 277–346. Information Security and Cryptography. Springer International Publishing, 2017.

[LNR18] Yehuda Lindell, Ariel Nof, and Samuel Ranellucci. Fast secure multiparty ECDSA with practical distributed key generation and applications to cryptocurrency custody. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, pages 1837–1854, New York, NY, USA, 2018. Association for Computing Machinery. Full version.

[LR17] Yehuda Lindell and Tal Rabin. Secure two-party computation with fairness—a necessary design principle. In Yael Kalai and Leonid Reyzin, editors, *Theory of Cryptography*, pages 565–580, Cham, 2017. Springer International Publishing.

[MV06] Hugh L. Montgomery and Robert C. Vaughan. *Multiplicative Number Theory: I. Classical Theory*. Number 97 in Cambridge studies in advanced mathematics. Cambridge University Press, 2006.

[MvOV97] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.

[NMt16] Shen Noether, Adam Mackenzie, and the Monero Research Lab. Ring confidential transactions. *Ledger*, 1:1–18, May 2016.

[PS00] David Pointcheval and Jacques Stern. Security arguments for digital signatures and blind signatures. *Journal of Cryptology*, 13(3):361–396, 2000.

[Vol99] Heribert Vollmer. *Introduction To Circuit Complexity: A Uniform Approach*. Texts in Theoretical Computer Science. Springer-Verlag, 1999.

[vzGG13] Joachim von zur Gathen and Jürgen Gerhard. *Modern Computer Algebra*. Cambridge University Press, third edition, 2013.

[Weg87] Ingo Wegener. *The Complexity of Boolean Functions*. Wiley–Teubner Series in Computer Science. Wiley, 1987.

[WGC19] Sameer Wagh, Divya Gupta, and Nishanth Chandran. SecureNN: 3-party secure computation for neural network training. *Proceedings on Privacy Enhancing Technologies*, (3):26–49, 2019.